Universidad de Salamanca

Departamento de Informática y Automática

Facultad de Ciencias

PROYECTO DOCENTE E INVESTIGADOR

Perfil

INGENIERÍA DEL SOFTWARE

ÁREA DE CONOCIMIENTO
CIENCIA DE LA COMPUTACIÓN E
INTELIGENCIA ARTIFICIAL

Francisco José García Peñalvo

Salamanca, Mayo de 2002

Datos de la convocatoria

Clave	2001DFCAC7
Código de la plaza	G062/D16208
Fecha de la convocatoria	28 de septiembre de 2001
Fecha de publicación en B.O.E	19 de octubre de 2001
Cuerpo docente	Profesores Titulares de Universidad
Área de conocimiento	Ciencia de la Computación e Inteligencia Artificial
Departamento	Informática y Automática
Centro	Facultad de Ciencias
Universidad	Universidad de Salamanca
Actividad a realizar	Ingeniería del Software

Cita recomendada:

García-Peñalvo, F. J. (2002). Proyecto Docente e Investigador Profesor Titular de Universidad. Perfil Ingeniería del Software. Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Salamanca, España: Departamento de Informática y Automática. Universidad de Salamanca. doi:10.5281/zenodo.1018343

Agradecimientos

A mi mujer por su apoyo incondicional.

A mis compañeros del Departamento de Informática y Automática, por tantos buenos momentos.

Capí tulo 1
Introducción

En el primer capítulo de la presente memoria de Proyecto Docente e Investigador se van a introducir los datos concretos de la convocatoria. Basándose en ellos se presentan los principales apartados que van a seguir a continuación justificando, tanto su presencia como el orden en el que aparecen.

1.1 Datos de la convocatoria

La presente memoria constituye el Proyecto Docente e Investigador que se ajusta a la plaza de Profesor Titular de Universidad, con código G062/D16208, convocada por resolución rectoral de la Universidad de Salamanca el 28 de septiembre de 2001 y publicada en el Boletín Oficial del Estado con fecha 19 de octubre de 2001 [BOE, 2001]. Esta plaza se adscribe al Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial que forma parte, en esta Universidad, del Departamento de Informática y Automática.

El perfil de la plaza establece que la actividad a realizar consiste en la docencia e investigación en materia de Ingeniería del Software dentro de las titulaciones de **Ingeniería en Informática de Sistemas** y de **Ingeniería Informática** de la Universidad de Salamanca.

1.2 Estructura de la memoria

La normativa que regula en la actualidad los concursos para la provisión de plazas en los Cuerpos Docentes Universitarios queda recogida en el Título Quinto, Artículo 37 de la Ley Orgánica 11/1983 de la Reforma Universitaria (L.R.U.) [BOE, 1983; MEC, 1983; USAL, 1997] de 26 de julio de 1983 (B.O.E de 25 de agosto de 1983). Por otra parte, el Real Decreto 1888/1984, de 26 de septiembre de 1984 (B.O.E. de 26 de octubre de 1984 [BOE, 1984]) desarrolla los mandatos legales contenidos en los correspondientes artículos del citado Título Quinto. Asimismo, la Orden de 28 de diciembre de 1984 (B.O.E. de 16 de enero de 1985 [BOE, 1985]) establece el desarrollo con carácter transitorio de dicho Real Decreto. Por último, el Real Decreto 1427/1986 de 13 de junio de 1986 (B.O.E. de 11 de julio de 1986) modifica parcialmente el Real Decreto 1888/1984 sobre los concursos para la provisión de plazas de los cuerpos docentes universitarios.

Considerando esta legislación se desarrolla el presente Proyecto Docente e Investigador. No obstante, antes de continuar se desea hacer constar que en la legislación previamente mencionada no se establecen las características concretas, ni la estructura, que debe tener un Proyecto Docente e Investigador.

Sin embargo, se puede encontrar una enriquecedora variedad de ideas sobre la estructura que debe satisfacer un Proyecto Docente e Investigador, aportadas por los distintos autores que han concursado con anterioridad a una plaza de un cuerpo docente universitario. Concretamente, la estructura que toma esta memoria se basa, aunque no de forma exclusiva, en las ideas sintetizadas de los siguientes Proyectos Docentes presentados en otras tantas oposiciones a plazas de los cuerpos docentes universitarios [Vivaracho, 1996; Zazo, 1996; Marqués, 1998; Maudes, 1998; Rodríguez, 1998; Moreno, 1999; Rodríguez, 1999; Corchado, 2000; García, 2000; Molina, 2000; Moreno, 2000a; Moreno, 2000b; Pelechano, 2000; Troyano, 2001; González, 2001; Mandow, 2001; Verdegay, 2001; Curto, 2002; Estívariz, 2002; Fabián, 2002].

Un Proyecto Docente puede definirse como una "*propuesta razonada de un conjunto de acciones a realizar con el fin de alcanzar unos objetivos educativos predeterminados que redundarán en una formación integral del alumno y que, en última instancia, influirá positivamente en la sociedad*" [Verdegay, 2001].

Todo proyecto Docente debe ser:

- Una reflexión sobre lo que el profesor desea hacer con sus alumnos.
- Coherente con las condiciones reales de las personas y de los medios disponibles.
- Bien especificado para poder ser llevado inmediatamente a la práctica.

- Acorde con al talento y aptitudes personales del profesor.
- Abierto a innovaciones y mejoras.

De este modo, la elaboración de un Proyecto Docente implica:

- El establecimiento de unos **objetivos docentes** (qué conocimientos, habilidades y capacitación ha de adquirir el alumno al finalizar la acción educativa).
- La presentación razonada de unos **métodos pedagógicos** (clases magistrales, clases de problemas, prácticas de laboratorio...) y los medios a utilizar (pizarra, cañón de proyección...) para la consecución de los objetivos propuestos.
- La elaboración de un **sistema de evaluación** (exámenes, prácticas, proyectos...) que permitan conocer con la mayor fidelidad posible, tanto el grado en que el alumno ha adquirido dichos conocimientos y habilidades, como la calidad y posible mejora de la enseñanza impartida.

Un Proyecto Docente se realiza sobre una materia que se encuadra en un área de conocimiento. A su vez, esta materia puede comprender varias asignaturas dentro del Plan de Estudios impartido en una determinada Universidad. De esta manera, la elaboración de un Proyecto Docente debe contemplar el ambiente académico en el que se sitúa(n) la(s) asignatura(s) sobre la(s) que se centra dicho Proyecto, entendiéndose de que forma debe(n) situarse la(s) asignatura(s) correctamente en el Plan de Estudios, ajustando los contenidos a la realidad en la que se va(n) a impartir y considerando las dependencias entre asignaturas.

Este planteamiento es consecuencia de la LRU que permite a las Universidades Españolas fijar la estructura de sus propios Planes de Estudio, respetando unos contenidos mínimos comunes para la misma titulación en todas ellas.

En el caso de la materia objeto del perfil de la plaza a concurso, *Ingeniería del Software*, se va a analizar con profundidad dentro de los estudios de la titulación de Ingeniería Técnica en Informática de Sistemas y de la titulación de Ingeniería Informática, que actualmente se imparten en la Universidad de Salamanca, y donde el Departamento de Informática y Automática asume la mayor parte de la carga docente. También se presenta la situación de esta materia en el Programa de Doctorado responsabilidad de este Departamento.

Así pues, en resumen, esta memoria tiene en cuenta la doble vertiente docente e investigadora a realizar por un Profesor Titular de Universidad, para presentar desde una perspectiva global la concepción actual de la Ingeniería del Software, así como la metodología

docente que permita la capacitación de los alumnos para la realización de las tareas profesionales y/o científicas que deberán desarrollar.

El resto de la memoria se organiza según se expone a continuación.

En el Capítulo 2 se analiza el ámbito docente en que se circunscriben las tareas a realizar. En este sentido, se comienza con una reflexión sobre las funciones de la Universidad, institución en la que en primer término se enmarca la plaza objeto de concurso. Se sigue con la exposición de los datos más destacables tanto de la Facultad de Ciencias como del Departamento de Informática y Automática. En él se incluirá información acerca del grupo de personas que lo componen y la docencia que se imparte. El conocimiento de estos aspectos es de indudable interés para un proyecto de trabajo, tanto docente como investigador, en el que se intenta aprovechar todo el potencial de su entorno y realizar una aportación lo más óptima posible al grupo de trabajo existente.

Posteriormente, en este mismo capítulo, se presenta el marco curricular en el que se encuentra situada la plaza, es decir, las titulaciones de Informática. En primer lugar, se realiza un repaso a los antecedentes históricos de los estudios de Informática y a los aspectos de la LRU concernientes a las titulaciones de Informática. Finalmente, se explican los Planes de Estudios, relacionados con las titulaciones de Informática, existentes en la actualidad en la Universidad de Salamanca, analizando los aspectos más destacados de ellos así como las implicaciones que el cambio de Planes tiene tanto en la formación de los alumnos como en el planteamiento de las asignaturas objeto de la plaza.

El Capítulo 3 efectúa una valoración crítica de los distintos métodos de enseñanza, instrumentos pedagógicos y sistemas de evaluación, concluyendo la conveniencia de la utilización de aquéllos que se estiman más adecuados para el ejercicio de la docencia en los estudios de la Ingeniería en Informática. En este capítulo se abordan temas que complementan la labor docente: *la investigación* como fuente necesaria de nuevos conocimientos en un área en constante evolución como es la Informática y *el aseguramiento de la calidad en la docencia* mediante la realización de planes de calidad.

En el Capítulo 4 se define el perfil que rige el presente Proyecto Docente e Investigador, analizando su presencia en la Ingeniería Técnica en Informática de Sistemas y en la Ingeniería Informática, los aspectos formativos de la plaza objeto de concurso y las interrelaciones existentes con otras asignaturas de la misma titulación. Para cerrar este capítulo se hace un estudio del perfil de la plaza tanto a escala internacional, analizando las propuestas curriculares existentes sobre la formación en Informática, como en el ámbito nacional recorriendo diferentes

Planes de Estudios de las Ingenierías Informática (tanto técnicas como superiores) impartidas por Universidades Españolas.

El Capítulo 5 constituye el bloque central del Proyecto Docente. Presenta y justifica los temarios de las asignaturas propias de la disciplina Ingeniería del Software que, dentro de Plan de Estudios vigente, recogen los conceptos teórico-prácticos del perfil de la plaza.

El Capítulo 6 se dedica a la parte de investigación, matiz que confiere a esta memoria un carácter de Proyecto Docente e Investigador. Así, en este capítulo se presentan las futuras líneas de investigación, con las que se pretenden completar el análisis de los objetivos docentes e investigadores necesarios para cubrir las labores exigibles a alguien que va a desarrollar su trabajo en el marco universitario. Concretamente, se introducen dos líneas de investigación que son consecuencia del trabajo realizado por el autor durante los últimos años. Dichas líneas de investigación son: *la reutilización del software basada en un enfoque de líneas de producto* y *el comercio electrónico*. En ambas se intenta conjugar la labor investigadora, en cuanto a su carácter aplicado, y la investigación básica, donde el objetivo es la obtención de modelos teóricos que permitan resolver diferentes problemas.

La memoria se completa con una serie de apéndices. El Apéndice A recoge de forma íntegra el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca. El apéndice B presenta bibliografía propia de la disciplina Ingeniería del Software comentada. Ésta bibliografía sirve en su mayoría como referencias básicas a las asignaturas presentadas en este Proyecto Docente. El apéndice C enumera de diferentes recursos (revistas, congresos, sitios web, grupos de noticias) que pueden ser útiles en el desarrollo de las asignaturas objeto del presente Proyecto Docente. Los Apéndices D y E presentan sendos glosarios, que incluyen las siglas y los términos respectivamente más utilizados en la materia en la que se centra el Proyecto Docente e Investigador.

Cabe reseñar que para la realización del presente Proyecto Docente e Investigador se ha seguido un *proceso iterativo e incremental*, que ha permitido ir elaborando cada uno de los capítulos de forma progresiva, incorporando en cada iteración mayor información, lo que se asemeja a la utilización, en las labores de documentación, del modelo de ciclo de vida en espiral definido por Barry Boehm [Boehm, 1988], pero aplicado a cada capítulo de forma independiente, e incluso, como en el caso del capítulo cinco, aplicado a la descripción detallada de cada uno de los temas de las asignaturas que conforman este informe, hecho que es más acorde con el modelo fuente de ciclo de vida definido por Henderson-Sellers y Edwards [Henderson-Sellers y Edwards, 1990]. Así, se tiene una aplicación de los modelos de ciclo de

vida orientados a objetos en otros contextos diferentes al del desarrollo del software, tal y como se desprende de [Concepcion, 1998].

Por último, señalar que para la elaboración de este Proyecto Docente e Investigador, además de la experiencia adquirida en mi actividad docente, la reflexión y el estudio personal, también han constituido una valiosa aportación las opiniones y apoyo de muchas personas, entre ellas mis compañeros; a ellas, por tanto, manifiesto mi más sincero agradecimiento. No obstante, me gustaría aclarar explícitamente que su exposición en la presente memoria es propia del candidato, quien es el único responsable de los posibles fallos que pueda haber en la misma.

1.3 Referencias

[BOE, 1983] *Boletín Oficial del Estado de 1 de Septiembre de 1983;* Ley Orgánica 11/1983, de 25 de agosto, de reforma universitaria.

[BOE, 1984] *Boletín Oficial del Estado de 26 de Octubre de 1984;* Real Decreto 1888/1984 de 26 de septiembre, por el que se regulan los concursos para la provisión de plazas de los cuerpos docentes universitarios.

[BOE, 1985] *Boletín Oficial del Estado de 16 de Enero de* 1985; Orden de 28 de diciembre de 1984, por la que se establece el desarrollo con carácter transitorio del Real Decreto 1888/1984.

[BOE, 1986] *Boletín Oficial del Estado de 11 de Julio de 1986*; Real Decreto 1427/1986 de 13 de junio, por el que se modifica el Real Decreto 1888/1984 sobre los concursos para la provisión de plazas de los cuerpos docentes universitarios.

[BOE, 2001] *Boletín Oficial del Estado de 19 de Octubre de 2001*; Resolución de 28 de septiembre de 2001 de la Universidad de Salamanca, por la que se convocan a concurso plazas de cuerpos docentes universitarios.

[Boehm, 1988] Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72. May 1988.

[Concepcion, 1998] Concepcion, A. I. "*Using an Object-Oriented Software Life-Cycle Model in the Software Engineering Course*". In Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 30-34. 1998.

[Corchado, 2000] Corchado Rodríguez, J. M. "*Proyecto Docente e Investigador. Inteligencia Artificial e Ingeniería del Conocimiento*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Mayo, 2000.

[Curto, 2002] Curto Diego, B. "*Proyecto Docente e Investigador. Ingeniería de Sistemas*". Área de Conocimiento de Ingeniería de Sistemas y Automática. Facultad de Ciencias. Universidad de Salamanca. 2002.

[Estívariz, 2002] Estívariz López, J. F. "*Proyecto Docente. Ingeniería del Software*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Escuela Técnica Superior de Ingeniería Informática UNED. Febrero, 2002.

[Fabián, 2002] Fabián Caparrós, E. A. "*Proyecto Docente. Derecho Peñal*". Área de Derecho Penal. Facultad de Derecho. Universidad de Salamanca. Marzo, 2002.

[García, 2000] García Peñalvo, F. J. "*Proyecto Docente. Ingeniería del Software y Orientación a Objetos*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Abril, 2000.

[González, 2001] González Arrieta, Mª A. "*Proyecto Docente e Investigador. Algoritmia*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Diciembre, 2001.

[Henderson-Sellers y Edwards, 1990] Henderson-Sellers, B., Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990.

[Madow, 2001] Madow Andaluz, L. "*Proyecto Docente. Sistemas Expertos y Sistemas Inteligentes*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Escuela Técnica Superior de Ingeniería Informática. Universidad de Málaga. Junio, 2001.

[Marqués, 1998] Marqués Corral, J. M. "*Proyecto Docente e Investigador. Ingeniería del Software e Inteligencia Artificial*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Octubre, 1998.

[Maudes, 1998] Maudes Raedo, J. M. "*Proyecto Docente para las Asignaturas: Sistemas de Gestión de Bases de Datos y Administración de Bases de Datos*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Escuela Universitaria Politécnica. Universidad de Burgos. Noviembre, 1998.

[MEC, 1983] Ministerio de Educación y Ciencia. "*Ley Orgánica de Reforma Universitaria*". Servicio de Publicaciones del MEC, 1983.

[Molina, 2000] Molina Marco, A. "*Proyecto Docente. Metodología y Tecnología de la Programación*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Valencia. Marzo, 2000.

[Moreno, 1999] Moreno Rodilla, V. "*Proyecto Docente e Investigador. Control e Instrumentación de Procesos Químicos*". Área de Conocimiento de Ingeniería de Sistemas y Automática. Facultad de Ciencias. Universidad de Salamanca. Junio, 1999.

[Moreno, 2000a] Moreno García, Mª N. "*Proyecto Docente e Investigador. Ingeniería del Software*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Facultad de Ciencias. Universidad de Salamanca. Mayo, 2000.

[Moreno, 2000b] Moreno Montero, Á. Mª. "*Proyecto Docente. Redes de Ordenadores y Bases de Datos*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Facultad de Ciencias. Universidad de Salamanca. Abril, 2000.

[Pelechano, 2000] Pelechano Ferragud, V. "*Proyecto Docente. Ingeniería del Software*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Valencia. Marzo, 2000.

[Rodríguez, 1998] Rodríguez García, N. "*Proyecto Docente. Derecho Procesal*". Área de Conocimiento de Derecho Procesal. Facultad de Derecho. Universidad de Salamanca. 1998.

[Rodríguez, 1999] Rodríguez Rubio, M. "*Proyecto Docente. Análisis de Sistemas Informáticos*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Facultad de Informática. Universidad de A Coruña. Julio, 1999.

[Troyano, 2000] Troyano Jiménez, J. A. "*Proyecto Docente e Investigador. Lenguajes Formales y Autómatas*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Facultad de Informática y Estadística. Universidad de Sevilla. Septiembre, 2000.

[USAL, 1997] Universidad de Salamanca. "*Normativa Universitaria 1997*". Secretaría General de la Universidad de Salamanca. Enero, 1997.

[Verdegay, 2001] Verdegay López, J. F. "*Proyecto Docente. Modelos de Computación*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Escuela Técnica Superior de Ingeniería Informática. Universidad de Granada. 2001.

[Vivaracho, 1996] Vivaracho Pascual, C. E. "*Proyecto Docente. Teoría de la Información y Codificación – Sistemas Operativos*". Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial. Escuela Universitaria Politécnica. Universidad de Valladolid. Julio, 1996.

[Zazo, 1996] Zazo Rodríguez, Á. F. "*Proyecto Docente para el Perfil Teleinformática Aplicada a las Ciencias de la Documentación*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Facultad de Traducción y Documentación. Universidad de Salamanca. Diciembre, 1996.

Capítulo 2
Ámbito Académico

Todo Proyecto Docente debe tener en cuenta las características contextuales en el que se va a desarrollar para poder adaptarse a ellas. Cada Universidad, cada Centro y cada Titulación presenta una serie de particularidades que el docente debe conocer y considerar a la hora de desarrollar su acción formativa. Por ello se muestra a continuación una panorámica de este entorno docente. En primer lugar se presenta el concepto de Universidad, sus objetivos y sus funciones, como marco de todas las actividades que en ella se realizan y, en particular, de la que nos ocupa. A continuación, se concretan las atribuciones y peculiaridades de la Facultad, para posteriormente, descendiendo un peldaño más en la estructura universitaria, comentar las características del Departamento y del Área de Conocimiento. Tras esta revisión de la estructura universitaria se procede a comentar las características y particularidades del alumnado, así como la infraestructura disponible. Con el análisis del marco curricular de las Titulaciones de Ingeniería Técnica en Informática de Sistemas y de Ingeniería en Informática, se cierra esta visión global que define de forma específica el marco educativo en el que se encuadra la labor docente objeto del presente estudio.

2.1 La Universidad

2.1.1 Concepto y misión

Para poder concretar el trabajo dentro de una institución tan compleja como es la Universidad, primero es preciso definir su misión. Así, valorando con posterioridad la realidad actual puede plantearse la necesaria reforma desde lo que es hacia lo que debiera ser.

La Universidad ha desempeñado y desempeña un papel clave en el desarrollo y avance de la sociedad, por tanto debe tener un cuidado especial al llevar a cabo sus actividades. Su finalidad en el ámbito educativo no es sólo proporcionar información, sino conseguir una formación integral de los estudiantes de cara al desempeño eficaz de su futura actividad profesional. Su actividad, así como su autonomía, se fundamentan en el principio de la libertad académica, que se manifiesta en las libertades de cátedra, de investigación y de estudio[1].

A pesar de su larga data, las funciones básicas de la enseñanza universitaria, tal y como se concibe en la actualidad, fueron establecidas por José Ortega y Gasset en su trabajo *Misión de la Universidad* [Ortega, 1982]. Estas funciones son:

- Transmisión de cultura, entendida ésta como sistema de ideas vivas de una época.
- Preparación para el ejercicio profesional.
- Realización de investigación científica y educación de nuevos hombres de ciencia.

Estas funciones de la Universidad propuestas por Ortega y Gasset, no sólo no han perdido vigencia, sino que incluso se encuentran recogidas en la Ley Orgánica 11/1983 de 25 de agosto de Reforma Universitaria [MEC, 1983] y, actualmente, se recogen en la Ley Orgánica de Universidades (L.O.U.), aprobada el 20 de diciembre de 2001 [BOCG, 2001; BOE, 2001], al afirmar en su preámbulo:

> "*No de menor magnitud ha sido la transformación tan positiva en el ámbito de la investigación científica y técnica universitaria, cuyos principios destinatarios son los propios estudiantes de nuestras Universidades, que no sólo reciben en éstas una*

[1] El artículo 2.1 de la extinta LRU [MEC, 1983] establecía que "*la actividad de la Universidad, así como su autonomía, se fundamentan en el principio de libertad académica, que se manifiesta en las libertades de cátedra, de investigación y de estudio*", términos que son adoptados al pie de la letra por el artículo 2.3 de la LOU. Los Estatutos de la Universidad de Salamanca reconocen, en su artículo 139.1.c, el derecho de los estudiantes a "*participar activa y críticamente en las actividades docentes impartidas por los Departamentos, en el marco de la libertad de estudio, así como en su programación y ordenación, y colaborar en las tareas investigadoras*".

formación profesional adecuada, sino que pueden beneficiarse del espíritu crítico y la extensión de la cultura, funciones ineludibles de la institución universitaria".

Así, en el Título preliminar, artículo primero, sobre las funciones de la Universidad se dice textualmente:

1. *La Universidad realiza el servicio público de la educación superior mediante la investigación, la docencia y el estudio.*
2. *Son funciones de la Universidad al Servicio de la sociedad:*
    a. *La creación, desarrollo, transmisión y crítica de la ciencia, de la técnica y de la cultura.*
    b. *La preparación para el ejercicio de actividades profesionales que exijan la aplicación de conocimientos y métodos científicos o para la creación artística.*
    c. *La difusión, la valoración y la transferencia del conocimiento al servicio de la cultura, de la calidad de la vida, y del desarrollo económico.*
    d. *La difusión del conocimiento y la cultura a través de la extensión universitaria y la formación a lo largo de toda la vida.*

Por tanto la Universidad tiene una labor docente y otra investigadora. Ambas actividades se complementan mutuamente: la investigación permite la ampliación de conocimientos y, por consiguiente, la actualización docente. A su vez la docencia hace posible la sedimentación y la difusión de los resultados de la labor investigadora.

Centrando la atención en el caso concreto de la Universidad de Salamanca, los fines particulares, establecidos el artículo segundo de sus Estatutos [BOE, 1988][2], son los siguientes:

- *La ampliación del conocimiento por medio de la investigación en todas las ramas de la cultura, la ciencia y la técnica.*
- *La transmisión crítica del saber a través de la actividad docente y de la discusión científica.*
- *La contribución a la formación y perfeccionamiento de profesionales cualificados.*

[2] Pendientes de la reforma exigida por la propia LOU, los Estatutos de la Universidad de Salamanca fueron aprobados por el Real Decreto 678/1988 de 1 de julio (BOE nº 160, de 5 de julio de 1988). Posteriormente fueron modificados por el Real Decreto 1292/1991 de 2 de agosto (BOE nº 191, de 10 de agosto, rectificado por BOE nº 247, de 15 de octubre de 1991).

- *La promoción, enaltecimiento y difusión de la lengua española.*
- *La contribución a la mejora del sistema educativo.*
- *El fomento y expansión de la cultura, mediante la acogida y estímulo de la actividad intelectual en todos los ámbitos.*
- *El asesoramiento científico, técnico y cultural a la sociedad, dirigido a la satisfacción de sus necesidades.*
- *La cooperación al desarrollo científico y técnico, cultural y social de Castilla y León, de España y de todos los pueblos.*

La enseñanza universitaria no se reduce, por tanto, a la formación técnica del estudiante, sino que debe perseguir el carácter educacional del aprendizaje. Los alumnos deben beneficiarse de una educación que les permita desarrollar al máximo sus aptitudes y su potencial creativo. Como se desprende de sus Estatutos, es objetivo primordial e ineludible de la Universidad de Salamanca lograr que la enseñanza que en ella se imparte alcance el más alto grado de calidad, y tienda a la formación integral y crítica de sus miembros. Promueve, como uno de los fines fundamentales de su actividad, la formación de investigadores, así como el fomento y coordinación de la investigación científica y técnica.

2.1.2 La Universidad y la cultura

El concepto de Cultura de Ortega y Gasset, se entiende repasando someramente la propia historia de la Universidad.

En la Edad Media la Universidad no formaba profesionales ni científicos. Tan sólo era el lugar donde se trasmitía el sistema de ideas sobre el mundo y la humanidad que entonces poseía el hombre, llegando a crearse un repertorio de convicciones que había de dirigir definitivamente la existencia de los hombres que lo adquirían. El universitario trabajaba para conseguir ideas claras y firmes sobre el Universo, convicciones positivas sobre lo que son las cosas y el mundo.

No es hasta finales del siglo XIX, donde por razones de abarcabilidad de conocimientos, cada vez más técnicos y complejos, se acaba imponiendo la especialización o separación entre lo que se denominó cultura o malentendido humanismo y ciencia. Pero según F. Savater "*las facultades que el humanismo pretende desarrollar son la capacidad crítica de análisis, la curiosidad que no respeta dogmas ni ocultamientos, el sentido de razonamiento lógico, la sensibilidad para apreciar las más altas realizaciones del espíritu humano, la visión de conjunto ante el panorama del saber, etc.*" [Savater, 1998], por lo que repitiendo sus palabras:

"*Francamente, no conozco ningún argumento serio para probar que el estudio del latín y el griego favorecen más estas deseables cualidades que el de las matemáticas o la química*" [Savater, 1998] o su consecuencia: el humanismo o cultura no va ligado a la condición práctica o técnica de la disciplina a impartir sino a su servicio a la interpretación del mundo. El hombre no puede vivir sin reaccionar ante su entorno, sin interpretarlo de alguna forma y sin plantearse una conducta a seguir. Esto supone, en palabras de Ortega y Gasset [Ortega, 1982] "*esa terrible faena de sostenerse en el Universo, de conducirse por entre las cosas y seres del mundo*". "*Cultura es el sistema de ideas vivas que cada tiempo posee. El sistema de ideas desde las cuales el tiempo se vive*".

Pero la casi totalidad de estas convicciones no se las fabrica el individuo, sino que las recibe de su medio histórico, de su tiempo. Aunque siempre coexisten sistemas ideológicos muy diferentes, hay uno que representa el nivel superior del tiempo, un sistema plenamente actual, **la Cultura**.

En nuestra época el contenido de la Cultura viene en su mayor parte de la Ciencia. Pero la Cultura hace con la ciencia lo mismo que con la profesión: saca de ellas lo vitalmente necesario para interpretar la existencia. La Cultura necesita poseer una idea completa del hombre y del mundo, y no se detiene como la Ciencia, allí donde terminan los métodos de absoluto rigor científico. Además, como advierte B. Russell [Russell, 1997] "*Una dictadura de hombres de ciencia sería muy pronto horrible. La habilidad sin sabiduría puede ser puramente destructiva y es muy probable que se revelara así. Aunque sólo sea por esta razón, es de gran importancia que quienes reciben una educación científica no se limiten a ser científicos, sino que posean cierta comprensión de esa clase de sabiduría que sólo puede impartir, si es posible tal cosa, la vertiente cultural de la educación*".

La Universidad, no debe, por tanto, convertirse en un lugar donde sólo se impartan conocimientos profesionales y se enseñe a hacer Ciencia. En ella se trata de conseguir el máximo desarrollo de los estudiantes universitarios, ayudándoles a descubrir sus cualidades, sus capacidades, su planteamiento vital; a ser capaces de comunicarse, de hacer crítica y autocrítica, de adquirir una postura ante su profesión, ante la ciencia, la técnica, la vida, ante el mundo. En definitiva, a adquirir ese sistema vital de ideas que les ayude a encontrar más honestamente su función social y su compromiso ante el mundo y ante ellos mismos. Se trata en suma de conseguir, parafraseando a Whitehead, "*cabezas bien hechas y no sólo bien llenas* ". Por ello, como indicaba Russell [Russell, 1997], los buenos profesores universitarios "*Deberían dar ejemplo del valor del intelecto y la búsqueda del conocimiento. Deberían dejar claro que lo que en cualquier época pasa por conocimiento puede en realidad ser erróneo. Deberían inculcar un*

temple de búsqueda continua y no de cómoda certeza. Deberían proponerse que sus alumnos fuesen conscientes del mundo como una totalidad... Mediante el reconocimiento de la probabilidad de error, deberían dejar clara la importancia de la tolerancia. Deberían recordar al alumno que aquéllos a quienes honra la posteridad muy a menudo han sido impopulares en su época y que, a este respecto, tener el valor de enfrentarse a la sociedad es una virtud de suprema importancia".

Sin embargo, fácilmente se observa que la Universidad actual no cumple esta misión, al menos desde un punto de vista formal y organizado. La enseñanza universitaria se ve actualmente desbordada por la cantidad de información que la Ciencia aporta a la enseñanza profesional, y por la propia investigación, usurpando todas estas materias el lugar de transmisión de la Cultura. Esto hace que el producto de la Universidad no sea un hombre centrado en su tiempo. Este problema ya había sido predicho por Ortega y Gasset y descrito en el año 1930. En sus propias palabras [Ortega, 1982], este fallo "*... produce un ser inculto. Un personaje medio que es el nuevo bárbaro retrasado con respecto a su época; arcaico y primitivo en comparación con la terrible actualidad y fecha de sus conocimientos. Este nuevo bárbaro es principalmente el profesional más sabio que nunca, pero también más inculto*".

Por eso es ineludible impulsar de nuevo en la Universidad la "*enseñanza de la Cultura o sistema de ideas vivas que cada tiempo posee*". Es una tarea universitaria imprescindible y como tal recogida en la Ley de Reforma Universitaria [MEC, 1983], que textualmente afirma también en su preámbulo: "*La Ciencia y la Cultura son la mejor herencia que las generaciones adultas pueden ofrecer a los jóvenes y la mayor riqueza que una nación puede generar, sin duda, la única riqueza que vale la pena acumular*".

Sin embargo, es fácil formular este objetivo y difícil concretar los medios para alcanzarlo en la estructura universitaria actual. Habría que empezar por convencernos a nosotros mismos y a los estudiantes de que no todas las actividades que se salen de la enseñanza estricta de la profesión son una pérdida de tiempo, siendo deseables la transmisión del gusto por la reflexión y la crítica constructiva sobre el entorno social que rodea a una persona y lo que ella misma, desde su ejercicio profesional, puede aportar a ese contexto.

2.1.3 La Universidad y la formación de profesionales

No sólo la Ley de Reforma Universitaria, y actualmente la Ley Orgánica de Universidades, como ya se mencionó, recogen entre los fines de la Universidad la formación de los profesionales, sino que los propios Estatutos de la Universidad de Salamanca establecen como

fin de la misma "*La contribución a la formación y perfeccionamiento de profesionales cualificados*".

Con anterioridad a cualquier planteamiento sobre la formación profesional, es preciso diferenciar claramente lo que es Profesión de lo que es Ciencia.

La Ciencia estudia los fenómenos, profundiza en sus causas y en sus mecanismos desarticulándolos con la finalidad de entenderlos, para posteriormente reestructurarlos aportando algo nuevo al conocimiento anterior. El conjunto de estos conocimientos, ordenados, en torno a una materia concreta, y puestos en práctica, es lo que se puede llamar Profesión; o viceversa, la Profesión es la puesta en práctica de los conocimientos obtenidos gracias a la Ciencia. De este modo como afirma Russell [Russell, 1997], el profesional desea cambiar la naturaleza, mientras que el científico desea comprenderla, y ninguna de las dos actitudes por sí sola es útil al hombre.

La Ciencia es la fuente de los conocimientos profesionales, y al entrar en la Profesión debe desarticularse como tal, y reorganizarse para ser puesta en práctica. Así, puede resumirse que el investigador "*hace ciencia*", el profesional "*pone en práctica los descubrimientos científicos*" y el docente "*transmite estos conocimientos*".

Acerca de esta diferente función de la Universidad, Ortega y Gasset dice lo siguiente [Ortega, 1982]: "*Es preciso separar la enseñanza profesional de la investigación científica, y que ni en los profesores ni en los alumnos se confunda lo uno con lo otro, so pena de que lo uno dañe a lo otro. En general, el estudiante normal, no es un aprendiz de científico. El Cientifismo en los profesores puede interferir en la formación de profesionales, ya que lo que se conseguirá será producir profesionales mediocres con un mínimo atisbo de Ciencia. Es preciso pues sacudir bien de Ciencia el árbol de las profesiones, a fin de que quede en ella lo estrictamente necesario, y pueda atenderse a las profesiones mismas*".

En el campo de la Informática la rápida evolución tecnológica implica que la formación no pueda limitarse a impartir los programas encaminados a la obtención del correspondiente título que certifique la capacitación para el ejercicio profesional en sí. Si de verdad se quiere dar respuesta a las necesidades reales de formación que la sociedad plantea, la docencia no puede restringirse a los estudiantes de primer, segundo o tercer ciclo. La Universidad como lugar donde "*se hace Ciencia*", debe estar abierta a cualquier profesional que desee una actualización o ampliación de sus conocimientos, "*enseñando los resultados de esa Ciencia*" en cada momento, a través de programas de formación continuada en permanente renovación.

Para no correr el riesgo de no formar ni buenos profesionales, ni buenos científicos, es necesaria la utilización de una metodología adecuada que obligue a detallar qué se pretende y qué medios se utilizarán para su consecución. Este soporte indispensable que constituye la metodología docente se tratará ampliamente en el capítulo siguiente, por lo que aunque se reconozca como fundamental esta función universitaria, no se desarrolla más bajo este epígrafe.

2.1.4 La Universidad y la investigación científica

La Ciencia se puede definir como un conjunto sistemático de conocimientos sobre la realidad observable, fruto de la investigación científica. No es otra cosa que el resultado de la investigación realizada de acuerdo con el método científico, por lo que la investigación científica es la fuente de la Ciencia.

Investigación deriva etimológicamente del latín "*in*" (en, hacia) y "*vestigium*" (huella, pista); su significado es "*hacia la pista*", averiguar algo siguiendo un rastro. Según esto, toda investigación, incluso la científica, es la búsqueda del conocimiento de algo no conocido, o la búsqueda de la solución de un problema. Por ello, los diferentes tipos de investigación no se pueden distinguir en su origen, sino que se van a distinguir por el método. De este modo para que una investigación se considere científica, es necesario que se utilice en ella el método científico.

Según R. Sierra Bravo [Sierra, 1986], "*El método científico, consiste en formular cuestiones o problemas sobre la realidad del mundo y de los hombres, sobre la base de la observación de la realidad y la teoría ya existentes; en anticipar soluciones a estos problemas, y en contrastar, con la misma realidad, dichas soluciones o hipótesis mediante la observación de los hechos, su clasificación y su análisis*".

Asimismo Russell [Russell, 1987] afirma que "*para llegar a establecer una ley científica existen tres etapas principales: la primera consiste en observar los hechos significativos; la segunda, en sentar hipótesis que si son verdaderas expliquen aquellos hechos; la tercera en deducir de estas hipótesis consecuencias que pueden ser puestas a prueba por la observación*". Puntualiza, además, que "*Decir que un hecho es significativo, en Ciencia, es decir, que ayuda a establecer o refutar alguna Ley General; pues la Ciencia, aunque arranca de la observación de lo particular, no está ligada esencialmente a lo particular, sino a lo general*".

El desarrollo del método científico puede resumirse en las siguientes etapas:

1. *Planteamiento del problema: reconocimiento de los hechos y reducción del problema a su núcleo significativo.*

2. *Construcción de un modelo teórico a partir de las hipótesis, y su traducción matemática si es posible.*
3. *Deducción de consecuencias particulares.*
4. *Prueba de las hipótesis, que en Informática al tratar con objetos materiales, se realiza por verificación, a diferencia de las ciencias formales cuyo criterio de verdad es la consistencia de los enunciados.*
5. *Introducción de las conclusiones en la teoría, con el consiguiente reajuste del modelo y la aportación de sugerencias para desarrollos posteriores. Esto es, discusión, conclusiones y generalización.*

Definido así el proceso metodológico científico, puede asumirse la definición de Sierra Bravo, que considera la investigación científica [Sierra, 1986] "*como una actividad compleja, inteligente, constituida fundamentalmente por la previa documentación existente, por el proceso de aplicación del método científico a problemas concretos en un área de la realidad observable, buscando respuesta a estos problemas para obtener nuevos conocimientos que se ajusten lo más posible a la realidad investigada, con elaboración de los resultados obtenidos en un trabajo de investigación o tesis y su presentación de forma escrita*".

El hecho de que la Universidad, en la actualidad, no es el único lugar donde se realiza investigación científica es un hecho incuestionable. Pero también lo es el hecho de que a pesar de sus limitaciones, la Institución Social mejor preparada para asumir hoy el reto del desarrollo científico-técnico e impulsar la mentalidad y el espíritu científico en España es la Universidad.

2.1.5 Historia de la Universidad de Salamanca

Cuando se está vinculado a una institución con casi 800 años de historia, considerada como la Universidad existente más antigua de España, y que ha acogido a nombres tan importantes como Antonio de Nebrija, Francisco de Vitoria, Fray Luis de Léon o Miguel de Unamuno, la tentación de reflejar algo de su dilatada existencia es demasiado grande como para resistirse.

La historia de la Universidad de Salamanca se puede decir que sigue una periodización convencional: etapa medieval, moderna y contemporánea. Durante la primera de ellas, la Universidad de Salamanca no es mucho más que una universidad jurídica y peninsular destacada. En los siglos modernos se convierte en la más afamada e influyente de la Monarquía Hispánica y, tras esta etapa clásica, se sume en un declive provinciano que, arrastrado por el siglo XIX, no se irá remontando sino en el curso del XX.

El breve recorrido histórico que aquí se presenta está extraído de [USAL, 2002] y de [CEHU, 2002], que a su vez citan a [Rodríquez-San Pedro, 1991; Rodríquez-San Pedro, 1996] como las fuentes principales de los contenidos de sus páginas.

2.1.5.1 Etapa medieval: Fundación y Consolidación

La Universidad de Salamanca fue fundada por Alfonso IX de León, posiblemente hacia fines del año 1218, principios de 1219, con categoría de "Estudio General" de su reino. Cabe señalarse que el término "Universidad", con la significación que hoy se le otorga, no aparece en los documentos salmantinos hasta el siglo XV, ya que durante el siglo XIII esta palabra poseía una significación corporativa.

Dentro del panorama europeo, la Universidad de Salamanca se fundó con posterioridad a otras Universidades destacadas como Bolonia, París, Montpellier u Oxford. Más aún, se inspira en modelos boloñeses, con lo que se sitúa dentro del tipo de las llamadas Universidades Meridionales de orientación jurídica, frente a la preferencia por la enseñanza de la teología o las artes que caracterizaban a París u Oxford, por ejemplo. Es, sin embargo, la más antigua de las universidades peninsulares hoy existentes, dada la efímera aparición de la Universidad de Palencia alrededor de los años 1175-1180 (para otros 1208-1214).

Es, por lo tanto, el título de "Estudio General" el que manifiesta la diversidad de sus enseñanzas, su característica no privada (abierto a todos) y la validez de sus graduaciones. La institución recibió en 1254 su Estatuto, otorgado por el rey Alfonso X el Sabio, y así en el libro de Las siete partidas se regula el funcionamiento de la institución. Se consolidaban, de este modo, 12 cátedras, con disciplinas de Derecho Canónico, Civil, Medicina, Lógica, Gramática y Música.

El espaldarazo final le llega en 1255, cuando el Papa Alejandro IV le otorga a la Universidad de Salamanca la "*licentia ubique docendi*", con reconocimiento de la validez internacional de sus grados, salvo en París y Bolonia, restricción que es abolida en el año 1333, y el uso de un sello propio.

La organización institucional de este período medieval quedó consolidada a través de diversas constituciones pontificias; las del Papa Benedicto XIII (el Papa Luna) en 1381 y 1411 que fija las rentas de la Universidad y dota 24 cátedras, y las definitivas de Martín V en 1422 que elabora sus primeros estatutos, que seguirán rigiendo en sus capítulos esenciales hasta el siglo XIX.

Figura 2.1. Reyes Católicos en la Fachada de la Universidad

Figura 2.2. Representante de Virgo en el Cielo de Salamanca

A finales del siglo XV y durante el XVI se estrecha la relación con la monarquía, representada por los Reyes Católicos, que dotan a la Universidad de nuevos privilegios y estatutos. La iconografía de la fachada muestra esa relación de la Universidad con la Corona (Figura 2.1), e ilustra su programa, erasmista, en opinión de algunos estudiosos, y contrapuesto al de Maquiavelo, en cuanto a la educación del príncipe cristiano: no se trata de conservar el poder por parte del tirano, sino de convertirlo en príncipe para la justicia y la paz.

Cristóbal Colón estuvo en Salamanca entre noviembre de 1486 y enero de 1487, siguiendo a los Reyes Católicos y su Corte. Los maestros del Estudio seguían con interés los avances de los portugueses en la costa atlántica, y entre ellos destacaba el astrónomo Abraham Zacut, de cuya obra Almanaque Perpetuo se sabe que Colón disponía de un ejemplar. Bajo la inspiración de Zacut, Fernando Gallego empezó a pintar las figuras del zodiaco representadas en el Cielo de Salamanca (Figura 2.2), hoy en el Museo Universitario.

Respecto a la distribución de poderes, se aprecia una amplia participación estudiantil en el gobierno del Estudio, según el mencionado modelo boloñés: el rector es un estudiante, y le asesora un consejo de otros 8 escolares territorialmente representativos. Frente a ellos se van estructurando contrapesos progresivos, con introducción de influencias parisinas, tales como la participación de los profesores desde los claustros de diputados y plenos, claramente consolidados para el siglo XV. En concreto, el claustro de diputados se diseñó para conseguir un cierto equilibrio de poderes: 10 de sus miembros eran catedráticos ordinarios o de propiedad, y otros 10 pertenecerían al profesorado auxiliar y a los graduados o simples estudiantes. Por lo que respecta al Claustro Pleno, se trata de la asamblea máxima, con participación del rector,

catedráticos, diputados y consiliarios estudiantes. A lo dicho hay que agregar la decisiva figura del maestrescuela catedral, vitalicio, representante del poder científico, juez del Estudio en lo civil y criminal, y en quien recae la potestad de la colación de grados. Finalmente, cabe señalar la existencia del primicerio o presidente del claustro de catedráticos.

La autonomía institucional se consigue por medio de una financiación peculiar. Se trata de una participación en los diezmos eclesiásticos a través de las tercias reales del obispado de Salamanca. Esto vincula la solidez económica con los ritmos agrarios del entorno, produciéndose agudas insuficiencias durante las convulsiones críticas del siglo XIV. Respecto a los repartos salariales, el profesorado jurista resulta el más favorecido proporcionalmente, lo que denota la destacada valoración de estas facultades en la época. Por lo demás, los profesores auxiliares o ayudantes no recibieron estipendios hasta alrededor del año 1439, y éstos jerarquizados y diferentes según disciplinas, del mismo modo que las cátedras vitalicias. No obstante, el profesorado podía recurrir a complementos económicos a través de beneficios eclesiásticos.

El predominio continuó recayendo en las disciplinas canónicas, dada la asistencia mayoritaria de clérigos. Además, la Facultad de Teología comienza su funcionamiento entre 1381-1386, y se robustece desde principios del cuatrocientos; los teólogos podían cursar en la Universidad o en los Estudios conventuales de dominicos y franciscanos, con posibilidad de convalidaciones. Resta señalar, en este apartado, el hecho de que regentaban las diversas cátedras ordinarias doctores y licenciados, mientras que las cursatorias quedaban encomendadas a bachilleres.

El método pedagógico comprendía "*lectiones*", "*repetitiones*" y "*disputationes*", como en el resto de las Universidades Europeas del momento. Se trataba de comentarios analíticos sobre textos consagrados, conferencias magistrales públicas y ejercicios dialécticos. La lengua académica era el latín, lo que facilitaba los intercambios y la movilidad internacional. Las autoridades de referencia eran el derecho civil romano justinianeo ("*corpus iuris civilis*") o el derecho pontificio medieval ("*corpus iuris canonici*"); así como los clásicos grecolatinos y Aristóteles. No existían exámenes de curso, sino pruebas finales o grados académicos: bachiller, licenciado y doctor. Hay que advertir, además, que Salamanca no impartió sus clases en edificios propios hasta el siglo XV, y que, con anterioridad, pululaban los maestros por dependencias catedralicias y locales dispersos, alquilados o cedidos.

Salamanca se constituye como uno de los más destacados centros universitarios hispanos medievales, junto a los de Coimbra, Valladolid y Lérida, principalmente. Predominaban en ellos

las enseñanzas jurídicas, y se produce cierta movilidad del alumnado por Universidades como Bolonia (Derecho), París (Teología) y Montpellier (Medicina).

El desarrollo jurídico contribuye a la conformación de las estructuras gubernativas de la Iglesia y del Estado, con un cierto talante autoritario-romanista. Los canonistas salmantinos llegan hasta la Curia romana o, junto a los teólogos, participan en concilios como los de Constanza y Basilea, a comienzos del cuatrocientos. Sin embargo, las posturas conciliaristas que allí defendieron se diluyeron posteriormente por el hecho de que la Salmantina terminó subsistiendo gracias a una decidida protección Papal. Por ello, a lo largo del siglo XV, Salamanca se configura como una Universidad dentro del sistema romanista y canónico; y únicamente desde fines de dicho siglo se aprecian algunos atisbos humanistas, en buena parte por influencias externas.

La Teología, por su parte, se mueve dentro de la ortodoxia, con raras excepciones, como la condena en 1479 de ciertas doctrinas del maestro Pedro Martínez de Osma sobre la penitencia. De otro lado, la abundancia de manuscritos científicos en algunos colegios, como el de San Bartolomé, vinculados a las cátedras de Filosofía Natural y Astrología, señalan un desarrollo importante de estas disciplinas, por lo menos en pequeños cenáculos. El panorama se completa con la existencia de una Biblioteca central universitaria (Figura 2.3), que contaba con unos 200 volúmenes hacia 1470, y que debía abrirse a los estudiosos unas cuatro horas al día. La imprenta se introdujo, asimismo, en la ciudad hacia 1472, pero la dependencia universitaria respecto a las grandes imprentas y circuitos europeos se mantuvo a lo largo de los siglos. XVI y XVII.

Figura 2.3. Imágenes de la biblioteca de la Universidad de Salamanca

Una primera consideración, dentro de los aspectos sociales, recae en el hecho de que la Universidad medieval excluyó sistemáticamente de sus aulas al potencial alumnado femenino. Y esto, que era cierto para toda Europa, lo era también para Salamanca. Pero no se debe

tampoco pensar en grandes contingentes de escolares varones. Frente a los 10.000 matriculados que ostentaba Bolonia a fines del siglo XII, la Universidad de Salamanca de fines del XIV quizás alcanzara los 500 ó 650, elevándose a unos 3.000 entrado el siglo XVI. En conjunto, predominaban los clérigos sobre los laicos, y entre aquéllos los canónigos. Estos estudiantes se agrupaban en "naciones" o asociaciones de apoyo mutuo. En un principio debieron ser 4: una comprendería las diócesis galaico-portuguesas; otra el resto de las leonesas; la tercera la provincia eclesiástica de Burgos; y la cuarta la provincia eclesiástica de Toledo. Para el siglo XV estas cuatro naciones se habían desdoblado en las 8 consiliaturas asesoras del rector. Por contingentes de procedencia, parece evidente un predominio de ambas Mesetas (sobre todo la Meseta Norte) y del Noroeste peninsular (Galicia-Asturias-Portugal); asimismo, se encuentran reducidas proporciones de Extremadura y Andalucía; raros aragoneses y rarísimos escolares extranjeros no peninsulares. Hay que advertir, no obstante, que la ausencia de procedencias de la Corona de Aragón se debía a una mayor preferencia por los traslados hacia Bolonia o a las universidades del mediodía francés.

Parecidas circunstancias a las descritas concurren con el profesorado: un predominio del originario de Castilla-León y Portugal, con algunas excepciones de extranjeros aventureros.

Por último, cabe destacar, entre fines del XIV y comienzos del XV, los inicios y desarrollo de la fundación de colegios, instituciones beneficiales de acogida de estudiantes, con amplia tradición en Francia e Inglaterra: en 1386 se fundaba el de Pan y Carbón; y en 1401 el que luego sería Mayor de San Bartolomé (inspirado en el de los Españoles de Bolonia, c. 1367).

2.1.5.2 Siglos XVI y XVII

El tránsito de la etapa medieval a la moderna irá convirtiendo a la Salmantina en Universidad modelo, una especie de estereotipo de prestigio, celebrándola como la primera, afamada y más influyente universidad de las Españas. Es decir, la institución de educación superior sobresaliente entre las 32 fundaciones con grados reconocidos existentes en la Península Ibérica hacia 1625; pluriforme en materias de enseñanza, con las cátedras mejor dotadas, y la menos regional en sus contingentes de alumnado. No cabe duda de que tales primacías se debieron substancialmente al desarrollo de los estudios jurídicos y, en segundo plano, de los teológicos, con lo que se convertía en foco universitario volcado en las necesidades burocráticas de vertebrar las estructuras del Estado y de la Iglesia y asumir la defensa y expansión de la fe Católica.

Más aún, la circunstancia americana otorgó a Salamanca "la ocasión para la mayor expansión de una universidad que han visto los siglos"; expresión ésta con la que se ha querido

definir la floración de numerosas Universidades referentes en suelo indiano. Por lo que se refiere al equilibrio de poderes, la consolidación de una Monarquía autoritaria desde fines del XV, con el reinado de los Reyes Católicos y sus sucesores, reafirmó la intervención regia en los asuntos académicos, a través del Consejo de Castilla, con una cierta marginación de las iniciativas Papales y su protagonismo medieval.

De modo que los controles estatales tomaron forma de visitadores periódicos, con potestad para impulsar y canalizar reformas y sucesivas modificaciones de estatutos internos. No obstante, el marco jurídico prosiguió dentro de las Constituciones Pontificias de 1422, a las que se fueron añadiendo estatutos complementarios en 1538, 1561, 1594, 1604 y 1618, culminando con la Recopilación General de 1625, que se constituirá en flexible marco de referencia jurídica hasta 1771 y las reformas subsiguientes.

Aún cuando existieron algunos intentos de reforma, el poder ejecutivo continuó en el rector, estudiante generoso y noble habitualmente, asesorado por un consejo consultivo de 8 estudiantes, representantes de las diversas cofradías regionales de estudiantes y elegidos a propuesta de éstas. El maestrescuela catedral mantuvo el simbolismo de la autoridad Papal, ejerciendo jurisdicción, mediante tribunal propio, sobre todo el gremio universitario. Asimismo, se mantuvieron los diversos claustros, como organismos de gobierno administrativo, económico y académico. Con todo, hay que señalar durante esta etapa una tendencia a la aristocratización de los poderes, intentando reducir la participación estudiantil, concentrando responsabilidades en los catedráticos de propiedad y aumentando las preeminencias de las oligarquías colegiales. Las facciones y camarillas fueron continuas y, desde mediados del siglo XVII, parece apreciarse una disolución del sistema asambleario gubernativo hacia la proliferación de juntas especiales decisorias.

La hacienda universitaria mantiene sus fundamentos medievales: la participación en los excedentes agrícolas circundantes a través de las tercias reales sobre el diezmo. Entre 1650 y 1700, las medias quinquenales de estos ingresos se situaron entre 6.000.000 de maravedís en años bajos y 15.000.000 en años prósperos, con medias seculares de 8.500.000 aproximadamente. Y, como las tercias suponían entre el 80 y 85% de la recaudación total, la Universidad mimetizará en su economía el discurrir cíclico de la Castilla interior, con una situación más próspera en el siglo XVI que en el XVII, y una progresiva recuperación a lo largo del XVIII. En este contexto, el pago de las cátedras se elevaba al 50% del gasto, bien entendido que siguieron manteniéndose fuertes desigualdades en las dotaciones y pago, con predominio de las disciplinas jurídicas y las cátedras de propiedad. Las facultades privilegiadas, Derecho y

Teología, copaban a fines del XVII el 65% de los salarios globales del profesorado, los cuales se complementaban, asimismo, con propinas de actos y grados académicos.

Sobre estas bases se alzaba el régimen docente, que hacia 1650 articulaba en torno a 26 cátedras de propiedad y unas 30 temporales, cursatorias o regencias. Estas cátedras se proveían por votos de estudiantes, según sistema boloñés que se mantuvo hasta 1623 y 1641; a partir de estas fechas fue suprimido por irregularidad, corrupción y conflictividad.

Las cátedras pasaron entonces a ser proveídas por el Consejo de Castilla, lo que abocó hacia acaparamientos partidistas por parte de las oligarquías burocráticas y colegiales. Y es que los colegios, surgidos como instrumento para la conformación de una élite académica preparada para el acceso a grados, cátedras y oficios de la administración, terminaron coaligándose en intereses con los altos burócratas del aparato estatal: si éstos promovían a los colegiales a cátedras y cargos, los colegios otorgarían becas a familiares y allegados de sus bienhechores. De este modo, la seguridad de la beca colegial y el turnismo de los ascensos, primando la antigüedad y la capacidad de influencias sobre el mérito, dislocaba todo interés por el estudio.

El estudiante manteísta meritorio, no colegial, termina desmoralizándose ante los rodillos de parcialidades y camarillas. Y esta selección endogámica del profesorado tomó la forma de turnismo en las cátedras jurídicas entre colegiales. Asimismo, las enseñanzas teológicas pasan a turnos en la primera mitad del siglo XVIII, aunque anteriormente, desde 1606, se había producido una progresiva dotación de cátedras sin oposición, vinculadas a las doctrinas de órdenes diversas.

El método medieval de enseñanza se mantuvo, fundamentado en la lección magistral, la relección y las disputas académicas y ejercicios dialécticos. El principio de autoridad se derivaba de ciertos libros y autores consagrados: Corpus de Derecho Romano y Decretales Pontificias; la Biblia y una Escolástica Teológica de predominio tomista en el siglo XVI; Síntesis Galénica en medicina; Lógica y Filosofía aristotélicas; Euclides, Ptolomeo y los clásicos latinos y griegos... Todo ello se consolidó en dos tiempos, los Planes de Estudio de 1561 y 1594, completados con modificaciones parciales para las artes en 1604.

A partir de aquí, la Recopilación de 1625 rige como referencia, aunque con negligencias y relajaciones en su cumplimiento. Además, los abusos en el dictado produjeron considerables retrasos de los programas. Por su parte, los cursos comprendían 6 meses y 1 día desde la fecha de la matrícula, y las clases cesaban únicamente entre el ocho de septiembre y el dieciocho de octubre. No existían exámenes finales y el “pase de curso” requería tan sólo matrícula y asistencia. La revalidación de conocimientos se producía a través de los grados de bachiller, licenciado y doctor: el primero de ellos servía para el ejercicio profesional, mientras que el

segundo probaba la habilidad erudita para la futura docencia, y el doctorado era mera cuestión de pompa y festejos. Todo esto tenía lugar en las Escuelas Mayores y Menores, que constituían la Universidad por excelencia. A ella se agregaban unos 20 conventos regulares masculinos y más de 25 colegios vinculados, con ciertas tensiones de disgregación y enseñanza autónoma, sobre todo en los primeros.

Con estas coordenadas, la Universidad Salmantina de los siglos modernos presenta un perfil de acusado carácter jurídico y de promoción burocrática y funcionarial: una institución estatal y eclesiástica muy vinculada al "*cursus honorum*" letrado, eclesiástico y administrativo. Esto no obstaculizó que, desde fines del XV y primera mitad del XVI, Salamanca se incorporase al movimiento humanista; aunque, ciertamente, ensombrecida por Alcalá que, en su apogeo renacentista, le restará alumnos. Por los años centrales del siglo XVI, la confluencia del Derecho, la Teología Tomista, las nuevas lógicas y las lenguas clásicas, cristalizan en la llamada "Escuela de Salamanca".

En esta etapa, Salamanca permaneció fiel a los cauces jurídicos del "*mos italicus*"; mientras en teología se multiplicaron las escuelas teológicas desde fines del siglo XVI y las sistematizaciones escolásticas durante el XVII. El cosmos aristotélico mantiene su pervivencia hasta bien entrado el XVIII. La Universidad de Salamanca, como otras de su tiempo, parece adormecerse, conservar sus saberes, erigirse como brazo letrado y legitimación ortodoxa de un orden social. Puede hablarse de un cierto declive teórico en esta institución, desincorporada del racionalismo filosófico y del cientifismo experimental de la naturaleza, propios de las vanguardias del seiscientos. No obstante, su contribución a la formación de los cuadros jurídicos y administrativos de la Monarquía y de la Iglesia resultó destacada. Y esta preocupación práctica, junto al desarrollo de una teología ortodoxa, contribuyó a desatender las disciplinas de pura erudición y las lenguas auxiliares.

Además, este acusado predominio del derecho y de la teología marginó, incluso económicamente, a las restantes disciplinas; así, a fines del XVII, la cátedra de Matemáticas-Astrología, junto con la de Música, eran las peor pagadas de todas las de propiedad. El temor a la herejía y a las "novedades peligrosas" fija la permanencia de los viejos Planes de Estudio.

En este punto, cabe referirse también a la Biblioteca universitaria que, hasta 1550, fue nutriendo sus fondos e incorporando libros de humanidades. Sin embargo, estas adquisiciones disminuyen desde esta fecha, y en 1610 presenta unos locales descuidados y un contingente de 1.250 volúmenes, que rezuma saberes medievales y arcaísmo. El hundimiento de las bóvedas de la sala de lectura en 1664 ennegrece aún más el panorama, pues los libros permanecerán arrinconados y desordenados hasta 1693.

Será en la primera mitad del siglo XVIII cuando la biblioteca se renueve con nuevas compras y locales; pero cumple señalar que, durante su etapa clásica, las bibliotecas de instituciones privadas (colegios-conventos) y las particulares constituyen fondos culturales más nutridos y efectivos que la propiamente universitaria. Y cuando, finalmente, se consolidan sus fondos en el siglo XVIII, en gran parte procederán del colegio-convento de los jesuitas expulsados.

Ante las perspectivas que se abrían, la matrícula alcanzó entre 5.000 y 7.000 alumnos anuales en la segunda mitad del siglo XVI, si bien a mediados del XVII se hará patente un declive que aboca a los 2.000 matriculados de las postrimerías del seiscientos. Entre ellos continuaron predominando los juristas, destacadamente los canonistas, siguiendo en importancia la Teología y las Artes, con pequeños contingentes de médicos.

Por lo que respecta a las procedencias, durante la segunda mitad del XVI, el prestigio de Salamanca atraía hacia sí una confluencia de estudiantes de todo el ámbito peninsular, e incluso europeos e indianos en proporciones superiores a cualquier otra universidad hispana de la época. De este modo, Salamanca se configura como la menos regional de las tres grandes universidades de la Monarquía (además de Valladolid y Alcalá); y esto a pesar del predominio del alumnado meseteño: y es así que los 9.000 portugueses que pasaron por sus aulas entre 1580-1640, podrían dar testimonio de su pluralidad. Estos estudiantes se agrupaban en asociaciones y cofradías regionales que, a fines del XVI y principios del XVII, eran ocho: Galicia-Portugal-Campos (Castilla la Vieja y León)-Vizcaya-Extremadura-La Mancha-Andalucía-Corona de Aragón.

Una de las figuras más destacadas de este período fue Fray Luis de León (1527-1591). En 1572, bajo el reinado de Felipe II, se produce el proceso a Fray Luis de León por una Inquisición que se halla entonces en su máximo poder. Su delito había sido traducir al castellano el Cantar de los Cantares del rey Salomón, contra la prohibición del Concilio de Trento de verter al romance los textos bíblicos. También se había atrevido a manifestar que la Vulgata, la versión latina de San Jerónimo, contenía numerosos errores. Denunciado ante el Tribunal de la Inquisición y descubierto su linaje converso, Fray Luis fue apartado de su cátedra y encarcelado en Valladolid durante casi cinco años, hasta que en 1576 fue puesto en libertad y regresó a su cátedra de Salamanca.

2.1.5.3 Siglo XVIII

En los aspectos institucionales, las reformas articuladas en torno al Plan de 1771 pretendieron un mayor control del Consejo Real sobre la autonomía universitaria. Asimismo, se reforzó la

autoridad rectoral, prolongando su mandato a períodos de dos años (desde 1770), y reservando el cargo para graduados mayores, con exclusión de catedráticos. Este reformismo vino acompañado de una pareja merma de la autoridad del maestrescuela y de su jurisdicción. Por otro lado, una vez desarticulada la prepotencia jesuita, tras la expulsión de la orden en 1767, la Monarquía y ciertos grupos ilustrados pretendieron atenuar la influencia colegial, tanto en la burocracia estatal como en la provisión de cátedras universitarias. Diversas disposiciones reales se sucedieron entre 1771 y 1777 para la reforma de los colegios, aunque, a medio plazo, parece que se reprodujeron los antiguos vicios.

A pesar de todo, el reformismo dieciochesco no proporcionó nuevas rentas económicas al Estudio, ni pretendió conseguir una distribución más equitativa de los ingresos. Los catedráticos de propiedad continuaron gozando de una desmedida participación en las rentas decimales, reivindicando privilegios remontables a 1422. Con ello, la mayor parte de la reforma hubo de sufragarse a través del arca de gastos comunes, con las dificultades a ello inherentes. En conjunto, la Facultad de Medicina fue la más favorecida económicamente por las nuevas disposiciones, incorporando, incluso, nuevos diezmos del obispado en el tardío año de 1789, al tiempo que se producía un incremento de los asignados de sus cátedras cursatorias. En el capítulo de régimen docente se había llegado al consentimiento tácito de una costumbre de oposiciones formularias, con turnismos y antigüedad de acceso para colegiales y regulares. Todo ello se mantenía hacia mediados del XVIII, y las reformas se dirigieron a conseguir un concurso-posición abierto, bajo la supervisión del Consejo. Se trataba de abolir los turnos y abrir las oposiciones al mérito y a la concurrencia (1770). Además de esto, el Plan de Estudios exigió una atención especial, lo que cristalizó en las disposiciones de 1771. No hay que considerarlas como revolucionarias, pero contribuyeron a la introducción de nuevos enfoques y materias de estudio. En Derecho Canónico se favorecían las corrientes regalistas, limitando el estudio del derecho medieval pontificio.

En leyes se reglamentó la enseñanza del Derecho Real o Nacional, aunque continuó predominando el romanismo. En Teología se pretendió acentuar los aspectos bíblicos y positivos, así como las disciplinas prácticas, pero la escolástica retornó al predominio tomista, frente a la proliferación de escuelas del siglo anterior. Se produjo una apertura hacia los estudios de física experimental para los médicos que, al mismo tiempo, incrementaron el talante empírico-clínico de la enseñanza.

Finalmente, tanto las Matemáticas como las Letras Clásicas recibieron protección decidida. Lo que no se modificó demasiado fue el método docente, consolidándose la lección magistral y las tradicionales disputas. No obstante, la introducción de compendios y manuales terminó con

el dictado, y la relección cayó en desuso. Se incrementó la normativa para el control de asistencia necesario para el "pase de curso", y no llegaron a introducirse exámenes anuales. Por su parte, se consolidaron los estudios de licenciatura, que tomaron apariencia de cursos académicos, con inclusión en ellos de nuevas materias de estudio.

Algunas de estas reformas resultaron particularmente acertadas. Y así, en el Plan de Estudios Médico, los proyectos salmantinos mantendrán vigencia a través de los programas estatales hasta mediados del ochocientos. En definitiva, cabe afirmar que hacia finales del setecientos se habían producido en Salamanca renovaciones y novedades en las disciplinas médicas, con el apoyo de las cátedras científicas de Artes y ciertos sectores del Derecho.

Esto iba preparando el tránsito hacia un nuevo tipo de Universidad que, en el siglo XIX, marginará de su ámbito a canonistas y teólogos, que habían sido secularmente sus auténticos señores.

En otro orden de cosas, la Biblioteca universitaria experimentará un impulso decisivo, no sólo por la restauración y nuevas adquisiciones de la primera mitad de siglo, sino por el ingreso de 1.000 volúmenes procedentes del colegio de los jesuitas expulsados. Lo que parece también evidente es que durante el siglo XVIII se redujo la proyección exterior de la Universidad en el ámbito de las Españas, mientras ascendía la importancia de otros centros hasta entonces periféricos.

El alumnado oscilará entre 2.000 y 1.500 matriculados, con fuertes contingentes de regulares y colegiales en la primera mitad del siglo, lo que delata a una Universidad en la que se han consolidado ciertos sectores privilegiados o influyentes. Además, progresivamente, la procedencia del alumnado se regionaliza hacia la Meseta Norte y Extremadura, disminuye la capacidad de convocatoria y promoción de Salamanca, y la competencia de otras universidades (Zaragoza, Valencia, Valladolid, Sevilla), así como el traslado del pálpito social hacia regiones periféricas, van sumiendo a la que había sido primera Universidad de la Monarquía en el declive provincial que heredaría el siglo siguiente.

2.1.5.4 Siglos XIX y XX

Salamanca, símbolo universitario del Antiguo Régimen, pierde sus referencias gloriosas durante la etapa contemporánea. A lo largo del siglo XIX no pasará de constituir una evocación ruinosa, y a partir del inicio del novecientos se va trabajosamente reconstruyendo al nivel de Universidad de provincias. La Ley Pidal (1845) constituye el punto de partida de la Universidad liberal española, un nuevo modelo constitucional que consolida sus directrices en la Ley Moyano de 1857: centralización de la educación superior por el Estado; control de fondos, programas y

libros de texto; funcionarización del profesorado... Salamanca pasó a la categoría de Universidad provinciana, uno más de los 10 distritos que se configuran de nuevo cuño. Será Madrid y su Universidad Central omnipotente quienes suplantarán a la Salamanca modélica, y la capital del Reino y su universidad se convierten en el punto de referencia de la vida académica y científica del siglo XIX.

En este contexto, se diluye la antigua estructura administrativa institucional de Salamanca, que queda integrada como una más en el tejer y destejer de los proyectos universitarios de carácter centralista y uniformizador. En 1820 se produce una breve restauración del Plan Caballero de 1807, con posterioridad a la pretensión de Fernando VII de retrotraerse a los planes dieciochescos de 1771. Por su parte, reglamentos y planes liberales como los de 1821 y 1836 culminarán en el mencionado de Pidal en 1845, cristalización del proyecto universitario moderado, de inspiración francesa. La Ley Moyano de 1857 supone, por su parte, el espaldarazo definitivo. El rector universitario pasa a convertirse en una especie de delegado del gobierno central, designado por el ministerio correspondiente.

No menores cambios registra el capítulo hacendístico, pues las leyes desamortizadoras privan a Salamanca de las tercias decimales, fundamento económico de su tradicional autonomía. A partir de aquí, los ingresos tenderán a establecerse sobre los derechos de matrícula, grados y exámenes; es decir, según una filosofía en la que la instrucción pública debía ser costeada por quienes la recibieran. Estos ingresos se complementan en forma de presupuesto de Instrucción Pública, otorgado por el gobierno central, y que, por su insuficiencia, hubo de completarse con ayudas paralelas de la Diputación y del Ayuntamiento locales. El 80% del gasto se destinaba a sueldos de personal, y el salario de los catedráticos aparece ya fijo y homologado, contrariamente a las variaciones y jerarquías del Antiguo Régimen.

Las modificaciones docentes fueron, asimismo, destacadas. Tras la Ley Moyano, Salamanca quedó reducida a las facultades de Teología, Derecho y Filosofía y Letras. Para culminar el desmantelamiento, las tendencias laicizantes de los gobiernos liberales condujeron a la supresión de los estudios de Teología en 1868, los cuales perdieron rango universitario y se trasladaron al Seminario Diocesano. El declive tuvo un cierto paliativo por el hecho de que al año siguiente, en 1869, la Diputación refundó y financió a su costa una Facultad libre de Medicina y Ciencias, de cuyo sostenimiento se hará cargo posteriormente el Ayuntamiento. Salamanca quedaba limitada a un puñado de Facultades con escasos alumnos, y de las cuales la de Derecho se mantendrá como la más influyente y nutrida.

En el método pedagógico, la ruptura con el pasado vendrá simbolizada en el abandono del latín por el romance, lo que suponía un claro distanciamiento de la trayectoria eclesiástica y

cultural tradicional. Se implanta la lección magistral a partir de los libros de texto oficiales, con un aprendizaje memorístico y la introducción de exámenes finales por asignaturas. La docencia se torna rutinaria, tanto más cuanto que no se exige la investigación e, incluso, ha desaparecido la posibilidad del doctorado, que pasa a ser conferido en Madrid.

Las clases continúan impartiéndose en los viejos edificios tradicionales, con algunos anexos en el Fonseca y la financiación de un jardín botánico. La Biblioteca central se incrementa con fondos procedentes de la desamortización del convento de San Esteban en 1834, y con algunas donaciones particulares. Por su parte, las facultades de Derecho y Letras comienzan la consolidación de bibliotecas propias.

Salamanca se configura en esta época como un poblachón monumental de acusado ruralismo en torno a una universidad disminuida. El colorido social del Antiguo Régimen se ha atenuado. Han desaparecido de las calles los generosos, los pícaros y los buscavidas, sustituidos por la uniformidad del estudiante burgués. Siguen siendo varones, pero los límites de procedencia se han replegado al distrito. Su número ha disminuido respecto al siglo anterior; aunque los 200/300 matriculados en la mitad de la centuria se hayan incrementado hasta unos 1.000 en sus postrimerías, contabilizando entre ellos a los alumnos libres. Se trata de futuros funcionarios de la administración pública o aspirantes a profesiones liberales, horizontes máximos para una promoción social de provincianos.

La Universidad no es en el ochocientos sino un despojo que, incluso, corrió el riesgo de desaparecer como tal. Poco quedaba de la tradición precedente y de sus símbolos; si acaso la retórica. A fines de la centuria, un grupo de tradicionalistas en torno del obispo Cámara abrigaron proyectos de convertirla en Universidad Católica. Por las mismas fechas, una celebración del IV Centenario del Descubrimiento de América promovió la idea de constituir una Universidad Hispano-Americana. El hito de 1900 nos sitúa ya casi en el ayer, e inicia un lento proceso de reconstrucción.

Una de las figuras emblemáticas, ligada a la Universidad de Salamanca de principios del siglo XX, fue sin duda alguna Miguel de Unamuno y Jugo (1864-1936). En la Universidad de Salamanca, sin menoscabo de sus muchos otros méritos - ni de los de quienes le precedieron y siguieron - Unamuno es El Rector por excelencia. Precisamente su mandato, iniciado en 1900, a los 36 años de edad, sigue al de Esperabé Lozano, que estuvo al frente de la Universidad durante 31 años -desde 1869- y a quien se debe la lenta y parcial recuperación del Estudio, en decadencia desde el siglo XVII, acentuada en el XIX por la reforma centralizadora de las Universidades. En 1901 se produjo el reconocimiento oficial de las Facultades de Medicina y Ciencias, que habían sobrevivido apenas gracias al apoyo de las instituciones locales.

Figura 2.4. Retrato de Unamuno

El de Unamuno fue un período conflictivo para el país y para la propia Universidad. En 1903 los estudiantes se enfrentaron al poder gubernamental con el saldo de dos muertos y varios heridos. Unamuno había intentado evitar la violencia aconsejando a los estudiantes que lucharan "contra la razón de la fuerza con la fuerza de la razón". En 1914, el Gobierno cesaba a Unamuno como rector. Ortega y Gasset, su gran polemista, le ofreció su apoyo. "De un modo o de otro venceremos. Luego seguiremos nuestra polémica", le escribió. El golpe de Estado del general Primo de Rivera, en 1923, provocó la reacción de Unamuno, que denuncia la arbitrariedad de aquel gesto. "*No caigáis, estudiantes españoles, en la dementalidad del carnero, el macho de la oveja, indigentísimo en seso y opulentísimo en sexo... Es la inteligencia la que ha de salvar a la patria...*", pedía no sólo a sus estudiantes sino a los de todo el país, en oposición al Manifiesto de Primo de Rivera.

Su destierro a Fuerteventura provocó la protesta de los mejores intelectuales españoles. En 1930, caída la dictadura de Primo de Rivera, Unamuno volvió al rectorado. En 1934, con motivo de los actos celebrados por su jubilación -en los que se le nombró rector vitalicio-, Unamuno previó el peligro de una guerra civil. Cuando ésta se inició en 1936, Unamuno advirtió a los insurrectos, en acto solemne celebrado en el paraninfo del Estudio el 12 de octubre de ese año: "Venceréis, pero no convenceréis". Poco tiempo después fue destituido de su cargo. Murió dos meses más tarde, el 31 de diciembre de 1936. La Universidad de Salamanca conserva su archivo, su biblioteca y sus objetos personales en la Casa Museo Unamuno.

La Universidad de Salamanca tiene hoy un gran prestigio en el ámbito exterior, así como entre las Universidades españolas. Cuenta en su planilla docente con unos 2.500 profesores y en la de administración y servicios con alrededor de mil personas. La Universidad de Salamanca imparte 39 Licenciaturas, 6 Ingenierías Superiores, 12 Ingenierías Técnicas, 1 Arquitectura Técnica, 11 Diplomaturas y 9 Maestrías Especializadas, que actualmente se encuentran repartidas en 4 Campus Universitarios: el de Salamanca, el de Ávila, el de Béjar y el de Zamora.

Fruto de la constante renovación, la Universidad de Salamanca ha incorporado recientemente el Programa de Estudios Simultáneos que permite a los alumnos obtener simultáneamente dos titulaciones. En el curso 2000-2001 realizaron sus estudios casi 40.000 estudiantes. La Universidad mantiene convenios con importantes Universidades extranjeras, así corno con instituciones, fundaciones, empresas y sociedades tanto públicas como privadas.

2.2 La Facultad de Ciencias

Tal y como se recoge en el artículo octavo de la Ley de Orgánica de Universidades [BOCG, 2001], las Facultades son:

> "*...los centros encargados de la organización de las enseñanzas y de los procesos académicos, administrativos y de gestión conducentes a la obtención de títulos de carácter oficial y validez en todo el territorio nacional, así como de aquellas otras funciones que determinen los Estatutos.*"

En la Universidad de Salamanca las Facultades se rigen por los Estatutos publicados en el B.O.E. de 5 de julio de 1988 [BOE, 1988]. Sus tareas principales quedan establecidas en el artículo 16, en el que se le atribuyen:

- *La elaboración de sus Planes de Estudio.*
- *La organización y coordinación de las actividades docentes, así como la gestión de los servicios y medios de apoyo a la investigación y la enseñanza.*
- *La organización de las relaciones entre Departamentos y con otros Centros, a fin de asegurar la coordinación de la enseñanza y la racionalización de la gestión académica y administrativa.*
- *La expedición de certificados académicos y la tramitación de propuestas de convalidación, traslado de expedientes, matriculación y otras funciones similares.*
- *La administración de su presupuesto.*

La Universidad de Salamanca cuenta con 16 Facultades que completan un total de 26 Centros universitarios repartidos en 4 Campus, cubriendo ramas tan diversas como: Ciencias de la Salud, Experimentales, Humanidades, Jurídico-Sociales y Técnicas.

Dentro de la Universidad de Salamanca, la Facultad que se hace cargo de los estudios de Informática es la Facultad de Ciencias. Las titulaciones que imparte son [GAFC-USAL, 2001]:

- Diplomado en Estadística.
- Ingeniero Geólogo.

- **Ingeniero en Informática.**
- **Ingeniero Técnico en Informática de Sistemas.**
- Licenciado en Ciencias Físicas.
- Licenciado en Ciencias Geológicas.
- Licenciado en Matemáticas.

Estas titulaciones son impartidas por 190 profesores adscritos mayoritariamente a la Facultad de Ciencias. El número de alumnos que han cursado estudios en estas titulaciones ha descendido conforme a la disminución de la natalidad española. Así, en el curso 1999-2000 los alumnos matriculados ascendían a un total de 2.069, en el curso 2000-2001 este número descendía a 1.927 y el curso actual la cifra es de 1.900 alumnos.

2.3 El Departamento de Informática y Automática

Con la entrada en vigor de la L.R.U. [MEC, 1983] se produce un cambio en la estructuración y organización de las enseñanzas universitarias. Se establece el Departamento como unidad básica de la estructura universitaria. Concretamente el Artículo 8 del Título Primero establece que los departamentos son:

> "*... los órganos básicos encargados de organizar y desarrollar la investigación y las enseñanzas propias de su respectiva Área de Conocimiento en uno o varios centros.*"

La actual L.O.U [BOCG, 2001] recoge en su artículo noveno que:

> "*Los Departamentos son los órganos encargados de coordinar las enseñanzas de una o varias áreas de conocimiento en uno o varios centros, de acuerdo con la programación docente de la Universidad, de apoyar las actividades e iniciativas docentes e investigadoras del profesorado, y de ejercer aquellas otras funciones que sean determinadas por los Estatutos.*"

En la misma línea, los Estatutos de la Universidad de Salamanca [BOE, 1988], en el Capítulo Primero del Título Segundo señalan que:

> "*... los Departamentos son las unidades fundamentales de enseñanza e investigación de la Universidad.*"

Los Departamentos gozan de autonomía en la gestión y utilización de sus recursos personales y materiales; a ellos corresponde organizar y desarrollar la investigación y las enseñanzas de sus áreas de conocimiento en los diferentes centros en los que tiene asignada

docencia. Más detalladamente, las funciones que los Estatutos de esta Universidad asignan a los Departamentos se establecen en el Artículo 14, y son las siguientes:

- Confeccionar programas e impartir docencia en las áreas de conocimiento de su competencia, bajo la coordinación de los centros afectados.
- Programar y realizar proyectos de investigación.
- Planificar e impartir cursos de especialización, perfeccionamiento y actualización de los conocimientos científicos de los titulados universitarios y de sus propios miembros.
- Promover la participación y el asesoramiento en trabajos de carácter científico, técnico o artístico.
- Programar e impartir los cursos de doctorado, así como coordinar la elaboración y dirección de tesis doctorales.
- Fomentar programas de enseñanzas e investigación interdisciplinares e interdepartamentales.
- Organizar y llevar a cabo investigaciones acordadas en contratos suscritos con personas físicas, entidades públicas o privadas, nacionales o extranjeras.

La estructura universitaria organiza a los docentes e investigadores en Departamentos. Por tanto, para enmarcar las tareas de un profesor universitario se hace necesario la descripción de la unidad a la que pertenece. Así, el Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial está adscrita en la Universidad de Salamanca al Departamento de Informática y Automática.

2.3.1 Evolución del Departamento de Informática y Automática

La historia del Departamento de Informática y Automática es reciente, pues su andadura se inicia por resolución de la Junta de Gobierno de esta Universidad el 31 de Octubre de 1996.En sus inicios, estaba constituido por dos áreas de conocimiento: *Ingeniería de Sistemas y Automática* y *Lenguajes y Sistemas Informáticos*. Posteriormente, se incorporan las áreas de conocimiento de ***Ciencia de la Computación e Inteligencia Artificial*** y la de *Arquitectura y Tecnología de los Computadores*.

La creación y evolución del Departamento viene marcada por la aparición de nuevas titulaciones en la Universidad de Salamanca (Tabla 2.1) [GAU-USAL, 2001]. Así, hasta finales de los años 80, el área Ingeniería de Sistemas centraba su actividad docente en la impartición de asignaturas de en la titulación de Ciencias Físicas. Es a partir de 1988 cuando comienza a impartir asignaturas relacionadas con la Informática en dos nuevas titulaciones: La Diplomatura

de Biblioteconomía y la Diplomatura de Informática. Ello da lugar a que aparezca en 1990 el área de Lenguajes y Sistemas Informáticos, intentando satisfacer en un mejor grado las nuevas necesidades docentes e investigadoras de la Universidad de Salamanca.

CURSO	TITULACIÓN
1987/1988	Diplomatura en Biblioteconomía y Documentación
1989/1990	Ingeniero Técnico Industrial. Esp. Electrónica (Intensificación Electrónica)
1989/1990	Diplomatura de Informática
1992/1993	Licenciatura en Traducción e Interpretación
1993/1994	Ingeniería Química
1998/1999	2º ciclo Ingeniero Industrial
1998/1999	Ingeniero en Informática (2º ciclo)
1998/1999	Diplomado en Turismo

Tabla 2.1. Año de creación de nuevas titulaciones técnicas en la Universidad de Salamanca

Como consecuencia de la implantación de los nuevos Planes de Estudio, además de las nuevas titulaciones, se incluyen otras titulaciones en las que las áreas de Ingeniería de Sistemas y Lenguajes y Sistemas Informáticos imparten docencia, en asignaturas relacionadas en buena parte de los casos con la introducción a la Informática para sus estudiantes. Estas dos áreas sufren un importante incremento en el número de personal y en la carga docente, debido a que cada año se imparten nuevas asignaturas en diferentes titulaciones. De esta forma, un área a la que pertenecían tres personas en 1988 se convierte en un Departamento en 1996, que se completa con la creación en 1996 del Área de Ciencia de la Computación e Inteligencia Artificial y, en 1998, del área de Arquitectura y Tecnología de los Computadores.

Se puede ver, por tanto, a la década de los 90 como una época de gran dinamismo y crecimiento en la que, sin abandonar las tareas de investigación, los esfuerzos se han centrado en la puesta en marcha de las nuevas titulaciones, con un marcado carácter aplicado. Esto supone, no solamente un gran esfuerzo de trabajo, sino también de planteamiento en tanto que se produce una transición desde un entorno con un elevado carácter teórico hacia uno nuevo con un enfoque mucho más aplicado. Además, se puede señalar que tanto en las titulaciones existentes como en las de nueva creación, ha existido una importante dinámica de modificación de los Planes de Estudios que ha hecho de esta década, si cabe, aún más activa desde un punto de vista docente. Se puede establecer como consecuencia de lo descrito, que el apasionante ritmo de crecimiento, aunque pueda haber frenado en parte la actividad investigadora, ha sido enormemente enriquecedor, pues ha exigido de las personas que estaban involucradas, el estudio de muy diferentes aspectos de la Ciencia y Técnica, hecho que revierte tanto en labor docente como en la investigadora.

2.3.2 Actividad docente del Departamento

Actualmente, la plantilla del Departamento (Tabla 2.2) está integrada por 46 docentes, y dos personas del cuerpo de Personal de Administración y Servicios (un Auxiliar Administrativo y un Técnico Grado Medio en Informática) [DPTOIA, 2001].

En la Tabla 2.3 se muestran los Centros y las titulaciones donde el Departamento tiene docencia asignada, impartiendo asignaturas con contenido informático y otras relacionadas con la Ingeniería de Sistemas.

	CUERPO DOCENTE[3]					
Área de Conocimiento	**CU**	**TU**	**CEU**	**TEU**	**PE**	**Total**
Ingeniería de Sistemas y Automática	2	1	0	5	3	***11***
Lenguajes y Sistemas Informáticos	0	4	1	12	11	***28***
Ciencia de la Computación e Inteligencia Artificial	1	2	0	3	0	***6***
Arquitectura y Tecnología de Computadores	0	0	0	1	0	***1***
Total	***3***	***7***	***1***	***21***	***14***	***46***

Tabla 2.2. Profesorado del Departamento de Informática y Automática (en mayo de 2002)

La actividad docente del Departamento se completa con el Programa de Doctorado que en él se imparte. Esta tarea es un punto de conexión de gran importancia entre los dos objetivos principales de la Universidad: docencia e investigación.

Con la creación del Departamento de Informática y Automática se pone en marcha el Programa de Doctorado con el mismo nombre. Dicho Programa de Doctorado está orientado a la formación con profundidad en aspectos, tanto teóricos como prácticos, relativos a las materias englobadas en las Áreas de Conocimiento que pertenecen al Departamento. Los distintos cursos contienen en sus programas temas de actualidad, y en lo posible relacionados con el trabajo desarrollado por los grupos de investigación.

En la actualidad están vigentes los Programas correspondientes a los bienios 2000-2002 y 2001-2003 [USAL, 2000; DPTOIA, 2002]. En la Tabla 2.4 aparece el período de investigación

[3] **CU** = Catedráticos de Universidad, **TU** = Titulares de Universidad, **CEU** = Catedráticos de Escuela Universitaria, **PE** = Profesorado Extraordinario (Ayudantes + Asociados).

correspondiente al bienio 2000-2002 y en la Tabla 2.5 el período de docencia del bienio 2001-2003.

Centro	Titulación
Facultad de Ciencias Agrarias y Ambientales	**Licenciatura en Ciencias Ambientales**
Facultad de Ciencias	**Ingeniería Técnica en Informática de Sistemas**
	Ingeniero en Informática (2º Ciclo)
	Licenciatura en Físicas
	Licenciatura en Matemáticas
	Licenciatura en Geología
	Diplomatura en Estadística
Facultad de Ciencias Químicas	**Ingeniería Química**
Facultad de Derecho	**Diplomatura en Gestión y Administración de Empresas Públicas**
Facultad de Traducción y Documentación	**Diplomatura en Biblioteconomía y Documentación**
	Licenciatura en Documentación (2º ciclo)
	Licenciatura en Traducción e Interpretación
Escuela Universitaria de Educación (Ávila)	**Diplomatura de Turismo**
Escuela Técnica Superior de Ingeniería Industrial (Béjar)	**Ingeniería Técnica Industrial en Mecánica**
	Ingeniería Técnica Industrial en Electrónica
	Ingeniería Técnica Industrial en Electricidad
	Ingeniero Industrial (2º Ciclo)
Escuela Politécnica Superior (Zamora)	**Ingeniería Técnica Industrial (Especialidad Mecánica)**
	Ingeniería Técnica en Obras Públicas (Especialidad Construcciones Civiles)
	Ingeniería Técnica Agrícola (Especialidad Industrias Agrarias y Alimentarias)
	Arquitectura Técnica

Tabla 2.3. Titulaciones en las el Departamento de Informática y Automática imparte docencia

Título del curso	Créditos
Sistemas Avanzados de Producción	6
Control de Procesos	6
Implementación de Algoritmos IA para robots móviles	6
Implementación de Algoritmos de Detección de Colisiones para Estructuras Robóticas	6
Optimización Dinámica de la Síntesis, el Control y la Operación de las Industrias de Procesos	6
Planificación de la Producción mediante Técnicas Analíticas y Técnicas de IA de Procesos Continuos y por Lotes	6
Fuentes de Información en Seguridad en Internet	6
Computación No Algorítmica	6
Trabajo de Iniciación a la Investigación en Lenguajes y Sistemas Informáticos	6
Minería de Datos	12
Ingeniería del Software Basada en Componentes	12
Trabajo de Iniciación a los Agentes y Sistemas Multiagente	12
Trabajo de Investigación en Ingeniería de Sistemas y Automática	12
Trabajo de Investigación en Lenguajes y Sistemas Informáticos	12

Tabla 2.4. Periodo de Investigación (Bienio 2000-2002)

Título del Curso	Créditos
Computación Soft (*)	4
Tecnología de Objetos Aplicada a la Creación de Aplicaciones Distribuidas y de Tiempo Real (*)	3
Ingeniería del Software Avanzada (*)	3
Reconocimiento Automático de Habla (*)	3
Diseño de Interfaces Gráficas	3
Optimización de Procesos	4
Sistemas Avanzados de Producción	4
Protección de la Información en Internet	4
Metodología de la Investigación en Informática	1
Minería de Datos	3
Control de Procesos Clásico y Avanzado	4
Técnicas de Programación Paralela	3
Extensiones de la lógica de primer orden	3
Teoría de la Agencia: Agentes y Sistemas Multiagente (*)	3

Tabla 2.5. Cursos del programa de doctorado de Informática y Automática (Bienio 2001-2003)

Con objeto de promover la formación de nuevos investigadores, potenciar la calidad de la investigación en el campo de la Informática y las Telecomunicaciones (TI y T), y mejorar la coordinación de la actuación conjunta de las cuatro Universidades de la comunidad de Castilla y León en actividades de I+D, los Departamentos de Informática y Telecomunicaciones de las cuatro Universidades tienen en proyecto ofrecer un programa de doctorado conjunto en Tecnologías de la Información y las Telecomunicaciones. Dicho Programa de Doctorado en TI y T será un programa interuniversitario, único y común para las cuatro universidades, en el que éstas participarán en igualdad de condiciones y derechos. El programa se ofertará como propio en cada una de las Universidades, simultáneamente a otros de ámbito exclusivo, y habilitará para la obtención del título de Doctor. Para la participación de los alumnos en los cursos y seminarios correspondientes al Periodo Docente, se establece un sistema de Aula Virtual por medio de multi-videoconferencia en cada una de las universidades.

2.4 El área de conocimiento de Ciencia de la Computación e Inteligencia Artificial

El área de Ciencia de la Computación e Inteligencia Artificial tiene en esta Universidad una corta pero intensa andadura, cuyo germen se vincula a la creación del actual Departamento de Informática y Automática.

Actualmente, al área de conocimiento está integrada por 6 personas[4] (5 de los cuales son doctores), adscritas todas ellas a la Facultad de Ciencias. Como ya contemplaba la derogada LRU [MEC, 1983] y, actualmente, contempla la LOU [BOCG, 2001], la adscripción de una plaza de profesorado a un centro tiene unos efectos que, en muchos casos, se limitan a los puramente administrativos. No existen profesores adscritos a la Facultad de Ciencias Agrarias y Ambientales, aunque el área tenga asignada docencia con carácter troncal en una de las titulaciones impartidas por dicha Facultad: Licenciatura en Ciencias Ambientales.

A continuación, se va a realizar una descripción de los aspectos docentes e investigadores de esta Área de Conocimiento en la Universidad de Salamanca.

2.4.1 Labor docente

La labor docente del área de Ciencia de la Computación e Inteligencia Artificial en la Universidad de Salamanca se encuentra fundamentalmente ligada a los estudios de Informática

[4] 1 Catedrático de Universidad, 2 Titulares de Universidad y 3 Titulares de Escuela Universitaria.

en dicha Universidad, con especial énfasis en las titulaciones de Ingeniería Técnica en Informática de Sistemas e Ingeniería Informática, ambas impartidas en la Facultad de Ciencias.

CENTRO	TITULACIÓN	CURSO	ASIGNATURA	CARÁCTER	CRÉDITOS
Facultad de Ciencias	Ingeniería Técnica en Informática Sistemas	1º	Algoritmia	Troncal	7,5
		1º	Programación	Troncal	6
		1º	Sistemas Informáticos	Obligatorio	6
		1º	Laboratorio de Programación	Obligatorio	4,5
		2º	Diseño de Bases de Datos	Troncal	4,5
		3º	Redes	Troncal	7,5
		3º	Ingeniería del Software	Obligatorio	6
		3º	Programación Orientada a Objetos	Optativo	6
		3º	Proyecto	Obligatorio	9
	Ingeniería Informática (2º ciclo)	1º	Análisis de Sistemas	Troncal	9
		1º	Procesadores de lenguaje	Troncal	9
		1º	Redes	Troncal	9
		2º	Ampliación de Bases de Datos	Obligatorio	6
		2º	Informática Gráfica	Optativo	6
		2	Proyecto	Troncal	6
	Licenciatura en Matemáticas	1º	Informática	Troncal	9
		4º	Ampliación de Informática	Optativo	7,5
Facultad de Ciencias Agrarias y Ambientales	Licenciatura en Ciencias Ambientales	1º	Informática	Troncal	4,5

Tabla 2.6. Asignaturas impartidas por el área de Ciencia de Computación e Inteligencia Artificial en la Universidad de Salamanca

En la Tabla 2.6 se recogen las asignaturas impartidas por el área en la Universidad de Salamanca.

Respecto al Programa de Doctorado, el área se encarga de impartir 5 cursos de los 14 de los que componen el periodo de docencia. Concretamente, son los que en la Tabla 2.5 están marcados con (*). En ellos se cubren diferentes temas de interés del área y hacia los que se dirige principalmente la investigación.

2.4.2 Labor investigadora

Las personas que integran el Área de Conocimiento de Ciencia de la Computación e Inteligencia Artificial desarrollan sus actividades investigadoras en las principales líneas de investigación del Departamento de Informática y Automática, entre las que cabe destacar [DPTOIA, 2001]:

- Redes Neuronales para reconocimiento de Patrones.
- Ingeniería del Conocimiento y Minería de Datos.
- Sistemas de Agentes y Multiagentes.
- Ingeniería de Software basada en componentes: aspectos de reusabilidad, arquitectura multinivel, integración de métricas.
- Calidad del software.
- Proceso de Lenguaje Natural en Español.
- Diseño de software para sistemas de fabricación flexible.
- Redes Activas: desarrollo y aplicaciones.
- Comercio Electrónico e Ingeniería Web.

El nivel de la investigación realizada se puede situar desde aspectos de investigación básica, donde se realiza la propuesta de formalismos y modelos con un marcado carácter teórico, hasta aspectos de investigación aplicada que se centran en el desarrollo de herramientas para su inmediata aplicación práctica.

Durante los últimos años, las labores de investigación se han desarrollado a través de la dirección y participación en Proyectos de Investigación propios del Departamento, coordinados con empresas y/o grupos de otras Universidades españolas, en Redes de investigación europeas, Programas de intercambio... Asimismo, la dirección de un conjunto de tesis doctorales, trabajos de grado (tesinas), proyectos fin de carrera... completan el historial investigador de este grupo de personas.

Hay que destacar que en la mayoría de las tareas investigadoras no participan exclusivamente personas de esta área, sino que también participan personas procedentes de las otras áreas del Departamento, fundamentalmente de las áreas de Lenguajes y Sistemas Informáticos e Ingeniería de Sistemas y Automática.

2.5 La infraestructura

Los medios materiales de apoyo a la docencia y a la investigación con que cuentan los estudios de Informática se encuentran en un proceso de continua actualización, tanto en calidad de los equipos como en su cantidad. Esta renovación es absolutamente necesaria debido a la rápida evolución que sufren los sistemas informáticos, que en un período de unos pocos años pasan a estar completamente obsoletos.

Para impartir las clases teóricas y de problemas, la Facultad de Ciencias cuenta con aulas dotadas con proyectores de transparencias y de diapositivas, a parte de la clásica pizarra. Además, dispone, en la actualidad, de seis aulas de ordenadores, con aproximadamente 30 ordenadores por aula. Cinco de estas aulas poseen computadores compatibles IBM, mientras que los de la otra son Macintosh. Todos ellos se encuentran conectados entre sí mediante una red de área local de tipo Ethernet 10BaseT o 100BASETX.

Para la realización de las prácticas específicas de la Ingeniería Técnica en Informática de Sistemas, los alumnos tienen acceso, desde las aulas de Informática, a diferentes estaciones de trabajo entre las que destacan:

- Una estación HP 9000/821 D250, con 288 MB de RAM y 15 GB de disco duro. El software más relevante instalado en esta máquina es:
  - Sistema Operativo HP-UX B.11.00
  - Compiladores de C y Fortran
  - Sistema Gestor de Base de datos INGRES 2.0
- Una estación HP 715/33, con 80 MB de RAM y 2 GB de disco duro.
- Una estación SUN Enterprise 250-Server, con microprocesador Ultra Sparc II a 400 MHz, 1 GB de RAM y dos discos duros de 18 y 36 GB respectivamente.

Las aulas de informática, gestionadas por dos técnicos especialistas, están a disposición de los alumnos que cursan alguna titulación de la Facultad de Ciencias, tanto durante el horario de prácticas de las asignaturas como en el tiempo asignado como de libre utilización, y en este último caso a disposición de cualquier alumno de la Universidad de Salamanca. Pueden trabajar

en los diferentes entornos y hacer uso de diversos lenguajes de programación y de las aplicaciones disponibles. Estas aulas cuentan además con un cañón de proyección que facilita enormemente la tarea del profesor en las prácticas guiadas.

Además, los estudiantes del último curso de la Ingeniería Técnica pueden hacer uso del Laboratorio de Proyectos para la realización de su proyecto fin de carrera. Cuenta con cuatro estaciones Silicon Graphics MIPS R4600 con 96 MB de RAM, 2,5 GB de disco, S.O. IRIX 6.2, cinco Pentium, dos Pentium II, un Macintosh 7600 PowerPC 604, un iMAC, escáner e impresoras.

Por otra parte, los alumnos de la Ingeniería Informática disponen de un laboratorio para su uso exclusivo, con el siguiente equipamiento:

- Un servidor UNIX Silicon Origin200, Tetraprocesador MIPS RISC R10000 de 64 bits:
  - Memoria RAM de 769 MB
  - Disco interno de 18 GB Ultra Fast Wide SCSI (9 + 9 GB)
  - 2 tarjetas Ethernet 10BASET/100BASETX
  - Interfaz Ultra Fast/Wide SCSI-2 a 40 MB/s
  - 3 slots PCI y 6 slots XIO de expansión
  - Destacando el siguiente software instalado:
    - Sistema Operativo IRIX 6.5
    - Software de desarrollo de Silicon Graphics Varsity versión Nodelock
    - Sistema Gestor de Base de Datos ORACLE 8
    - Herramientas de diseño y desarrollo de aplicaciones: ORACLE DEVELOPER/2000 y ORACLE DESIGNER/2000
- Un servidor NT Pentium II de Fujitsu:
  - Doble procesador Pentium II a 400 MHz
  - 512 Kb de caché por procesador
  - Disco duro Ultra Wide SCSI de 9 Gb
  - 128 MB de memoria SDRAM
  - Adaptador gráfico SVGA AGP con 2 MB de memoria
  - 2 adaptadores de red Ethernet Intel Etherexpress 100BASETX/10BASET PCI

    - Sistema Operativo Windows NT Server, version 4.0
- 25 PC´s Pentium II a 266 MHz
    - 64 MB de memoria DIMM SDRAM
    - Dos discos duros Ultra DMA de 3,2 GB y 6 GB respectivamente
    - Tarjeta de red 3COM Fast Ethernet 10/100
    - Sistemas Operativos Windows NT WorkStation y Linux
- Red Fast Ethernet 100BASETX con los siguiente elementos de red:
    - Armario de cableado estructurado
    - 1 Hub 100BASETX de 3COM (Office Connect 4 TP400 100 BTX NG)
    - 1 Switch 3300 de 3COM 24 port 10/100
    - 1 Router 3COM SuperStack II Netbuilder 432

Además, el Departamento, y en concreto el área de Ingeniería de Sistemas y Automática, tiene a su cargo dos laboratorios para impartir las clases prácticas de las titulaciones donde tiene asignada docencia, para la realización de proyectos fin de carrera y labores de investigación. Respecto al primero, el Laboratorio de Robótica, dispone del siguiente equipamiento

- Una estación SUN ULTRA SPARC II Creator, con 128 MB de RAM y 4 GB de disco, destacando el siguientes software instalado:
    - S.O. Solaris 2.7
    - Matlab 5.3 Release 11
    - ACSL 11.4.1
- Una estación SUN SPARC Station 5, con 64 MB de RAM y 7GB de disco, S. O. Solaris 2.5.
- Siete Pentium (>100 MHz), Sistemas Operativos Windows, Linux, SCO, RT-Match (Tiempo Real).
- Una Célula CIM (*Computer Integrated Manufacturing*) formada por:
    - 2 Robots tipo PUMA
    - Cinta transportadora (Bosch)
    - Un autómata programable (SLC-500) de Allen Bradley, con una carta de ejes IMC-110, varios módulos de entradas y salidas analógicas y digitales

    - o Sistema de visión integrado por una cámara SONY, una tarjeta de procesamiento de imagen Matrox IMAGE-LC y procesador digital de señal DicomLab

- Un robot móvil AmigoBot de la empresa ActivMedia.
- Cuatro módulos de la serie Robotics Invention MindStorm de la marca Lego.

En cuanto al segundo, el Laboratorio de Automática, dispone del siguiente material:

- Seis Pentium (>100 MHz) con sistemas operativos Windows, Linux y RT-Linux.
- Dos maquetas de control de motores DC, fabricadas por Feedback Instruments.
- Tres autómatas programables y sus correspondientes tarjetas de E/S fabricados por Siemens y Telemecanique-Schneider.
- Cinco maquetas de la marca Tecquipment que incluyen viga y bola, bola y aro, tanques acoplados, un motor de cuatro tiempos y una de control multivariable.
- Una planta de control de nivel, construida con instrumentación industrial.

Además, de los medios informáticos, la utilización de las bibliotecas debería ser práctica habitual de los alumnos, puesto que la consulta de libros y publicaciones es un complemento fundamental a la enseñanza recibida en las clases. En concreto, los estudiantes pueden hacer uso de las diferentes bibliotecas de la Universidad de Salamanca, y en particular de la biblioteca Abraham Zacut, ubicada en el Campus de Ciencias, que es la que dispone de la mayor parte de los textos relacionados con las materias que se imparten en las titulaciones de Informática, además de contar con terminales de acceso a Internet para realizar consultas.

2.6 El alumnado

A la hora de analizar el entorno educativo no puede olvidarse el elemento clave en la actividad universitaria: los alumnos. A ellos va dirigida la labor del docente, y son ellos quienes han de cubrir los objetivos y realizar las actividades oportunas. Por tanto, es importante tener un conocimiento del alumnado, y para ello es necesario hacer un estudio previo de sus características y sus motivaciones.

Desde que comenzó a impartirse la Diplomatura de Informática en la Universidad de Salamanca, curso 1989/1990, el número de solicitudes de matrícula ha sido muy elevado, tal y como queda reflejado en los datos de la Tabla 2.7. Sin embargo, la Universidad no podía atender esta fuerte demanda si quería ofrecer unos mínimos de calidad en su enseñanza. Por esta razón el número de admitidos se limitó inicialmente a 105, y esta es la cifra que,

aproximadamente, se mantuvo en años sucesivos. Debido a la demanda social, en el presente curso académico el número de estudiantes se ha limitado a 150.

	89/90	90/91	91/92	92/93	93/94	94/95	95/96	96/97	97/98	98/99	99/00	00/01	01/02
Solicitudes	***476***	***297***	***356***	***517***	***576***	***531***	***573***	***548***	***470***	***428***	***339***	***576***	***628***

Tabla 2.7. Número de solicitudes de matrícula en las que se eligen, en primera opción, los estudios de Informática (1er ciclo)[5]

Inicialmente, los estudios de Informática estaban restringidos a alumnos que hubieran cursado C.O.U. En el curso 1992/1993 se amplió también a estudiantes procedentes de la Formación Profesional. La proporción de plazas asignadas a unos y otros viene en la actualidad establecida por las disposiciones legales vigentes, que las distribuyen del siguiente modo:

- 1% de titulados universitarios.
- 3% de países no comunitarios, ni pertenecientes al espacio económico europeo.
- 3% de discapacitados.
- 3% de deportistas de élite.
- 3% de mayores de 25 años.
- 22% de alumnos procedentes de Ciclos Formativos de Grado Superior.
- 8% de alumnos procedentes de Módulos Profesionales y Formación Profesional de II Grado.
- El 57% restante de alumnos no contemplados en los apartados anteriores.

En el proceso de admisión tienen preferencia los alumnos procedentes de la opción Científico-Técnica: Opción A para los alumnos procedentes de COU, opciones 3, 4 y 5 para los de bachillerato experimental y opción 1 para los de LOGSE.

El Real Decreto 69/2000, de 21 de enero, establece que las Universidades reservarán al menos un 20% de las plazas ofertadas para los estudiantes que vayan a cursar el primer ciclo de estudios universitarios, con independencia de la Universidad en que hayan superado la prueba de acceso. Las Universidades públicas de Castilla y León ofertan, por la modalidad de distrito abierto, el 100% de las plazas en las titulaciones que tienen fijados límites a la admisión de alumnos.

[5] Los datos de las tablas han sido facilitados por el Centro de Proceso de datos de la Universidad.

En función de la distribución, de la demanda y de la limitación anual de matriculados antes mencionada, se establecen cada año unas calificaciones mínimas para poder cursar esta titulación. Estas calificaciones son las que se muestra en la Tabla 2.8.

	92/93	93/94	94/95	95/96	96/97	97/98	98/99	99/00	00/01	01/02
P.A.U.	5,93	6,04	5,60	6,04	6,14	6,46	6,48	6,65	6,76	6,86
F.P. II	6,00	6,75	6,54	6,70	6,37	6,19	6,18	6,03	5,64	6,02
Ciclos formativos									6,6	5,8
Titulados	1,58	1,62	1,90	1,12	1,70	1,79	1,96	1,52	1,76	1,84
> 25 años									6	5,02

Tabla 2.8. Calificaciones mínimas necesarias para cursar los estudios de Informática de 1er ciclo

La titulación de Ingeniero en Informática (2º ciclo) comenzó a impartirse en el curso académico 1998/1999, siendo el número de solicitudes, como en el caso de los estudios de 1er ciclo, muy superior al número de admitidos, que en este caso es únicamente de 40 alumnos por curso. Los datos sobre solicitudes de matrícula y puntuaciones mínimas de acceso se recogen en la Tabla 2.9.

	98/99	99/00	00/01	01/02
Número de solicitudes	82	88	80	75
Puntuación mínima	1,49	1,44	1,31	1,22

Tabla 2.9. Solicitudes y puntuaciones mínimas relativas a los estudios de Informática de 2º ciclo

Para acceder al 2º ciclo se debe estar en posesión del título de Diplomado en Informática, Ingeniero Técnico en Informática de Sistemas o de Gestión. Debido a la limitación de matriculados, se consideran de forma prioritaria los solicitantes que hayan cursado en la Universidad de Salamanca estos estudios de primer ciclo, los titulados por otras Universidades donde no se imparta esta titulación de 2º ciclo y los estudiantes que acrediten desempeñar una actividad laboral en el ámbito territorial de la Universidad de Salamanca, que incluye a las provincias de Ávila, Salamanca y Zamora.

Otro dato interesante a la hora de hacer la programación docente es el número de alumnos por curso. La Tabla 2.10 muestra el número de alumnos por curso y el total, en la Ingeniería Técnica en Informática de Sistemas, mientras que la Tabla 2.11 hace lo propio para la Ingeniería en Informática, en los últimos años. Se ha seguido el criterio de que un alumno es del curso

superior de todos los que tenga matriculadas asignaturas. Estos datos también son útiles para hacer un análisis de la situación.

	95/96	96/97	97/98	98/99	99/00	00/01	01/02
Primer Curso	127	119	142+2	120	121	132	173
Segundo Curso	122	132	107	130+5	122	124	125
Tercer Curso	105	176	205	237	95+152	170+91	204+36
TOTAL	354	427	456	492	490	517	538

Tabla 2.10. Número de alumnos por curso y el total en la Ingeniería Técnica en Informática (La línea negra refleja la implantación del Plan de 1997)

	98/99	99/00	00/01	01/02
Primer Curso	40	47	50	55
Segundo Curso	-	29	44	45
TOTAL	40	76	94	100

Tabla 2.11. Número de alumnos por curso y el total en la Ingeniería en Informática

Teniendo en cuenta todos estos datos se puede llegar a las siguientes conclusiones:

- Buena motivación para cursar estos estudios

  El interés por realizar este tipo de estudios es muy grande, hay una demanda mucho mayor que el número máximo de admitidos, por lo que sólo consigue cursarlos una parte de los alumnos que los han elegido como primera opción. Esto presupone, en principio, que existe una buena motivación por su parte. Sin embargo, en el caso de la titulación media, también hay que tener en cuenta que es una carrera que resulta atractiva por el hecho de que sean estudios de ciclo corto, así como por la incorporación al mundo laboral de este tipo de titulados es, en términos generales, mayor que la de los procedentes de una gran parte de las disciplinas universitarias.

- Mejor preparación y capacidad

  Las calificaciones mínimas de acceso a la titulación de primer ciclo muestran una clara tendencia creciente. De ello puede deducirse que la preparación y la capacidad de los estudiantes son, en principio, cada vez mejores. Las puntuaciones

requeridas para acceder a la titulación de Ingeniero en Informática revelan que estos alumnos tienen una sólida base para realizar los estudios superiores.

- Grupos no homogéneos

  En la Ingeniería Técnica en Informática de Sistemas, la diversa procedencia de los alumnos hace que los grupos no sean homogéneos en cuanto a su formación inicial. En general, los alumnos procedentes de la LOGSE tienen una mejor base teórica en materias tales como matemáticas o física, pero menos específica en informática. Aunque cada vez es menos frecuente, en ocasiones sus conocimientos iniciales en informática son prácticamente nulos. En cambio, los alumnos de F.P. llegan con una mayor preparación práctica y un conocimiento en las asignaturas de contenido informático, si bien presentan una falta de base matemática. Su mentalidad va dirigida a la práctica profesional específica y, en general, no muestran interés en los fundamentos teóricos. En cuanto a los titulados y extranjeros, suelen tener conocimientos en informática y bastante interés en su aprendizaje, pero son dos grupos muy poco significativos dentro del total de alumnos del curso. Esta falta de homogeneidad supone una dificultad para el docente, particularmente en las asignaturas de contenido fuertemente teórico.

  En el segundo ciclo, aunque la mayoría de los alumnos provienen de la Universidad de Salamanca, hay un pequeño grupo de alumnos procedentes de otras Universidades, en las que los Planes de Estudio son diferentes, por lo que es necesario hacer un esfuerzo para que estos alumnos puedan seguir aquellas asignaturas que puedan tener ciertas dificultades.

- Elevado número de alumnos por clase

  A diferencia de lo que ocurre en la titulación superior, en los tres cursos de la Ingeniería Técnica el número de alumnos por clase es bastante elevado; esto dificulta la labor docente, ya que impide la utilización de ciertos métodos o recursos didácticos. Condiciona de forma importante la participación activa durante las clases y también la realización de prácticas.

2.7 El marco curricular

2.7.1 Perspectiva histórica de los estudios de Informática

La investigación en máquinas capaces de calcular de manera automática ha tenido gran actividad desde antes del siglo XX, pero es en la década de los 40, principalmente como respuesta a las necesidades militares, cuando surgieron las primeras computadoras electrónicas. Aunque fueron desarrolladas en entornos universitarios, hasta mediados de la década de los 50 no aparecen los primeros centros de computación universitarios. Generalmente estos centros estaban vinculados a Departamentos de Matemáticas o Ingeniería, y servían de soporte a la investigación en los mismos.

Los primeros programas académicos en Instituciones Superiores de educación aparecieron a mediados de la década de los 50. Su principal orientación era la formación de los usuarios de computadoras y por ello su contenido consistía en temas relacionados con el manejo de los equipos. Entre estos centros cabe destacar las siguientes Universidades: *University of Michigan*, *University of Houston*, *Stanford University* y *Purdue University*. En Europa la educación en Informática se desarrolló más o menos de forma simultánea a la de EE.UU. Algunas de las primeras computadoras fueron instaladas, e incluso construidas, en las propias Universidades, con el fin de servir a los propósitos de investigación de los diferentes Departamentos. A partir de 1965 comienzan a aparecer titulaciones de Informática en Gran Bretaña, Francia y Alemania.

La primera computadora instalada en España fue una IBM 650 (modelo del año 1953) en la Compañía Nacional Telefónica, en el año 1958. En 1962 empezaron a introducirse las primeras computadoras en empresas privadas. En esta época, la enseñanza del manejo y fundamentos de estas máquinas corrió a cargo de las propias empresas constructoras.

En el año 1969 comienza la enseñanza de Informática de manera oficial en España con la creación del **Instituto de Informática** en Madrid. En él se imparte una carrera de cinco años, de los que los tres primeros son comunes y los dos últimos de especialización. En 1971 se crea una delegación de este Instituto en San Sebastián, y en 1972 se forma en la Universidad Autónoma de Barcelona un Departamento de Informática. De forma paralela, se incluyen en diversas titulaciones universitarias de carácter técnico o científico asignaturas específicas de Informática. A finales de los años setenta se empiezan a impartir Estudios Superiores de Informática en la Universidad. Éstos fueron concebidos como diplomaturas y licenciaturas. Actualmente, según las directrices del Ministerio de Educación y Ciencia sobre nuevas titulaciones, se han reconvertido las titulaciones a Ingenierías Técnicas y Superiores.

En el ámbito internacional, en la década de los 60, algunas universidades norteamericanas (*Harvard, MIT, Columbia, Pensylvania*) incorporaron currículos completos de Informática dentro de sus enseñanzas. En ellos se plantearon dos enfoques, por un lado el que se denominó "*Computer Science*", más orientado a la programación y a la algoritmia, y por otro lado el enfoque conocido como "*Computer Engineering*" enfocado hacia la arquitectura y tecnología de computadores.

Desde estas fechas hasta nuestros días se han realizado numerosos esfuerzos por redactar un documento que unifique los diferentes currículos en Informática. Este documento debería ser lo suficientemente flexible para poder adaptarlo a situaciones particulares de disponibilidad de medios materiales, entorno económico-social, plantilla de profesorado... Existen dos sociedades destacadas en éste esfuerzo: **ACM** (*Association of Computer Machinery*) – http://www.acm.org, e **IEEE-CS** (*Institute of Electrical and Electronic Engineers – Computer Society*) – http://computer.org. Como resultado de esta iniciativa surge el *Computing Curricula 1991*, **ACM/IEEE-CS'91** [Tucker et al., 1990], referencia obligada en la creación de Planes de Estudio de Informática[6], y recientemente (diciembre de 2001) el *Computing Curricula 2001 Computer Science* [ACM/IEEE-CS, 2001].

Del repaso de los diferentes currículos de ámbito internacional que se han generado a lo largo de estos años, se desprende, que lo que en España se engloba bajo el título de Informática se corresponde con diversas disciplinas en el mundo anglosajón. Las más extendidas y aceptadas actualmente son: *Computer Science, Computer Engineering, Information Systems* y *Software Engineering*. Cada una de estas titulaciones puede tener diferentes enfoques en distintas Universidades.

Los conceptos que se ocultan detrás de estos nombres no tienen una significación común para todo el mundo, siendo su interpretación más adecuada la siguiente [Camps, 1999]:

- **CS – *Computer Sicence* – Ciencia de la Computación:** Se trata de la Informática como disciplina científica construida sobre fundamentos lógicos y matemáticos. El científico CS es un investigador que desarrolla principios fundamentales, teorías y herramientas formales. Con frecuencia se usa el término CS en un sentido más amplio, casi sinónimo a nuestro término "Informática". Pero la palabra *Science* no ofrece mucha diversificación. Así, se está adoptando el término *Computing* como vocablo global polivalente, más cercano a la palabra Informática.

[6] De hecho, los actuales Planes de Estudio de Informática de la mayoría de las Universidades españolas se automanifiestan inspirados en las recomendaciones recogidas en el ACM/IEEE-CS'91 [Camps, 1999].

- **CE – *Computer Engineering* – Ingeniería de los computadores:** Ingeniería relativa al diseño y construcción de herramientas informáticas, tanto hardware como software. Aplica los principios fundamentales de la Ciencia de la Computación.
- **SE – *Software Engineering* – Ingeniería del Software:** Actualmente el término CE se tiende a restringir a la ingeniería del hardware, utilizando el término SE para la ingeniería del software.
- **SI – *Information Systems* – Sistemas de Información:** Se trata de la aplicación de la tecnología informática (por ejemplo herramientas software desarrolladas por ingenieros del software) a la gestión de la información en el mundo de las organizaciones (empresas, instituciones). Abarca un campo que va desde la planificación estratégica de las tecnologías de la información en la empresa, hasta el diseño/programación de aplicaciones de gestión, es decir comparte terreno con la Ingeniería del Software y la Administración de Empresas.

Las propuestas del *Computing Curricula 1991* y del *Computing Curricula 2001* se centran en recomendaciones para Planes de Estudios de Informática (*Computing*) que incluyen planes para Ciencia de la Computación, Ingeniería de Ordenadores, Ciencia de la Computación e Ingeniería del Software o similares, quedando fuera los Sistemas de Información.

Pero hay sectores que abogan por la separación de la parte de Ciencia de la Computación de la parte de Ingeniería Informática (Ingeniería del Software), argumentando que ambas tienen orientaciones distintas por mucho que se influyan y fertilicen mutuamente [Lutz y Naveda, 1997], llegando a cuestionar incluso la validez de cara al futuro de la propuesta curricular ACM/IEEE-CS'91 como núcleo común entre la Ciencia de la Computación y la Ingeniería Informática, abogando por propuestas curriculares independientes [Bagert, 1999]. Este hecho se ha puesto de manifiesto en el nuevo *Computer Curricula 2001*, donde se aboga por establecer un modelo curricular para las principales corrientes derivadas de la Informática como única forma de recoger el tremendo crecimiento, evolución y explosión sufrida por esta disciplina desde la revisión de 1991. De hecho, el [ACM/IEEE-CS, 2001] se refiere sólo a la rama de Ciencia de la Computación, previendo informes similares para *Computer Engineering*, *Software Engineering* e *Information Systems*.

Quizás uno de los mayores defensores de esta división sea David Lorge Parnas que expresa sus dudas de que la formación que se está ofreciendo a los informáticos sea la adecuada para ejercer su labor profesional, necesitándose una correcta definición de la profesión, un reconocimiento de la Ingeniería del Software como una nueva rama de la Ingeniería, una

identificación del cuerpo de conocimiento propio de la Ingeniería del Software y una comunicación más efectiva entre las partes en conflicto, de manera que se solucionen las carencias de los ingenieros del software a la hora de aplicar los formalismos oportunos en su labor profesional y se palie el desconocimiento de los científicos sobre lo que significa ser ingeniero [Parnas, 1997; Parnas, 1999]. El sentir de David L. Parnas puede resumirse en la siguiente cita:

> *"La Ingeniería consiste en el uso de la Ciencia y la Tecnología para construir productos que serán utilizados por otras personas. El software es uno de esos productos. Los ingenieros encuentran sus problemas en la práctica. Los científicos en la literatura. Un 'científico' ve una Máquina de Estados Finita como un modelo de computación, mientras que un 'ingeniero' ve en ella una herramienta de diseño. La mayoría de los científicos que estudian la 'ciencia de la programación' no entienden lo que es un ingeniero. Porque la Ingeniería del Software se basa en la Ciencia de la Computación, creen que la Ingeniería del Software es Ingeniería de los Ordenadores. Esto sería como confundir la Ingeniería Electrónica con la Física. Actualmente la mayoría de titulaciones de Ciencias de la Informática no son titulaciones de ciencias puras, pero tampoco pueden acreditarse como ingenierías. Son soluciones de compromiso y no hacen ninguna de las dos cosas correctamente".*
>
> **David Lorge Parnas**. "*Software Engineering: An Unconsummated Marriage*". Tutorial en el ESEC'97. Agosto, 1997.

Pero como era de imaginar no todo el mundo está de acuerdo con esta separación tan drástica de la Ciencia de la Computación y la Ingeniería del Software, buscando una definición de la profesión de informático (*computing*) como una nueva ingeniería única, no separando las áreas de conocimiento, basada en una nueva propuesta curricular que tome como punto de partida el ACM/IEEE-CS'91 (ahora el *Computing Curricula 2001*), y que unifique un marco de acreditación para los titulados (*ingenieros*) [El-Kadi, 1999].

Como resumen de la postura no separatista puede citarse a Peter J. Denning, quizás una de las personas que más influyeron en el *Computing Curricula 91*, y presidente del *ACM Education Borrad* en el *Computing Curricula 2001*, que expone en [Denning, 1998]:

> *"La separación entre la teoría y la ingeniería ha sucedido en otras disciplinas porque éstas habían madurado lo suficiente para que hubiera una comunicación fluida entre sus ramas científica, ingenieril y de aplicación. Una separación similar sería un desastre en la Ciencia de la Computación. La segregación de los ingenieros del software provocaría*

que la comunicación entre ingenieros, teóricos y especialistas en aplicaciones se acabase. La comunicación, no el divorcio, es la respuesta".

2.7.2 Las titulaciones de Informática a partir de la reforma universitaria

Las enseñanzas universitarias en nuestro país están pasando en la actualidad por una fase de renovación, que se inició en 1983 con la L.R.U. [MEC, 1983] y continúa actualmente con la L.O.U [BOCG, 2001; BOE, 2001]. Los dos postulados básicos en el proceso de reforma han sido:

- La vertebración de las enseñanzas universitarias en una estructura cíclica que permita la obtención de sucesivos títulos oficiales consecutivos.
- La redefinición de los contenidos formativos y las exigencias académicas de los Planes de Estudios, intentando con ello acercar la formación universitaria a la realidad social y profesional de nuestro entorno.

Las directrices generales comunes de los Planes de Estudios se presentan en el Real Decreto 1497/1987 de 27 de noviembre, publicado en el B.O.E. de 14 de diciembre [BOE, 1987][7]. Con ellas se busca una mayor flexibilidad de sus fórmulas y soluciones académicas que permitan una mayor rentabilidad de la oferta universitaria, un mejor aprovechamiento del discente y un más amplio abanico de opciones. Para conseguir este fin se plantea:

- La racionalización de la duración de las carreras y de la carga lectiva, hasta el momento excesiva.
- La convicción de que la enseñanza práctica debe asumir una mayor relevancia en la Universidad.
- La incorporación de un sistema de cómputo mediante créditos, que potencia una mayor flexibilidad en el currículo del estudiante.

Resultan así unos estudios caracterizados por ser:

[7] Rectificaciones en BOE nº 299, de 15 de diciembre de 1987. Este Real Decreto trataba de conciliar los objetivos científicos propios de la institución universitaria, resumidos en la nueva LOU en su artículo 1.1 cuando afirma que "*La Universidad realiza el servicio público de la educación superior mediante la investigación, la docencia y el estudio*", con la necesaria preparación del estudiante para una futura vida profesional.

- ***Cíclicos***, es decir, estructurados, como máximo, en tres ciclos. La superación del primero de ellos dará derecho, en cada caso, a la obtención del título de Diplomado, Ingeniero Técnico o Arquitecto Técnico. El segundo corresponderá al título de Licenciado, Ingeniero o Arquitecto, y el tercero al de Doctor.
- ***Modulares***, constituidos por una serie de materias, cada una de las cuales tendrá un valor medido en créditos. Los créditos tienen una correspondencia en horas de docencia, que en la actualidad es de 10 horas por crédito. Para la obtención del título correspondiente a un ciclo, el alumno deberá lograr una determinada cantidad de créditos previamente establecida.
- ***Flexibles***, para ofrecer al alumno la posibilidad de decidir y elaborar su perfil académico, seleccionando entre una serie de materias optativas las que considere más convenientes.

Para hacer posible esta flexibilidad y al tiempo asegurar una base común en todos los titulados, se establecen tres tipos de materias:

- ***Materias Troncales***: Constituyen los contenidos mínimos exigibles a un mismo título oficial, y son por tanto obligatorias en todo el territorio nacional. Deben constituir no menos del 30% de la carga docente total durante el primer ciclo, y al menos el 25% de la misma durante el segundo ciclo. Las Universidades, al establecer los correspondientes Planes de Estudios, podrán organizar las materias troncales en disciplinas o asignaturas concretas.
- ***Materias determinadas por la Universidad en sus Planes de Estudios***: Son materias definidas particularmente por cada Universidad para cada titulación; se dividen a su vez en obligatorias y optativas.
  - ***Materias obligatorias:*** Libremente establecidas por cada Universidad, que las incluye dentro del correspondiente Plan de Estudios como obligatorias para el alumno.
  - ***Materias optativas:*** Libremente establecidas por cada Universidad, que las incluye en el correspondiente Plan de Estudios para que el alumno escoja entre las mismas.
- ***Libre configuración***: Al menos el 10% de la carga lectiva de un Plan de Estudios deberá quedar abierta para que el estudiante pueda cursar aquellas materias que libremente escoja entre las ofrecidas por la Universidad. La

finalidad de esta categoría es potenciar la formación interdisciplinaria, y se orienta principalmente a materias de carácter general.

El Consejo de Universidades evaluó el cumplimiento por las Universidades de las directrices generales de los Planes de Estudios, y el contenido científico, técnico o artístico, y la adecuación profesional de los mismos. Esta evaluación ha puesto de manifiesto algunos problemas interpretativos de la normativa establecida por el Real Decreto de 1987. Entre los problemas interpretativos y desajustes se pueden citar los derivados:

- De la distinta duración de los segundos ciclos de estudios que llevan a la misma titulación, que puede conducir a la distorsión del currículo académico del alumno.
- Del incremento excesivo de la troncalidad que puede falsear el contenido homogéneo de las enseñanzas.
- De la tendencia a una especialización excesivamente temprana.
- De la falta de inclusión en los Planes de Estudios, de materias obligatorias u optativas de carácter complementario o instrumental no específicas de la titulación, pero coherente con la formación básica y general que se exige para el primer ciclo.

Como consecuencia de todo esto se estimó necesario efectuar ciertas aclaraciones e introducir modificaciones parciales en algunos de los artículos del Real Decreto 1497/1987 de 27 de noviembre, por medio de la publicación del Real Decreto 1267/1994 de 10 de junio.

Además, estas directrices comunes fueron modificadas por el Real Decreto 779/1998 de 30 de abril de 1998 (B.O.E. número 104 de 1 de mayo de 1998 [BOE, 1998b]). En él se modifica el concepto de crédito (puede dedicarse hasta un 30% a actividades académicas dirigidas) y se establece en 6 el límite máximo de materias a cursar por los alumnos de forma simultánea.

Una vez establecidas las directrices generales comunes, aplicables a todos los Planes de Estudio, se establecen las directrices generales propias de Informática. Esto tiene lugar en los Reales Decretos 1559/1990, 1560/1990 y 1561/1990 (B.O.E. número 278 de 20 de noviembre [BOE, 1990b; BOE, 1990c; BOE 1990d]) en los que se contemplan los títulos de Ingeniero en Informática, Ingeniero Técnico en Informática de Gestión e Ingeniero Técnico en Informática de Sistemas (Tablas 2.12, 2.13 y 2.14 respectivamente). En la actualidad en la Universidad de Salamanca no se imparte el título de Ingeniero Técnico en Informática de Gestión, pero se menciona por considerar que es interesante conocerlo para saber la preparación que tienen los alumnos que acceden al segundo ciclo con esta titulación.

Materias troncales	Créditos	Áreas de Conocimiento
PRIMER CICLO		
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados	6	Ciencia de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de Datos y de la Información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: ficheros, bases de datos	12	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Estructura y Tecnología de Computadores. Unidades funcionales: memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos	15	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica
Fundamentos Físicos de la Informática. Electromagnetismo. Estado Sólido. Circuitos	6	Electrónica, Electromagnetismo, Física Aplicada, Física de la Materia Condensada, Ingeniería Eléctrica y Tecnología Electrónica
Fundamentos Matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos	18	Álgebra, Análisis Matemático, Ciencia de la Computación e Inteligencia Artificial, Matemática Aplicada
Metodología y Tecnología de la Programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: descomposición modular y documentación. Técnicas de verificación y pruebas de programas	15	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Sistemas Operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros	6	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Teoría de Autómatas y Lenguajes Formales. Máquinas secuenciales y autómatas finitos. Máquinas de Turing. Funciones recursivas. Gramáticas y lenguajes formales. Redes neuronales	9	Álgebra, Ciencia de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Lenguajes y Sistemas Informáticos y Matemática Aplicada
SEGUNDO CICLO		
Arquitectura e Ingeniería de Computadores. Arquitecturas paralelas. Arquitecturas orientadas a aplicaciones y lenguajes	9	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica
Ingeniería de Software. Análisis y definición de requisitos. Diseño, propiedades y mantenimiento del software. Gestión de configuraciones. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones	18	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Inteligencia Artificial e Ingeniería del Conocimiento. Heurística. Sistemas basados en el conocimiento. Aprendizaje. Percepción	9	Ciencia de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática y Lenguajes y Sistemas Informáticos
Procesadores de lenguaje. Compiladores. Traductores e intérpretes. Fases de compilación. Optimización de código. Macroprocesadores.	9	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Redes. Arquitectura de redes. Comunicaciones	9	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Ingeniería Telemática y Lenguajes y Sistemas Informáticos
Sistemas Informáticos. Metodología de análisis. Configuración, diseño, gestión y evaluación de sistemas informáticos. Tecnologías avanzadas de sistemas de información, bases de datos y sistemas operativos. Proyectos de sistemas informáticos	15	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa, Ingeniería de Sistemas y Automática, Ingeniería Telemática, Lenguajes y Sistemas Informáticos y Organización de Empresas

Tabla 2.12. Título de Ingeniero en Informática

Materias troncales	Créditos	Áreas de Conocimiento
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados	9	Ciencia de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de Datos y de la Información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: ficheros, bases de datos	12	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Estructura y Tecnología de Computadores. Unidades funcionales: Memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos	9	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica
Fundamentos Matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos	18	Álgebra, Análisis Matemático, Ciencia de la Computación e Inteligencia Artificial, Matemática Aplicada
Ingeniería de Software de Gestión. Diseño, propiedades y mantenimiento del software de gestión. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones de gestión	12	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Metodología y Tecnología de la Programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: descomposición modular y documentación. Técnicas de verificación y pruebas de programas	15	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Sistemas Operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros	6	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Técnicas de Organización y Gestión empresarial. El sistema económico y la empresa. Técnicas de administración y técnicas contables	12	Economía Financiera y Contabilidad y Organización de Empresa

Tabla 2.13. Título de Ingeniero Técnico en Informática de Gestión

Materias troncales	Créditos	Áreas de Conocimiento
Estadística. Estadística descriptiva. Probabilidades. Métodos estadísticos aplicados	9	Ciencia de la Computación e Inteligencia Artificial, Estadística e Investigación Operativa y Matemática Aplicada
Estructura de Datos y de la Información. Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Estructura de información: ficheros, bases de datos	12	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Estructura y Tecnología de Computadores. Unidades funcionales: Memoria, procesador, periferia, lenguajes máquina y ensamblador, esquema de funcionamiento. Electrónica. Sistemas digitales. Periféricos	15	Arquitectura y Tecnología de Computadores, Electrónica, Ingeniería de Sistemas y Automática y Tecnología Electrónica
Fundamentos Físicos de la Informática. Electromagnetismo. Estado Sólido. Circuitos	6	Electrónica, Electromagnetismo, Física Aplicada, Física de la Materia Condensada, Ingeniería Eléctrica y Tecnología Electrónica
Fundamentos Matemáticos de la Informática. Álgebra. Análisis matemático. Matemática discreta. Métodos numéricos	18	Álgebra, Análisis Matemático, Ciencia de la Computación e Inteligencia Artificial y Matemática Aplicada
Metodología y Tecnología de la Programación. Diseño de algoritmos. Análisis de algoritmos. Lenguajes de Programación. Diseño de programas: descomposición modular y documentación. Técnicas de verificación y pruebas de programas	12	Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos
Redes. Arquitectura de redes. Comunicaciones	9	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial, Ingeniería de Sistemas y Automática, Ingeniería Telemática y Lenguajes y Sistemas Informáticos.
Sistemas Operativos. Organización, estructura y servicio de los sistemas operativos. Gestión y administración de memoria y de procesos. Gestión de entrada/salida. Sistemas de ficheros	6	Arquitectura y Tecnología de Computadores, Ciencia de la Computación e Inteligencia Artificial y Lenguajes y Sistemas Informáticos

Tabla 2.14. Título de Ingeniero Técnico en Informática de Sistemas

2.7.3 Las titulaciones de Informática en la Universidad de Salamanca

2.7.3.1 Estudios de primer ciclo

Como se ha comentado anteriormente, la enseñanza oficial de la Informática en España comenzó en el año 1969, con la creación del Instituto de Informática en Madrid. En 1976 es cuando se crean en nuestro país las primeras Facultades de Informática, en Madrid, Barcelona y San Sebastián. Limitándonos al entorno geográfico de Castilla y León, fue la Escuela Universitaria de Ingeniería Técnica Industrial de Valladolid la que primeramente estableció la Sección de Informática, y comenzó a impartir docencia en el año académico 1985-1986.

La Universidad de Salamanca inició este tipo de estudios en el curso 1989-1990, bajo la responsabilidad de la Facultad de Ciencias. La primera titulación impartida en este campo fue la de **Diplomatura en Informática**, tal y como se recoge en el Real Decreto 1025/1989 publicado

en el B.O.E. de 10 de agosto de 1989 [BOE, 1989]. Se trataba de estudios de primer ciclo, con una duración de tres años académicos. La carga lectiva total era de 227 créditos, de los cuales 190 correspondían a materias obligatorias, 12 a asignaturas optativas y el resto a asignaturas de libre elección [BOE, 1990a], tal y como se indica en la Tabla 2.15.

Poco más tarde, en concreto en octubre del año siguiente, se aprueba el Real Decreto 1561/1990, publicado en el B.O.E. de 20 de Noviembre de 1990 [BOE, 1990d], en el que se establece el título oficial de **Ingeniero Técnico en Informática de Sistemas**. En este decreto se recogen las directrices generales de los Planes de Estudio que deben cursarse para su obtención y homologación según propuesta del Consejo de Universidades y del Ministerio de Educación y Ciencia.

Siguiendo dichas directrices, la Universidad de Salamanca procede a la implantación de este Plan de Estudios, de acuerdo con la resolución de 23 de Noviembre de 1992, publicada en el B.O.E. de 7 de Enero de 1993 [BOE, 1993]. La nueva titulación de Ingeniería Técnica en Informática de Sistemas comienza su andadura en el curso 1993-1994 y está vigente en la actualidad.

El comienzo de estos nuevos estudios supuso, entre otras cosas, una redistribución de la carga lectiva, que pasó a ser de 213 créditos, repartidos tal y como aparece reflejado en la Tabla 2.15. Otra de sus características más destacadas fue la aparición de la asignatura Proyecto como materia obligatoria de 12 créditos en tercer curso, tal y como puede verse en la Tabla 2.16. En esta tabla se muestran además todas las asignaturas troncales (T) y obligatorias (Obl), indicando el curso en el que se imparten y el número y distribución de los créditos asociados a cada una de ellas. Las materias optativas (Opt) y de libre configuración (LC) no se detallan, únicamente se indica el número total de créditos de cada tipo necesarios en cada curso.

	Troncal	**Obligat.**	**Optativa**	**Libre C.**	**TOTAL**
Diplomatura	**190**	**-**	**12**	**25**	**227**
Ingeniería Técnica (Plan 1992)	**159**	**12**	**18**	**24**	**213**
Primer curso	60	-	-	8	68
Segundo curso	57	-	9	8	74
Tercer curso	42	12	9	8	71
Ingeniería Técnica (Plan 1997)	**100.5**	**49.5**	**30**	**21**	**201**
Primer curso	42	24	-		66
Segundo curso	46.5	10.5	12	6	75
Tercer curso	12	15	18	15	60

Tabla 2.15. Comparación de la carga lectiva de la titulación de Informática en los diferentes Planes de Estudios

	Asignatura	Tipo	Créditos		
			Totales	Teóricos	Prácticos
Primer Curso	Álgebra	T	12	6	6
	Análisis Matemático	T	12	6	6
	Electrónica	T	9	6	3
	Fundamentos Físicos de la Informática	T	9	6	3
	Introducción a los Computadores	T	9	6	3
	Programación I	T	9	6	3
		L.C.	8		
Segundo Curso	Estructura y Tecnología de Computadores	T	12	6	6
	Sistemas Operativos	T	12	9	3
	Bases de Datos	T	9	6	3
	Estadística	T	9	6	3
	Programación II	T	9	6	3
	Estructura de la Información	T	6	4.5	1.5
		Opt	9		
		L.C.	8		
Tercer Curso	Ampliación de Matemáticas	T	9	6	3
	Redes de Ordenadores	T	9	6	3
	Teoría de Autómatas y Lenguajes Formales	T	9	6	3
	Teoría de la Señal y Transmisión de Datos	T	9	6	3
	Ingeniería del Software	T	6	4.5	1.5
	Proyecto	Obl	12	0	12
		Opt	9		
		L.C.	8		

Tabla 2.16. Asignaturas del Plan de Estudios de 1992

En el curso 1997 la Universidad de Salamanca se volvió a replantear la estructura y contenidos de varias titulaciones, entre ellas la Ingeniería Técnica en Informática de Sistemas. Dicha revisión se realizó atendiendo a:

- Las recomendaciones del Consejo de Universidades (B.O.E. de 17 de Enero de 1997 [BOE, 1997a]).
- Las normas que han modificado el Real Decreto 1497/87 hasta el momento presente.

- La evaluación de la Comisión a la que la Junta de la Facultad de Ciencias encargó el análisis del Plan de 1992 una vez implantados los tres años académicos de que consta; esta evaluación ha tenido en cuenta las opiniones del profesorado y de los alumnos, principalmente sobre el rendimiento académico.

	Asignatura	Tipo	Créditos		
			Totales	Teóricos	Prácticos
PRIMER CURSO	Fundamentos Físicos de la Informática	T	7.5	6	1.5
	Álgebra	T	6	3	3
	Cálculo Diferencial	T	6	3	3
	Programación	T	6	6	0
	Sistemas Informáticos	Obl	6	3	3
	Laboratorio de Programación	Obl	4.5	0	4.5
	Electrónica	T	9	5	4
	Algoritmia	T	7.5	4.5	3
	Cálculo Integral	Obl	7.5	4.5	3
	Álgebra Computacional	Obl	6	3	3
SEGUNDO CURSO	Estadística	T	7.5	4.5	3
	Unidades Funcionales del Ordenador	T	7.5	5	2.5
	Matemática Discreta	T	6	3	3
	Sistemas Operativos	T	6	6	0
	Estructuras de Datos	T	6	3	3
	Diseño de Bases de Datos	T	4.5	4.5	0
	Lenguajes Formales	T	4.5	3	1.5
	Sistemas de Bases de Datos	T	4.5	1.5	3
	Transmisión de Datos	Obl	6	4.5	1.5
	Laboratorio de Sistemas Operativos	Obl	4.5	0	4.5
		Opt	12		
		L.C.	6		
TERCER CURSO	Redes	T	7.5	4.5	3
	Informática Teórica	T	4.5	3	1.5
	Ingeniería del Software	***Obl***	***6***	***4.5***	***1.5***
	Proyecto	Obl	9	0	9
		Opt	18		
		L.C.	15		

Tabla 2.17. Asignaturas del Plan de Estudios de 1997

Con ello se elabora un nuevo Plan, que fue aprobado por resolución de 15 de octubre de 1997; publicándose en el B.O.E. de 4 de Noviembre de 1997 [BOE, 1997b] y en [GAFC-USAL, 1997]. Las modificaciones más destacadas fueron: *una estructuración básicamente cuatrimestral* y *una nueva disminución y reorganización de la carga lectiva*. La distribución global de créditos de este nuevo Plan aparece también en la Tabla 2.15. En la Tabla 2.17 se muestran con detalle las materias troncales y obligatorias de que consta y su distribución concreta.

Con el Plan de 1997 se amplía la flexibilidad de los estudios, reduciendo la proporción de materias troncales en favor de las obligatorias y optativas. Ello permite una mayor libertad, tanto a la Universidad, a la hora de definir sus programas, como al alumno para elegir su especialización. Destaca, además, que la optatividad aumenta con el curso, para tener en cuenta que el estudiante, a medida que avanza en los estudios, tiene un mayor conocimiento y, por tanto, más capacidad de elección. La Tabla 2.18 muestra la relación concreta de las asignaturas optativas de este Plan de Estudios [GAFC-USAL, 2001] y [USAL, 2002].

Asignatura	Créditos		
	Totales	Teóricos	Prácticos
Arquitecturas Avanzadas	6	4.5	1.5
Programación Orientada a Objetos	6	3	3
Interfaces Gráficas	6	3	3
Administración de Sistemas Informáticos	6	3	3
Control de Procesos	6	4.5	1.5
Tecnología de Control	6	3	3
Modelado y Simulación	6	4.5	1.5
Introducción a la Economía de Empresa	6	4.5	1.5
Lógica Matemática	6	4.5	1.5
Modelos Estadísticos Lineales	6	4.5	1.5
Periféricos	6	3	3
Sistemas de Transmisión de Señal	6	4.5	1.5
Paquetes Estadísticos	6	1.5	4.5

Tabla 2.18. Asignaturas optativas del Plan de Estudios de 1997

Otra de las características que se ha buscado en este Plan es disminuir la carga lectiva del sexto cuatrimestre para que los alumnos puedan dedicarse más intensamente a la elaboración del proyecto de final de carrera.

La implantación de este nuevo Plan de la Ingeniería Técnica en Informática de Sistemas comenzó en el curso académico 1997-1998.

2.7.3.2 Estudios de segundo ciclo

La titulación de Ingeniería en Informática comienza a impartirse en la Universidad de Salamanca en el curso académico 1998-1999, siendo el Centro Universitario responsable de la organización del Plan de Estudios la Facultad de Ciencias. Para acceder a estos estudios de segundo ciclo es necesario estar en posesión del título de Ingeniero Técnico en Informática de Gestión o de Sistemas, de acuerdo con la Orden de 11 de septiembre de 1991 (B.O.E. de 26 de septiembre [BOE, 1991]).

El Plan de Estudios aprobado por la Universidad de Salamanca fue homologado por acuerdo de 27 de octubre de 1998 [BOE, 1998b], de la Comisión Académica del Consejo de Universidades. La resolución de la Universidad de 10 de junio de 1999 aparece publicada en el B.O.E. número 156 de 1 de julio de 1999 [BOE, 1999] y en [GAFC-USAL, 1998]. La elaboración del Plan de Estudios de dicha titulación se basó en las directrices generales de los Planes de Estudio (Real Decreto 1497/1987 de 27 de noviembre [BOE, 1987]) y en las directrices para la obtención del título oficial de Ingeniero en Informática (Real Decreto 1559/1990, B.O.E. de 20 de noviembre [BOE, 1990b]) que se recogen en la Tabla 2.12.

La carga lectiva total es de 127 créditos distribuidos en dos cursos. El reparto de los créditos aparece en la Tabla 2.19. Cabe resaltar que para la obtención del título de Ingeniero Informático es necesario realizar un proyecto de fin de carrera que está incluido dentro de la troncalidad con una carga de seis créditos.

Curso	Materias troncales	Materias obligatorias	Materias optativas	Créditos de libre configuración	Totales
1º	*36*	*9*	*12*	*6*	*63*
2º	*33*	*6*	*18*	*7*	*64*
Total 2º ciclo	**69**	**15**	**30**	**13**	**127**

Tabla 2.19. Distribución de la carga lectiva global del segundo ciclo

	Asignatura	Tipo	Créditos		
			Totales	Teóricos	Prácticos
PRIMER CURSO	Arquitectura e Ingeniería de Computadores	T	9	6	3
	Análisis de Sistemas	***T***	***9***	***6***	***3***
	Procesadores de Lenguaje	T	9	6	3
	Redes	T	9	6	3
	Ampliación de Sistemas Operativos	Obl	9	4,5	4,5
SEGUNDO CURSO	***Administración de Proyectos Informáticos***	***T***	***9***	***6***	***3***
	Inteligencia Artificial e Ingeniería del Conocimiento	T	9	6	3
	Sistemas de Información	T	9	0	9
	Proyecto	T	6	0	6
	Ampliación de Bases de Datos	Obl	6	4.5	1,5

Tabla 2.20. Asignaturas de la titulación de Ingeniero en Informática (Plan de 1999).

Asignatura	Créditos		
	Totales	Teóricos	Prácticos
Programación Paralela y Distribuida	6	3	3
Administración de Sistemas Informáticos	6	2	4
Procesamiento de Imágenes	6	3	3
Informática Gráfica	6	3	3
Microelectrónica	6	4.5	1.5
Diseño de Circuitos Digitales	6	3	3
Técnicas de Investigación Operativa	6	3	3
Reconocimiento de Patrones	6	3	3
Técnicas de Control de Calidad	6	3	3
Criptografía	6	3	3
Lógica Computacional	6	3	3
Cálculo Numérico	6	3	3
Tecnología de Control	6	4.5	1.5
Robótica	6	4.5	1.5
Lógica para la Informática y la Inteligencia Artificial	6	1.5	4.5

Tabla 2.21. Asignaturas optativas de la titulación de Ingeniero en Informática

Las asignaturas troncales y obligatorias que se imparten en los dos cursos en que se divide el segundo ciclo se muestran en la Tabla 2.20, mientras que la oferta de asignaturas optativas figura en la Tabla 2.21 [GAFC-USAL, 2001; USAL, 2002]. Observando la primera tabla se pueden destacar una serie de aspectos particulares de esta titulación:

- La materia troncal Ingeniería del Software se ha dividido en dos asignaturas, Análisis de Sistemas, que se imparte el primer curso, y Administración de Proyectos Informáticos que aparece en segundo curso.
- La Universidad de Salamanca, además de las asignaturas correspondientes a las materias troncales, ha incluido dos asignaturas obligatorias, Ampliación de Sistemas Operativos y Ampliación de Bases de Datos, para completar la formación de los estudiantes de esta titulación.
- En segundo curso aparecen dos asignaturas troncales cuya carga crediticia es totalmente práctica, la asignatura Sistemas de Información, que se cursa en el primer semestre, y la asignatura Proyecto, que se cursa en el segundo. Para examinarse de esta última será necesario tener aprobados el resto de los créditos, cualquiera que sea la naturaleza de éstos.

2.8 Referencias

[ACM/IEEE-CS, 2001] The Joint Task Force on Computing Curricula: IEEE Computer Society and Association for Computing Machinery. "*Computing Curricula 2001 – Computer Science*". Final Report, http://www.computer.org/education/cc2001/final/cc2001.pdf. [Última vez visitado, 27-12-2001]. December 15, 2001.

[Bagert, 1999] Bagert, D. J. "*Talking the Lead in Licensing Software Engineers*". Communications of the ACM, 42(4): 27-29. April 1999.

[BOCG, 2001] Boletín Oficial Cortes Generales. "*Ley Orgánica de Universidades*". Congreso de los Diputados. Serie A. Núm. 45-13. Páginas 463-495. 26 de diciembre de 2001.

[BOE, 1987] *Boletín Oficial del Estado de 14 de Diciembre de 1987*; R. D. 1497/1987 de 27 de noviembre, por el que se establecen las directrices generales comunes de los Planes de Estudios.

[BOE, 1988] *Boletín Oficial del Estado de 5 de Julio de 1988*; R. D. 678/1988 de 1 de Julio, por el que se aprueban los estatutos de la Universidad de Salamanca.

[BOE, 1989] *Boletín Oficial del Estado de 10 de Agosto de 1989*; R. D. 1025/1989 de 28 de Julio, por el que se autoriza la implantación de los estudios de Diplomado en Informática en la Universidad de Salamanca.

[BOE, 1990a] *Boletín Oficial del Estado de 18 de Julio de 1990*; Resolución de 17 de Mayo de 1990, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de la Diplomatura de Informática.

[BOE, 1990b] *Boletín Oficial del Estado de 20 de Noviembre de 1990*; R. D. 1559/1990, por el que se establece el título oficial de Ingeniero en Informática.

[BOE, 1990c] *Boletín Oficial del Estado de 20 de Noviembre de 1990*; R. D. 1560/1990, por el que se establece el título oficial de Ingeniero Técnico en Informática de Gestión.

[BOE, 1990d] *Boletín Oficial del Estado de 20 de Noviembre de 1990*; R. D. 1561/1990, por el que se establece el título oficial de Ingeniero Técnico en Informática de Sistemas.

[BOE, 1991] *Boletín Oficial del Estado de 26 de Septiembre de 1991*; Resolución de 11 de septiembre de 1991, por la que se aprueba las normas de acceso a segundo ciclo de Ingeniero en Informática en la Universidad de Salamanca.

[BOE, 1993] *Boletín Oficial del Estado de 7 de Enero de 1993*; Resolución de 23 de Noviembre de 1992, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero Técnico en Informática de Sistemas.

[BOE, 1997a] *Boletín Oficial del Estado de 17 de Enero de 1997*; Recomendaciones del Consejo de Universidades para la Ingeniería Técnica en Informática de Sistemas.

[BOE, 1997b] *Boletín Oficial del Estado de 4 de Noviembre de 1997*; Resolución de 15 de Octubre, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero Técnico en Informática de Sistemas.

[BOE, 1998a] *Boletín Oficial del Estado de 27 de Octubre de 1998*; Homologación por la Comisión Académica del Consejo de Universidades del Plan de Estudios de Segundo Ciclo de Ingeniero en Informática de la Universidad de Salamanca.

[BOE, 1998b] *Boletín Oficial del Estado de 20 de Noviembre de 1998*; R. D. 779/1998, por el que se establecen las directrices generales comunes de los Planes de Estudio de los títulos universitarios de carácter oficial y validez en todo el territorio nacional.

[BOE, 1999] *Boletín Oficial del Estado de 1 de Julio de 1999*; Resolución de 10 de junio de 1999, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero en Informática (2º ciclo).

[BOE, 2001] *Boletín Oficial del Estado de 24 de Diciembre de 2001*, Ley Orgánica de Universidades. LO 6/2001, de 21 de diciembre.

[Camps, 1999] Camps Paré, R. "*¿Qué Informática Se Enseña en la Universidad? (Primera Parte)*". Novática. Nº 141: 48-51. Septiembre/Octubre, 1999.

[CEHU, 2002] Centro de Historia Universitaria Alfonso IX (CEHU). "*Historia de las Universidades*". Universidad de Salamanca. http://www3.usal.es/alfonsoix/. 2002.

[Denning, 1998] Denning, P. J. "*Computer Science and Software Engineering: Filing for Divorce?*". Communications of the ACM, 40(8):128. August 1998.

[DPTOIA, 2001] Departamento de Informática y Automática. "*Memoria de Actividades*". Universidad de Salamanca. 2001.

[DPTOIA, 2002] Departamento de Informática y Automática. "*Web del Departamento de Informática y Automática de la Universidad de Salamanca*". http://dptoia.usal.es, 2002.

[El-Kadi, 1999] El-Kadi, A. "*Stop that Divorce*". Communications of the ACM, 42(12):27-28. December 1999.

[GAFC-USAL, 1997] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1997-1998*". Ediciones Universidad de Salamanca, 1997.

[GAFC-USAL, 1998] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1998-1999*". Ediciones Universidad de Salamanca, 1998.

[GAFC-USAL, 2001] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 2001/2002*". Ediciones Universidad de Salamanca, 2001.

[GAU-USAL, 2001] Universidad de Salamanca. "*Guía Académica de la Universidad de Salamanca 2001/2002*". Ediciones Universidad de Salamanca, 2001.

[Lutz y Naveda, 1997] Lutz, M. J., Naveda, J. F. "*The Road Less Traveled: A Baccalaureate Degree in Software Engineering*". In Proceedings of the twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97). (February 27 - March 1, 1997, San Jose, CA USA). ACM. Pages 287-291. 1997.

[MEC, 1983] Ministerio de Educación y Ciencia. "*Ley Orgánica de Reforma Universitaria*". Servicio de Publicaciones del MEC, 1993.

[Ortega, 1982] Ortega y Gasset, J. "*Misión de la Universidad*". En P. Garagorri *Obras de J. Ortega y Gasset*, edición nº 22. El Arquero, Alianza Editorial, 1982.

[Parnas, 1997] Parnas, D. L. "*Software Engineering: An Unconsummated Marriage*". Communications of the ACM, 40(9):128. September 1997.

[Parnas, 1999] Parnas, D. L. "*Software Engineering Programs Are Not Computer Science Programs*". IEEE Software, 16(6):19-30. November/December 1999.

[Rodríquez-San Pedro, 1991] Rodríquez-San Pedro, L. E. "*La Universidad de Salamanca, Evolución y Declive de un Modelo Clásico*". Studia Histórica. Historia Moderna IX, pp. 9-21. 1991.

[Rodríquez-San Pedro, 1996] Rodríquez-San Pedro, L. E. "*Universidades en Castilla y León*". En: *Historia de una cultura: Castilla y León*. Valladolid: Junta de Castilla y León, Conserjería de Educación y Cultura. Tomo IV, pp. 403-423. 1996.

[Russell, 1987] Russell, B. "*La Perspectiva Científica*". 2ª edición. Editorial Ariel, 1987.

[Russell, 1997] Russell, B. "*Respuestas*". Edición de Lee Eisler. Península, 1997.

[Savater, 1998] Savater, F. "*El Valor de Educar*". 9ª edición. Editorial Ariel, S.A., 1998.

[Sierra, 1986] Sierra Bravo, R. "*Tesis Doctorales y Trabajos de Investigación Científica. Metodología General de su Elaboración y su Documentación*". Paraninfo, 1986.

[Tucker et al., 1990] Tucker, A. B. (Editor and Co-chair), Barnes, B. H. (Co-chair), Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*Computing Curricula 1991. Report of the ACM/IEEE-CS Joint Curriculum Task Force*". ACM. http://www.computer.org/education/cc1991/. [Última vez visitado, 27-12-2001]. December 1990.

[USAL, 2000] Universidad de Salamanca. "*Guía de Tercer Ciclo y Doctorado 2000/2001*". Secretaría General de la Universidad de Salamanca, 2000.

[USAL, 2002] Universidad de Salamanca. "*Web de la Universidad de Salamanca*". http://www.usal.es. 2002.

Capí tulo 3
Aspectos Metodológicos

La propuesta de un proyecto docente no se limita a la definición de unos contenidos temáticos y a su distribución temporal. Es necesario acompañar estos contenidos de una metodología que permita la consecución de los objetivos planteados. Los avances de las Tecnologías de la Información proporcionan herramientas cuya utilidad, mediante un uso adecuado, es indiscutible. Estos aspectos son analizados en detalle en este capítulo.

3.1 Introducción

Etimológicamente, la palabra "método" procede de las voces griegas "meta" (a través de) y "odos" (camino). Literalmente, el método es "*el camino para la realización y cumplimiento de un determinado objetivo*". El término método presenta dos acepciones bien diferenciadas: por un lado sería *el modo de hacer o decir una cosa*; y por otro *el procedimiento que siguen las ciencias para hallar la verdad y enseñarla*. De estas acepciones se puede inferir que **un método de enseñanza** o **una metodología docente** es "*el conjunto de normas y procedimientos destinados a dirigir el aprendizaje de forma eficiente*". A través de él, se debe procurar la correcta ordenación de todos los elementos que integran la acción educativa con el fin de mejorar el proceso e incrementar la seguridad y eficacia del mismo en la consecución de los objetivos establecidos.

El método docente aplicable al desarrollo de una asignatura debe partir siempre de la definición de los objetivos que se persigue alcanzar. A continuación hay que establecer los contenidos y, sirviéndose de los medios o recursos instrumentales disponibles, se determinan las

estrategias metodológicas que se van a llevar a cabo. Por último, es preciso establecer los mecanismos de evaluación que permitan determinar en qué medida se han alcanzado los objetivos propuestos y en qué medida el proceso docente seguido ha sido el más apropiado.

Teniendo en cuenta todos estos puntos se pasa a continuación a describir las peculiaridades del método, que sirve como base para el desarrollo de la actividad docente a la que se refiere este Proyecto Docente. En particular, se van a presentar los puntos clave que se han mencionado:

- *Establecimiento de objetivos.*
- *Análisis y selección de contenidos.*
- *Método y técnicas docentes.*
- *Evaluación.*

Se ha de tener en cuenta que la descripción concreta de los contenidos y los medios se describirán posteriormente en los capítulos dedicados a los programas de las asignaturas objeto del perfil de la plaza a concurso.

3.2 Ideas básicas de pedagogía y didáctica

Sin pretender ser exhaustivos, el objetivo de este apartado es exponer unas ideas generales sobre la actividad educativa y docente que pasan a detallarse a continuación.

3.2.1 Pedagogía

Sin duda, se está asistiendo en estos momentos a una fase en la que los aspectos o criterios pedagógicos alcanzan cada vez más importancia. Es evidente que un profesor de Informática no tiene que ser un experto en temas pedagógicos, pero como profesional de la docencia sí debe preocuparle, ya que los objetivos educativos, que le corresponden, dependerán en gran medida, de cómo se enseña. "*Descuidar la atención a los métodos con la intención de dedicarse a los contenidos es falso camino; porque los métodos - sin perder su función instrumental - pueden impedir, si no son adecuados, la transmisión de cualquier contenido*" [Gómez, 1981].

La pedagogía moderna ha originado un giro importante en los planteamientos tradicionales de la enseñanza en lo que se refiere a contenidos. En este sentido, se descarta la materia o disciplina de forma aislada y se tiende al desarrollo integral del individuo, buscando su inserción en el contexto social en el que ha de desenvolverse. El objetivo fundamental de este

enfoque es la formación integral del alumno, para lo que deben tenerse en cuenta: *las necesidades de la sociedad y sus recursos, los grupos profesionales, el progreso científico, las aptitudes de los estudiantes, el sistema cultural y social...*

Los métodos tradicionales se basan casi exclusivamente en la enseñanza, dando por sentado que la información impartida al estudiante es siempre aprendida. Estos métodos se preocupan de la forma cómo se transmite la información, sin analizar a fondo qué se aprende, por quién, con qué rapidez, y sobretodo con qué fines, colocando al estudiante en una posición pasiva.

El método de basarse en la enseñanza es el único concebido hasta la aparición de las teorías de Rousseau. En palabras de Rousseau en "*Emilio o la Educación*" (libro III): "*Me basta con que sepa encontrar el para qué de todo lo que hace y el por qué de todo lo que cree. Pues una vez más mi objetivo no es darle la Ciencia sino enseñarle a adquirirla cuando la necesite, hacerle estimar exactamente lo que vale y hacerle amar la verdad por encima de todo*". Hasta entonces, los objetos importantes eran el saber y el maestro. La innovación de Rousseau y sus sucesores fue simplemente trasladar el fundamento de la ciencia pedagógica desde el *saber y el maestro* al *discípulo,* y reconocer que es el discípulo y sus condiciones peculiares lo único que puede servir de guía para construir una metodología docente.

El aprendizaje es un proceso dinámico de interacción en el que es primordial el estudiante. Éste no sólo recibe, sino que aporta su contribución. Su percepción de la información es tan importante como la emisión de la misma por parte del docente, además su participación y opinión sobre un programa de formación pueden ser más válidas que las de los propios docentes. El punto de partida básico debe de ser, por tanto, *la adquisición de conocimientos* y no su transmisión, haciendo que el aprendizaje se base en la satisfacción personal de alcanzar el grado de competencia requerido para el ejercicio profesional.

Es por ello que la pedagogía moderna concede al aprendizaje activo y a la participación del alumno en su proceso de formación un papel fundamental. Ni que decir tiene que el papel del profesor, ante estos nuevos enfoques, se altera de modo sustancial. Se ha de pasar del profesor que posee el saber, toma decisiones, se hace escuchar y explica conocimientos, al profesor que promueve el saber, crea responsabilidades y capacidades para afrontarlas, enseña a tomar decisiones, incita a la participación y, junto a la exposición de sus conocimientos, aplica técnicas de trabajo y de enseñanza. Desde esta óptica, la función del profesor es garantizar que el alumno realice su propio aprendizaje, lo que lleva a un entendimiento de la enseñanza como un proceso activo bidireccional. Esta concepción conlleva un cambio de actitud y de funciones del docente: de la instrucción - transmisión de conocimientos propiamente dichos - se pasa a la

investigación y a la discusión. Las aulas se convierten entonces en un centro de trabajo que une al profesor que investiga y enseña, y al alumno que investiga y aprende.

La aplicación de un aprendizaje activo a la enseñanza de la Informática, hoy es algo que parece incuestionable. Una metodología activa que fomente el espíritu de participación, el hábito de estudio, la capacidad de enfrentarse a problemas y decisiones diferentes, la búsqueda de fuentes bibliográficas y otros medios apropiados, la actitud de crítica constructiva..., es algo consustancial a este tipo de asignaturas.

3.2.2 Didáctica

La didáctica es *el conjunto de técnicas a través de las cuales se realiza la enseñanza para que ésta resulte más eficaz.* Se diferencia de la pedagogía en que su ámbito es más reducido que el de esta última, puesto que la pedagogía se ocuparía a la par tanto de la enseñanza como de la educación.

Por tanto, el objetivo principal de la didáctica es orientar la enseñanza mediante un conjunto de procedimientos y normas destinadas a dirigir el aprendizaje de forma eficaz. El marco en el que esos procedimientos y normas se materializan en el desarrollo de enseñanza y aprendizaje constituyen el acto didáctico, en el que intervienen:

- *El sujeto que se instruye, que aprende, el discente.*
- *El sujeto que orienta, que ayuda, que enseña, el docente.*
- *La propia naturaleza del objeto de la enseñanza.*

Estos tres elementos señalan, en cierta manera, el método, el modo y el programa de la enseñanza, indicando también las cualidades que el profesor ha de tener, en cuanto a conocimiento de la materia y para el conocimiento psicológico de los alumnos. Estos conocimientos le permitirán, una vez conocida la realidad de su alumnado, adaptar su enseñanza al mismo, teniendo una visión clara de la finalidad que se persigue. Es por lo que a la didáctica se la ha señalado como una conquista personal, sin normas rígidas, en cuanto que cada profesor tomará las que mejor le vayan con su idiosincrasia y también las que estén más cerca de la realidad de sus alumnos.

El acto didáctico da lugar a un diálogo constante *docente/discente*, a través de la comunicación directa de los contenidos o bien, a través de la orientación al alumno para que él los adquiera por medio de otras fuentes. A. del Pozo Pardo [Pozo, 1982] distingue las siguientes características del acto didáctico, sobre la base de las consideraciones anteriores:

- Se trata de una actividad coordinada y conexionada que implica a ambos agentes (discente y docente) y supone una comunicación entre ambos.
- Es una asociación intencional y consciente que conlleva una decisión consistente en un propósito de enseñar por parte del docente y un propósito de aprender por parte del discente.
- Persigue la consecución de dos objetivos: *la enseñanza* y *el aprendizaje*.

Siguiendo al autor citado, A. del Pozo Pardo [Pozo, 1982], esta interrelación de elementos se materializa en el acto didáctico a través de tres actividades, éstas son la enseñanza, el aprendizaje y la planificación:

- **Enseñanza**: El docente se relaciona con el discente y la materia que debe impartir mediante la actividad considerada como la enseñanza, transmitiendo nociones, conocimientos, habilidades... o posibilitando al discente el acceso a ellas: *buscando el máximo desarrollo de sus facultades, hábitos y conductas*. La eficacia del docente depende de dos factores fundamentales: *conocimiento profundo de la materia*, y *arte o técnica de enseñarla*. El primero exige un continuo contacto con las fuentes de erudición, una puesta al día y una profunda reflexión sobre la disciplina. El segundo requiere no sólo el conocimiento de las técnicas pedagógicas, sino su continua puesta en práctica con espíritu crítico sobre la propia tarea.
- **Aprendizaje**: El discente se relaciona con el docente a través de la materia objeto de la enseñanza, dando lugar a la actividad del aprendizaje. Recibidos los contenidos o el estímulo del docente, el discente los asimila, posibilitando su posterior utilización. En este sentido, se hace fundamental considerar el nivel intelectual y humano de los discentes, teniendo en cuenta que se hallan en un período de formación, tanto académica como de su personalidad.
- **Planificación**: El docente se relaciona con la materia que debe impartir a través de la planificación o programación, actividad ésta que le es exclusiva y mediante la que formula los objetivos que pretende conseguir. En esta actividad selecciona, analiza y ordena los contenidos que impartirá, delimita la metodología que se aplicará y por último, elige los recursos y medios más adecuados para lograr la máxima eficiencia en su labor docente. Una depurada selección de materias, prescindiendo de los temas menos importantes, junto a la elaboración de un programa lógico y coherente, que no excluya la profundización en aquellos

aspectos que se consideren necesarios, son requisitos fundamentales para el docente respecto a la materia que va a enseñar.

La planificación conduce a preguntarse cómo transmitir los conocimientos. En palabras de Lafourcade [Lafourcade, 1974], "*uno de los supuestos claves que contribuyen al logro de una enseñanza de calidad, es la preparación de un plan de acción que articule, de modo racional, los diversos componentes de la tarea didáctica que se debe cumplir*". Podría afirmarse que el nivel de los rendimientos que logran los alumnos es un resultado directo del tipo de estrategia que se haya planteado, así como de los modos que se hayan seleccionado para llevarla a la práctica. A modo de síntesis, se puede señalar que cualquier plan de acción ha de recorrer una serie de fases o etapas que permitan responder por este orden a las siguientes preguntas:

- *¿Qué se pretende con la asignatura y con sus contenidos? Es decir, sus objetivos.*
- *¿Con qué medios se cuenta?*
- *¿Qué actividades y experiencias de aprendizaje deben realizarse para alcanzarse esos objetivos?*
- *¿Cómo organizar y secuenciar estas experiencias y actividades para facilitar su asimilación en el alumno?*
- *¿Cómo conocer si se han alcanzado los objetivos propuestos? ¿Cómo evaluar la eficiencia de esas actividades en función de los objetivos?*

De donde, en principio, se podrían establecer tres etapas para llevar a cabo el proceso didáctico: Programación, Ejecución y Control. En la Figura 3.1 se muestra un esquema de estas etapas, junto con los factores que influyen en cada una de ellas, y que se explicarán más adelante.

La primera etapa representa el diseño de la intervención y supone el establecimiento de los objetivos, el análisis y selección de los contenidos, y la elección de los métodos o técnicas aplicables. La segunda fase consistirá en ejecutar lo programado, y la tercera en analizar la eficacia, comportamiento y relevancia de todas las variables, así como de las posibilidades mostradas por las estrategias docentes y del grado de asimilación y efecto provocado en el alumno.

El concepto de programación es uno de los logros de la moderna tecnología educativa y se fundamenta en la convicción de que la educación, entendida como tarea racional y sistemática, exige saber, previamente, de manera precisa y concreta, qué objetivos deben alcanzar los alumnos, puesto que aquéllos serán, en definitiva, los que objetivan los medios para conseguirlos y los criterios que se emplearán en su valoración.

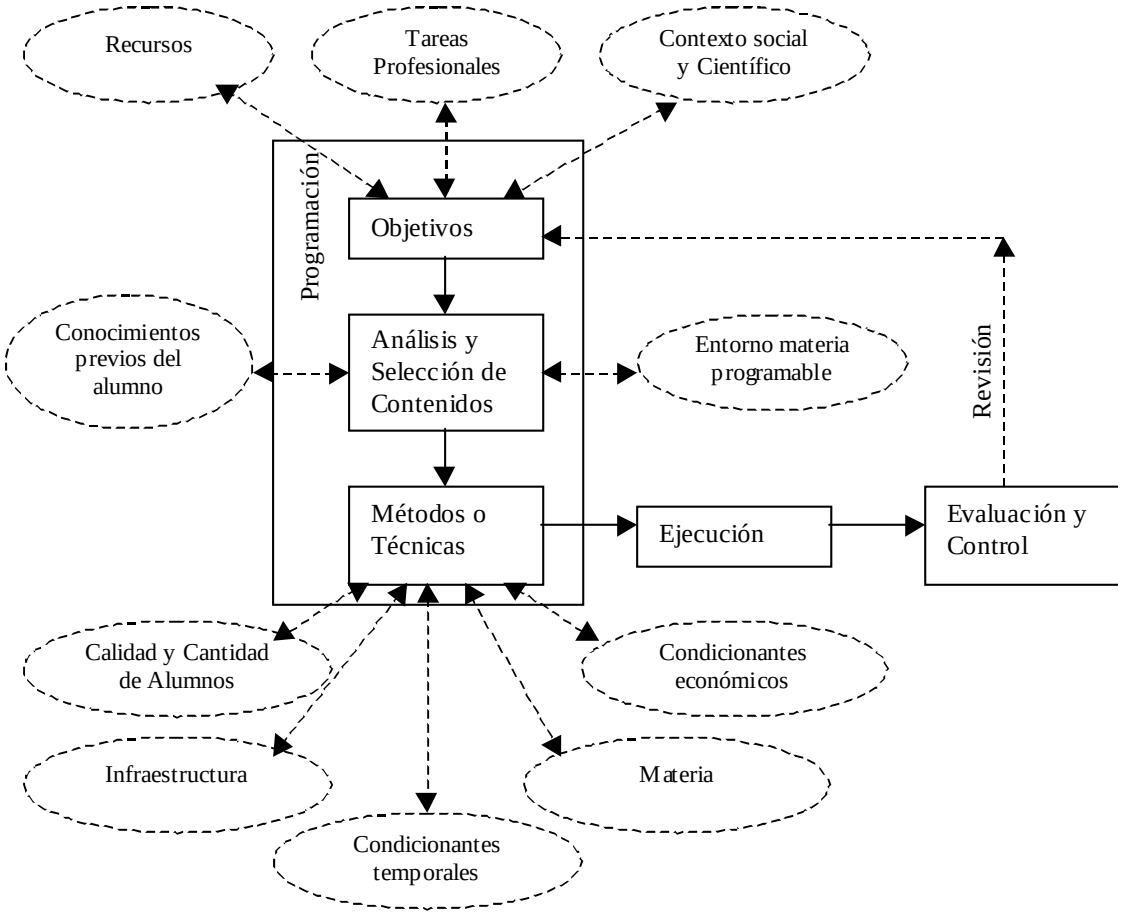


Figura 3.1. Etapas de proceso didáctico

Según lo expuesto, el concepto de programación sería un proceso que coordina objetivos, medios y criterios de medida. O dicho de otra forma, la programación es un proceso abierto, continuo y en permanente transformación, a través del cual se confecciona un programa, siendo este último "*todo intento de enunciar por escrito lo que se va a hacer*" [Pozo, 1982].

Es frecuente utilizar indistintamente los términos programar y planificar, sin embargo, existen matices que los diferencian. Estos matices estarían basados en la mayor o menor concreción y precisión de los aspectos previsibles. La planificación será más general y a más largo plazo; la programación, en cambio, será más específica, más concreta y a corto plazo. De ambas tareas se debe ocupar el profesor, de un lado tiene que elaborar una programación a corto plazo y de otro tiene que planificar las actividades que se realizarán a lo largo de los cursos con el fin de conseguir los objetivos propuestos.

Ya se trate de planificación o de programación, el primer punto a definir en ambos casos será un **conjunto de objetivos**. Los objetivos se pueden definir como *el aprendizaje de nuevas capacidades que una institución se esfuerza por obtener en sus estudiantes.*

Una vez definidos los objetivos educativos se debe planificar **un programa de formación**, es decir, seleccionar una serie de actividades encaminadas a conseguir el logro de los objetivos. Como paso final, se hace necesario definir **los procesos de evaluación**. Éstos deben permitir valorar en qué grado se han alcanzado los objetivos propuestos a través del programa de formación concreto, permitiendo la medida de la competencia final de los educandos, así como la determinación de la eficacia de programas y docentes.

La evaluación del programa va íntimamente ligada a su evolución. Es por ello que la programación nunca es algo totalmente acabado, sino que tiene un carácter dinámico, ha de estar en permanente revisión en función de los resultados obtenidos. Por tanto, la elaboración y modificación del programa ha de tener presente el entorno en el que esté la materia programable, tanto en el ámbito global como particular en lo que se refiere al Centro en el que se imparte y a las relaciones que se pudieran establecer con el resto de las disciplinas de la titulación.

Adicionalmente, los nuevos elementos que llegan al Centro desde el contexto social y científico pueden ocasionar la revisión de los objetivos y en consecuencia del programa. Esto se hace particularmente patente en el ámbito de la docencia en Informática, dada la rápida evolución tecnológica a la que se ve sometida.

A continuación se van a tratar de forma individual las tres etapas para realizar la fase de programación: *objetivos, elaboración del programa de formación* y *los métodos o técnicas para evaluar el proceso formativo*.

3.3 Los objetivos educativos

Los objetivos educativos son definidos por Gullbert [Gullbert, 1989] como: "*Lo que es necesario que los estudiantes sean capaces de llevar a cabo a la terminación de un período de enseñanza/aprendizaje y que no eran capaces de realizar antes*". Por tanto, puede decirse que lo que se persigue, mediante la acción del sistema educativo, es que el alumno adquiera algo que no tenía. Éste sería en definitiva el objetivo básico, que puede enunciarse como el enriquecimiento que se pretende implantar en el alumno.

Ningún sistema de enseñanza puede ser eficaz si no se fijan sus objetivos con claridad. Dentro del proceso educativo los objetivos representan el punto de partida y la referencia para la valoración de su eficacia, ya que con ellos quedan descritos con precisión los resultados a lograr. Es a partir del establecimiento de los mismos cuando el docente se encuentra en

condiciones de seleccionar los contenidos a impartir, los medios a emplear y las estrategias didácticas y metodológicas a utilizar.

Del análisis de los objetivos se pueden extraer tres rasgos o características esenciales, que se desprenden de todas aquellas definiciones de objetivos que se planteen. Según J. Rodríguez Dieguez [Rodríguez, 1980]: "*los objetivos son planteados, (1) poniendo como sujeto al alumno, (2) se enuncian como resultados futuros, y (3) se pueden observar y evaluar*".

El establecimiento de los objetivos con precisión tiene múltiples ventajas, ya que:

- Facilita la evaluación de las metodologías.
- Orienta al alumno en su proceso de aprendizaje.
- Sirve como medio para la autoevaluación del profesorado.
- Incrementa la motivación de los alumnos, al conocer con precisión lo que se les va a pedir.
- Facilita la calificación, ya que favorece la objetividad.
- Contribuye a la elección correcta de los métodos didácticos.
- Posibilita una mayor coordinación entre profesores.

Por otro lado, la formulación de los objetivos no está exenta de pocas dificultades, entre las cuales resaltan:

- El hecho de que todos los objetivos no se pueden explicar. A menudo, se suscitan en el aula cuestiones que no figuran en el programa, por lo que se formularán sólo los objetivos más relevantes, aquéllos que dan sentido de dirección al proceso de enseñanza.
- El problema de la especificación de los objetivos no se circunscribe a una cuestión meramente técnica, sino que implica una visión personal de la educación.

Es por ello que debe de cuidarse la corrección en la formulación de los objetivos. Para que el enunciado de un objetivo sea correcto debe de poseer las siguientes características:

- **Prospectivos**, ya que se forman profesionales para el futuro.
- **Pertinentes**, por su correlación con las tareas profesionales y la realidad social.
- **Precisos y concretos**.

- **Realizables**, contando con los recursos existentes y, principalmente, con la capacidad de aprender del alumno, condicionada tanto por los conocimientos previos que le hayan sido impartidos como por la cantidad de información que éste pueda asimilar.
- **Mensurables**, para que mediante la técnica adecuada de evaluación pueda conocerse en qué grado se han conseguido.

De todas estas características la más esencial es la pertinencia. Cualquier objetivo que reúna todas las demás y no sea pertinente es potencialmente peligroso desde el punto de vista docente.

Los objetivos son de diversa índole, distinguiéndose entre *objetivos de carácter formativo del individuo como tal* y *objetivos específicos*, propios de la naturaleza de la materia a impartir. A continuación se desarrolla esta clasificación con mayor profundidad.

3.3.1 Objetivos formativos de tipo general

Desde un punto de vista general se pueden identificar en la literatura objetivos de formación que se podrían decir que son válidos para cualquier disciplina. Así, Bloom [Bloom, 1956] enuncia una taxonomía que se encarga de enumerar y definir los objetivos de la acción educativa. Dicha taxonomía propone una serie de objetivos, algunos de ellos se refieren a hábitos no meramente intelectuales. Del trabajo de Bloom se observa que los objetivos educativos pueden dividirse en tres grandes bloques o categorías:

- **Objetivos cognitivos**, de desarrollo de la capacidad de recuerdo de datos, de su interpretación y de resolución de problemas;
- **Objetivos psicomotores**, o de adquisición de las aptitudes que requiere la metodología propia de cada disciplina educativa;
- **Objetivos afectivos**, o de motivación del alumno, desarrollando su receptividad y su capacidad de respuesta ante los problemas o situaciones planteadas.

En el caso concreto de la enseñanza universitaria, y más aún de la de la enseñanza técnica, son los objetivos cognitivos los que presentan una importancia mayor. Sin embargo, no conviene olvidar aspectos como la motivación, que en ocasiones pueden resultar fundamentales a la hora de transmitir los conocimientos y de lograr no sólo el aprendizaje, sino la formación integral.

Siguiendo con la referencia al trabajo de Bloom, los objetivos cognitivos se clasifican a su vez en las siguientes categorías:

- **Conocimiento**. El alumno debe aprender una serie de datos específicos, principios, métodos, criterios, clasificaciones..., dentro del campo de estudio. A partir de los conocimientos debe adquirir la capacidad de enunciar, enumerar, describir o definir.
- **Comprensión**. El aprendizaje no puede ser puramente memorístico, el alumno debe comprender las relaciones existentes entre los distintos elementos, diferenciar lo accesorio de lo fundamental y ser capaz de interpretar los conocimientos adquiridos.
- **Aplicación**. Conocer y comprender un determinado método puede no ser suficiente para saber aplicarlo. Efectivamente, la aplicación lleva consigo un grado de abstracción más elevado; al enfrentarse a un determinado problema el alumno debe recurrir a unos ciertos procesos mentales para intentar reducirlo a una situación que pueda ser resuelta mediante la utilización de abstracciones conocidas. Sólo si lo consigue se puede decir que ha adquirido la capacidad de aplicación.
- **Análisis**. El análisis se refiere a la capacidad del alumno para dividir un todo en sus partes fundamentales, obtener sus interrelaciones y su modo de organización.
- **Síntesis**. Es el proceso inverso, que supone, por ejemplo, que el alumno sepa extraer información de diversas fuentes y sea capaz de organizarla de un modo general para elaborar un nuevo material. Este proceso supone el desarrollo de conductas creadoras, es un proceso constructivo y presenta también una gran importancia.
- **Evaluación**. Otra de las conductas básicas que el alumno universitario debe desarrollar es la actitud crítica, es decir, la capacidad de valorar las ideas, los métodos...

La educación es un proceso cuyo fin esencial es modificar la conducta en cuanto a conocimientos, actitudes y aptitudes. La consecuencia de dicha modificación debe ser la aportación al educando de ideas, actitudes, hábitos e intereses que no tenía antes. Por ello, en el ánimo de completar la taxonomía anteriormente presentada, se pueden añadir los siguientes objetivos generales:

- *La capacidad de expresión oral y escrita*. El saber sintetizar las conclusiones de un trabajo. Exponerlas tanto en documentos escritos como en exposiciones audiovisuales ante el público.
- *El respeto por valores de tipo humanístico y unos principios éticos asociados a la profesión*.
- *El interés por la exactitud y rigor*.
- *La capacidad para cooperar con otros profesionales*. Saber trabajar en grupo. Coordinar y planificar tareas.

3.3.2 Objetivos específicos

Ahora bien, preservando estos objetivos de carácter general, no se deben de descuidar aquéllos de carácter específico que estén asociados a la materia que se pretende abordar.

Como se ha señalado anteriormente, se admite que en cualquier materia existen tres taxonomías o campos [Bloom, 1956]: *cognoscitivos, afectivos* y *psicomotores*. Por tanto, el enriquecimiento que deben de proveer unos objetivos educativos en una materia concreta, consiste en los conocimientos que se deben adquirir y también, de las habilidades que se deben desarrollar.

En el ámbito universitario significa proveer al alumno, por una parte, de los conocimientos especializados (finalidad material de la didáctica) y, por otra, de una educación de base más general, que le amplíe las perspectivas sobre el mundo y sus problemas (finalidad formal de la didáctica) como dice Gullbert [Gullbert, 1989]. A través de este último cauce es posible enunciar una serie de objetivos que, con matices, es común a la mayoría de disciplinas, a saber:

- Que los conocimientos expuestos aporten al futuro titulado la terminología y las leyes principales de una materia, así como la familiarización con las principales aplicaciones de la misma. De tal forma que entienda, aprenda y asimile los conceptos básicos, ideas fundamentales y datos específicos, permitiéndole enfrentarse, aplicando los conocimientos adquiridos, con la resolución de casos prácticos similares a los que se le plantearán en su vida profesional.
- Manejar bibliografía, hacer que el alumno disponga de la suficiente información y bibliografía que permita ampliar sus conocimientos en un determinado tema cuando sus necesidades lo requieran, buscarla bien en soporte papel o magnético (ayudas en línea, Internet).

- La capacidad no sólo de comprender la extensión y significado de lo que ya se conoce en el campo donde se encuadra la disciplina, sino de ser receptivo ante lo nuevo, de afrontarlo y de trabajar confiadamente de forma personal. Según Fernando Lara [Lara, 1997] "*Uno de los objetivos imprescindibles debería de ser siempre preparar al alumno para que pueda aprender y para que desee aprender nuevas cosas relativas a la asignatura*".

Estos dos últimos puntos tienen especial importancia en el campo de la Informática. El vertiginoso avance de las Tecnologías de la Información y de la Comunicación obligan a que los profesionales estén sometidos a un proceso de autoformación continua, debiendo discernir entre lo que en la actualidad es válido e innovador, de lo que pueda quedar obsoleto.

Para definir objetivos más específicos de la materia que se esté impartiendo, se deben conocer las necesidades y posibilidades de los alumnos a los que va dirigida la enseñanza, las realidades sociales y los recursos con los que se cuenta para la educación, además del conocimiento claro y preciso de los fines que se deben alcanzar, de modo que a la finalización del proceso, los alumnos estén especialmente preparados para el ejercicio de su profesión.

Consecuentemente, el primer paso que se debe dar en una planificación educativa es definir correctamente las tareas profesionales. Esta definición debe derivar del estudio de las necesidades sociales para este tipo de profesionales, y saber de modo claro y preciso lo que las personas tendrán que hacer en el curso de su ocupación. Si los objetivos se elaboran de este modo, cuando la sociedad evolucione y sus necesidades y tareas profesionales cambien, los objetivos educativos lo harán en consecuencia. Si por el contrario se deja de lado la definición de las tareas profesionales, podría suceder, como indica R. F. Mager [Mager, 1979], que "*si no se está seguro de a dónde se quiere ir, se corre el riesgo de encontrarse en otra parte y no darse cuenta*".

Para la determinación de dichas tareas profesionales, existen distintos métodos de los cuales los más importantes se introducen brevemente a continuación.

- **Técnica del Incidente Crítico**. Basado en estudiar para una situación o incidente determinado, cuáles han sido los aciertos y errores, así como los factores fundamentales que han podido contribuir ha dicho éxito o fracaso. Esto puede dar orientaciones al respecto de la trascendencia de determinados conocimientos, actitudes o aptitudes. En este sentido el incidente crítico puede venir dado muchas veces por la propia experiencia profesional del docente. Por ello, se hace muy interesante que el profesor haya sido en algún momento por un lado discente de la

materia que está impartiendo, así como que por otro lado, haya intentado ponerla en práctica como profesional.

- **Técnica del Análisis de Funciones**. Consiste en estudiar las competencias de la actividad profesional sobre la que se pretende impartir docencia. Esta técnica exige que este análisis venga dado a través de una institución representativa o de prestigio, de lo contrario es fácil caer en conclusiones subjetivas. Existen distintos modelos de currículos propuestos por entidades de renombre que serán presentados en secciones posteriores.

- **Estudio de la Demanda Laboral**. El estudio de la demanda laboral se puede llevar a cabo por un lado a través de un estudio de dicha demanda expresada a través de los medios de comunicación. Un ejemplo de este tipo de información se puede ver en [González, 1996][8]. Este estudio es fácil de obtener y muy útil de cara a identificar los elementos informativos, no así tanto los formativos que pueden requerir el contacto directo con empleadores y responsables de recursos humanos de organizaciones. Dichos responsables deben, a ser posible, constituir una muestra con una heterogeneidad representativa del entorno real de trabajo, lo que permitiría llegar a unas conclusiones válidas.

- **Encuestas Profesionales**. Este tipo de encuesta es difícil de obtener en cuanto debe de haber organismos profesionales encargados de llevarlas a cabo. En el caso que nos compete las únicas encuesta a las que se ha tenido acceso son los trabajos de [McLeod, 1996] y [Sanchís y Torralba, 1997].

La combinación de todos estos métodos permite un mayor acercamiento a la realidad del momento. Así mismo, la previsible variación de algunos de estos datos de entrada hace recomendable la revisión periódica de los objetivos. En todo ello está de forma implícita la aceptación, por parte de las personas encargadas de la docencia, de una mayor responsabilidad que lo que supone la mera preparación de profesionales que se adapten a la situación del instante presente: unos buenos objetivos docentes deben definir también las futuras necesidades y tendencias de la profesión, y preparar personas capaces de ajustarse a ellas.

[8] Aunque este estudio es muy exhaustivo se centra fundamentalmente en la demanda de titulados superiores y medios en Informática con relación a los no titulados, por lo que no será válido para determinar objetivos de asignaturas.

3.4 Análisis y selección de contenidos

Los contenidos son el instrumento que va a permitir alcanzar los objetivos formulados. En este sentido se entiende que el contenido es un medio para algo, y no un fin en sí mismo. La selección de contenidos no es entonces arbitraria, sino que está determinada por una serie de factores, entre los que destacan:

- *Los objetivos generales de la carrera y los específicos de la asignatura.*
- *La estructura de la materia.*
- *Los Planes de Estudios.*
- *El contexto académico.*
- *El contexto socio-profesional.*
- *Los conocimientos previos de los alumnos.*
- *Los contenidos de los programas de asignaturas relacionadas con la materia.*
- *La experiencia del profesor.*

A la hora de establecer los contenidos más adecuados ha sido preciso tener en cuenta unos criterios básicos de selección, como pueden ser los siguientes:

- *Congruencia con los objetivos previstos.*
- *Identificación de los núcleos básicos de la materia.*
- *Representatividad dentro del amplio conjunto de conocimientos.*
- *Posibilidad de transferencia entre los conocimientos proporcionados por la misma u otras materias.*
- *Validez y permanencia en el tiempo.*
- *Actualidad, para no perder de vista la evolución a la que puedan estar sometidos por el desarrollo de la técnica, la ciencia o la sociedad.*
- *Graduación secuencial, para avanzar de lo más simple a lo más complejo.*

En este sentido, se han definido los contenidos teóricos de las asignaturas de objeto de este Proyecto Docente siguiendo de forma importante los criterios que se acaban de señalar y las limitaciones existentes.

3.5 Métodos y técnicas

Una vez establecidos los criterios a tener en cuenta en la definición de los objetivos educativos y los contenidos que se incluyen en el proyecto, es necesario determinar que actividades de aprendizaje o métodos de enseñanza son los más idóneos para conseguir cada uno de los objetivos. En esta sección se pretende ver cómo se puede elaborar el programa de las materias objeto de este Proyecto Docente, tanto desde una perspectiva metodológica, como desde el análisis de las técnicas a disposición del docente para enfrentarse a su trabajo.

3.5.1 Criterios metodológicos en la elaboración del programa

Hay ciertos principios metodológicos básicos observables para la consecución del aprendizaje de los alumnos en las materias de estudio, así como para que éste sea exitoso. Estos principios pueden resumirse en los siguientes:

- *Continuidad*
- *Progresión continua de dificultad*
- *Dignidad en los contenidos y en su presentación*
- *Posibilidad de revisión*
- *Realismo en los contenidos*
- *Diversidad en la presentación*

3.5.1.1 Continuidad

Evitar, en lo posible, que partes de la materia queden como compartimentos aislados sin relación con el resto. Hay que fomentar una visión integradora e interdisciplinaria de los conocimientos científicos, posibilitando el ejercicio de las habilidades de síntesis. Este criterio implica la ausencia de saltos, lagunas o, lo que es peor, incongruencias en la materia que se imparte y que afecta, de manera directa, al planteamiento del programa de las asignaturas.

3.5.1.2 Progresión continua de dificultad

Conviene llevar al alumno siempre un paso por delante de su capacidad actual, pero no más. Así, el elemento motivador de superar las dificultades mediante el esfuerzo, no se convertirá en un muro de apariencia infranqueable que lo desanime. Este criterio entraña conflictos con el método habitual de ir desde lo que es general a lo que es particular, ya que es frecuente que lo

general entrañe mayor dificultad. Sin embargo, progresar de lo general a lo específico facilita la visión global del tema y ahorra tiempo en el aprendizaje.

En cada caso, hay que decir si es conveniente o no el empleo de esta técnica basándose en las características de los contenidos docentes particulares.

3.5.1.3 Dignidad de los contenidos y en su presentación

La necesidad del establecimiento de este punto es obvia para conseguir la competencia del alumno como profesional. A parte de esto, se considera una reacción normal del alumno la pérdida de interés por la asignatura cuando, por cualquier motivo, llega a juzgar sus contenidos como carentes de utilidad o como presentados de forma inadecuada. Por tanto, han de estar diseñados de forma motivadora y en clara relación con la realidad. En este sentido la planificación de los contenidos ha de ser:

- **Colectiva**: elaborada mediante trabajo en equipo.
- **Completa**: debe de dar un enfoque sistemático que interrelacione todos los elementos del programa.
- **Concreta**: los elementos estructurales básicos de un programa pertinente deben de ser las tareas profesionales concretas perfectamente definidas.
- **Integrada**: tanto por la inclusión de aspectos teóricos y prácticos, como por su coordinación con la programación de otras asignaturas, evitando la reiteración de materias o exposiciones incomprensibles por falta de conocimientos previos.
- **Motivadora**: la enseñanza es un proceso activo más eficaz cuanto más motivado esté el estudiante para aprender. Es importante planificar una enseñanza que sea capaz de despertar actitudes e intereses. Si la forma de enseñanza sólo se preocupa de desarrollar conocimientos y habilidades técnicas, por muy completa que sea la práctica de estos aspectos, nunca podrán sustituir a las actitudes y los intereses, es decir al compromiso personal del estudiante.

Conviene, especialmente tratándose de alumnos de cierta madurez, como es el caso, incluir breves reseñas sobre la conveniencia de la metodología empleada y del enfoque presentado. Esto, junto con la credibilidad que proporciona al alumno la comprobación de la capacidad del profesor, la cual constata diariamente, será suficiente para evitar la desmotivación en el sentido apuntado.

Se da por supuesto que lo anterior implica una labor de convencimiento del alumno, para lo que el profesor debe dotarse de argumentos convincentes, siendo el contenido del programa uno de los fundamentales.

3.5.1.4 Posibilidad de revisión

Hay que considerar, en todo momento, al contenido como un bien cultural susceptible de adecuación al momento que se vive. Este punto es especialmente notorio en lo concerniente a la docencia en Informática, debido a su rápida evolución.

Por tanto se ha de recoger tanto tendencias y líneas de investigación, como sobre todo aspectos de aplicación sobradamente extendidos en ese momento, revisando los contenidos para restar importancia a aquéllos que queden obsoletos.

3.5.1.5 Realismo en los contenidos

Los contenidos del programa deben, por un lado, ser adecuados al nivel de conocimientos que tienen los alumnos y a la utilidad de éstos para el desempeño futuro de su profesión.

Señalar la gran dificultad que existe, al tratar de elaborar el programa, por las diferencias de nivel de conocimientos entre los distintos alumnos originada por la diferente formación de los mismos (Formación Profesional, COU, LOGSE). Por otra parte, hay que procurar que sus contenidos se ajusten al tiempo real de que se dispone para su desarrollo.

3.5.1.6 Diversidad en la presentación

Se deben contemplar métodos pedagógicos diversos, de manera que el resultado sea un conjunto equilibrado, tanto en cuanto a las diferentes técnicas utilizadas, como a la aplicación de cada una de ellas en cada momento.

También es interesante la alternancia de las cuestiones teóricas con ejercicios u otras prácticas que den lugar a una visión más pragmática de la asignatura y faciliten la comprensión de sus contenidos.

En otro orden de cosas, es interesante recordar los tres tipos de enseñanza que existen según la metodología que se siga en el proceso de aprendizaje. Dichos métodos se deben considerar a la hora de establecer los contenidos del programa para combinarlos de forma adecuada. Estos métodos son los siguientes:

- **Didáctico**: Siguiendo este método, el profesor explica a los alumnos la realidad objetiva u objetivada que se supone posee y que es transmitida al alumno en el acto docente. Éste recibe de las clases más información que formación, privándole, por

tanto, del necesario proceso de deducción. El método didáctico tiene el inconveniente de que el alumno se ve abocado a una excesiva memorización debido a que no deja mucho espacio para su participación, pero, de otro lado, tiene la ventaja de que permite al profesor programar la enseñanza adaptándola al tiempo disponible para su desarrollo.

- **Dialéctico**: Implica la búsqueda de la verdad mediante el contraste de opiniones y enfoques distintos. En esta dialéctica, el profesor es el que tiene mayor responsabilidad y debe dirigir la discusión hacia los puntos de interés, pero con habilidad suficiente para que las conclusiones aparezcan como fruto de la discusión y del razonamiento en común. El profesor requiere de una gran capacidad de improvisación y asimilación que le permita mantener el tema dentro de los límites sustanciales sin que derive hacia cuestiones secundarias que, espontáneamente, surgen en el debate. Por tanto, este método exige de aquél un mayor esfuerzo y una adecuada preparación, así como un buen dominio de las materias, de forma que pueda hacer frente a cuestiones inesperadas sugeridas por los alumnos.
- **Heurístico**: Aquí el alumno es el que debe redescubrir (o descubrir, lo que también pudiera suceder) las soluciones por su cuenta, valiéndose de los conocimientos que ya tiene, realizando así un proceso de autoformación. Los temas que se tratarán son distribuidos entre los alumnos, pudiendo éstos agruparse o trabajar individualmente. La actividad del profesor es esencial para que el alumno no desvíe su atención hacia temas de su interés dejando sin actualizar aspectos relevantes. Para ello el profesor debe de controlar a los alumnos mediante una adecuada asignación de funciones a los mismos, así como la implantación de las directrices que deben seguir para el desarrollo de su trabajo.

3.5.2 Metodologías docentes para la ejecución del programa

Los métodos de docencia universitaria son los instrumentos por los que el profesor traslada los contenidos de una materia hacia el alumnado, con el fin de conseguir los objetivos propuestos. Por ello, no debe existir una disociación entre los contenidos y los métodos, ya que estos últimos tienen una función destacada en la configuración de los contenidos, por tanto, es un paso obligado en todo proyecto docente analizar los distintos métodos de enseñanza y su aplicabilidad en la materia correspondiente.

La aplicación de los diferentes métodos pedagógicos - didáctico, dialéctico y heurístico -, ya comentados anteriormente, a la enseñanza en las materias objeto de este Proyecto Docente

puede desarrollarse, básicamente, a través de clases teóricas, clases prácticas, seminarios y horarios de apoyo a los alumnos o tutorías.

Por tanto, en este punto se expone la forma en que se propone llevar a cabo las diversas actividades de la función docente. Se aborda también las cuestiones de método que son de aplicación en ámbitos distintos al de la elaboración del programa. No obstante, hay cuestiones de método que afectan tanto al programa como a otros aspectos de la función docente y, también, serán considerados aquí.

Actualmente prima la innovación en las técnicas docentes. En este sentido Fernando Lara [Lara, 1997] distingue en función de las técnicas docentes dos tipos de enseñanza:

- "*Una enseñanza en la que el profesor, desde 'la tarima' transmite la Ciencia; y el alumno, desde 'el asiento' la recoge en sus apuntes para estudiarla y para soltarla en el examen, quizás también para entenderla*".
- "*Una enseñanza en la que el profesor, además de transmitir la Ciencia, desea provocar el aprendizaje posterior del alumno, transmitiendo el interés personal por la asignatura, motivándole a seguir investigando sobre la materia, motivándole a formular preguntas que aclaren lo que no entiende, etc.*".

La pedagogía moderna, junto con la tecnología educativa, ofrecen en la actualidad una amplia gama de técnicas y recursos para el desarrollo del proceso enseñanza-aprendizaje. En general, se puede decir que cualquier técnica o método es válido siempre que se adapte a la actividad programada, a los objetivos propuestos y a las circunstancias. En realidad, método existe siempre; se trata de elegir el mejor para cada circunstancia, sólo así los contenidos serán transmitidos con el mejor nivel de eficacia y rentabilidad respecto a la inversión educativa a que se refiere. Como indica la UNESCO [UNESCO, 1973], "*la educación debe poder ser impartida y adquirida por una multitud de medios, ya que lo importante no es saber qué camino ha seguido el sujeto, sino lo que ha aprendido y adquirido*". El fin, y no el medio es, en consecuencia, el punto de mira esencial. Pero ello no implica, ni mucho menos, que el medio sea indiferente, como quizás pudiera deducirse de modo erróneo de las frases de la UNESCO, sino por el contrario:

- *Es el objetivo perseguido el que condiciona la técnica más adecuada.*
- *Las técnicas no son buenas ni malas, sino que pueden estar bien o mal aplicadas en relación con el propósito que se presenta, en función del espíritu que las impregne y, en la medida en que se apliquen de un modo activo, propiciando el ejercicio de la reflexión, el espíritu crítico del alumno...*

- *Si bien pueden existir varias técnicas aplicables a un objeto educativo concreto, siempre podrá aludirse a técnicas más apropiadas y a otras menos adecuadas.*

Por tanto, la ejecución de la acción docente puede ser albergada por diversas técnicas o escenarios, cada uno de los cuales puede llegar a ser apropiado en mayor o menor grado para cada punto concreto del programa. Es por tanto, tarea del profesor distribuir en estos escenarios la materia, tomando como criterio de decisión los medios, la calidad y cantidad de alumnos, los condicionantes temporales y económicos, y la medida subjetiva de la pertinencia de la aplicación de un determinado método docente a una determinada materia.

A fin de referenciar cada una de las actividades docentes a lo largo de la exposición de los programas de las asignaturas, se procede a enumerarlas:

1. Clase teórica o lección magistral.
2. Clases de problemas.
3. Clases prácticas.
   3.1. Prácticas guiadas.
   3.2. Prácticas libres.
4. Actividades docentes complementarias
   4.1. Seminarios y conferencias.
   4.2. Visitas y prácticas en instalaciones y centros profesionales.
   4.3. Tutorías.
   4.4. Internet como vía de comunicación con los alumnos.

A las técnicas mencionadas se han de añadir (cualquiera que sea la modalidad escogida), aquellas actividades complementarias que, para un acto concreto, se consideren necesarias, tales como lecturas complementarias, búsqueda de bibliografía, confección de trabajos, asistencia a conferencias...

Antes de iniciar el desarrollo de cada una de las técnicas o modos con los que ha de impartirse el conocimiento, se comentan los principios que han de tenerse en cuenta cara a su elección:

- Se adecuarán a los objetivos pretendidos en cada una de las partes del programa, lo que implica la selección de diferentes medios o recursos para los diferentes contenidos.

- Del punto anterior se desprende que la congruencia entre medios y fines es algo que se debe tener siempre presente.
- Que transmitan no sólo conocimientos sino también procedimientos, esquemas de razonamiento, mecanismos de aplicación, generalización y síntesis y, en definitiva, metodología científica.
- Que permitan al alumno desempeñar un papel activo (documentarse, exponer, observar, participar...).
- Que estimulen al alumno la necesidad de aprender y la iniciativa para la aplicación de sus conocimientos a problemas reales.
- Que presenten una cierta flexibilidad para que el alumno pueda tomar decisiones razonables respecto a cómo desarrollarlas.
- Que fomenten tanto el trabajo individual como en equipo.
- Que puedan ser cumplidas por la gran mayoría de los alumnos, teniendo en cuenta sus diversos niveles de capacidad y sus diferentes intereses.
- La selección debe ser equilibrada, repartiéndose a lo largo del curso de forma ponderada. Resulta muy arriesgado realizar estimaciones con carácter fijo sobre lo que cada técnica debe ocupar en el desarrollo del programa.

En todo caso, se ha de tener en cuenta que la adecuada combinación de estas técnicas ha de contribuir a la creación de actividades críticas, reflexivas y analíticas en los alumnos; de tal forma que puedan llegar a obtener una visión equilibrada e integradora de los distintos aspectos que conforman la realidad estudiada. Se trata, pues, de incorporar una **metodología activa, globalizadora y participativa** en la medida en que las condiciones en las que el desarrollo de la actividad docente, lo permita. De ahí que, muchas veces, no se pueden hacer extensivas estas técnicas de trabajo a la totalidad de alumnos, llegando únicamente a aquéllos que, encontrándose motivados, acepten las mismas voluntariamente, de tal forma que, con este punto de partida, el éxito de las mismas esté casi garantizado.

En general, dentro de las actividades que debe realizar el profesor se suelen considerar tres categorías básicas: *explicación, motivación* y *orientación*.

La ***explicación*** es la base de la transmisión de los conocimientos; aunque en ella es el profesor el que realiza la parte más activa, hay que intentar evitar que el alumno se sienta como elemento meramente pasivo.

La ***motivación*** tiene una gran importancia puesto que influye fuertemente en la capacidad receptiva del alumno. De poco sirve realizar un gran esfuerzo en que la transmisión sea correcta si falla la recepción. Favorecer este aspecto es por tanto esencial para el rendimiento de la actividad docente. Las formas de hacerlo pueden ser, entre otras:

- Dejar constancia de los objetivos que se buscan en cada momento.
- Utilizar un lenguaje claro, directo y conciso.
- Poner ejemplos reales y hacer comentarios que despierten su interés.
- Averiguar qué experiencias comunes pueden utilizarse como estímulos para el aprendizaje.
- Utilizar los medios y el material que se consideren estimulante.

La ***orientación*** constituye también una labor fundamental. No hay que olvidar que, en último término, el factor decisivo en el aprendizaje es el trabajo personal de los alumnos. El docente da las pautas y después debe mantener hacia ellos una orientación que les permita trabajar solos, de acuerdo con su propio ritmo y al plan de trabajo que tienen trazado.

Además de estas tres actividades básicas, el profesor debe realizar una importante labor personal, como es *la preparación y actualización del material, la revisión de bibliografía, la puesta al día de sus conocimientos, la revisión de su programa*... Este continuo perfeccionamiento personal es fundamental en todas las áreas del saber, pero especialmente necesario en las Ingenierías, en los que la evolución es vertiginosa. En este sentido se considera de gran importancia que el docente realice también tareas de ***investigación***, puesto que ello tendrá influencia y una repercusión directa en su docencia.

La ***materialización*** de la actividad docente en las disciplinas universitarias se realiza a través de las clases (teóricas o de problemas), las prácticas y otras actividades complementarias, como los seminarios, tutorías... En los siguientes apartados se introducen brevemente cada una de las técnicas didácticas, haciendo hincapié sobre sus ventajas e inconvenientes, así como de la forma más eficaz de llevarlas a cabo.

3.5.2.1 Clases Teóricas o lección magistral

En la enseñanza universitaria, la lección magistral es la técnica de trabajo más antigua. De hecho, como método de enseñanza, nace con la misma Universidad, en la época medieval. Recoge la idea de *lectio* de las escuelas monacales, es decir, la lectura y comentario de un texto elegido como base de un curso. Actualmente, se ha convertido en la técnica más extendida y,

lamentablemente, en muchos casos, la única, al existir limitaciones importantes, tales como *la masificación del alumnado, la dependencia del Plan de Estudios, los medios...*

Su misión es la exposición completa, sistemática y ordenada del programa de la asignatura a lo largo del período lectivo de un curso académico.

Se trata de un tipo de enseñanza ocupada entera o principalmente por la exposición continua del docente. Aun cuando los estudiantes pueden preguntar o participar en una cierta discusión, su actividad fundamental es escuchar y tomar notas. La parte activa corresponde al profesor y presenta un carácter fundamentalmente instructivo. La Ciencia se ofrece bajo la forma de una definición, solución o resultado, teniendo por tanto una enseñanza primordialmente temática.

Seguramente sea éste el método que más polémica despierta entre alumnos y profesores, existiendo tantos detractores como defensores del mismo. Sin embargo, cuando aparecen los condicionantes anteriormente mencionados resulta difícil pensar en métodos más personalizados, o en los que se proponga un pleno contacto con el estudiante.

En los últimos años ha habido una fuerte tendencia opuesta a esta idea de clase magistral, y se ha defendido la denominada ***clase activa***. Lo que en principio es una idea válida, posibilitar que el alumno sea protagonista de su propio aprendizaje, se ha llevado en ocasiones a extremos exagerados. Además, hay un problema muy claro y que limita fuertemente la aplicación de esta idea: *el número de alumnos por clase*; lo que para un grupo pequeño puede ser correcto, pasa a ser totalmente inviable a medida que aumenta la cantidad de participantes. En realidad, clase magistral y clase activa, bien entendidas, deberían complementarse, y procurar, dentro de lo posible, una participación activa del alumnado, pero con la constante intervención del profesor, que siempre tendrá un papel importante que no podrá ser sustituido por los textos, ni por los modernos métodos audiovisuales o informáticos.

Inconvenientes de la lección magistral

A pesar de las críticas a las que se ha sometido, al descansar únicamente en la iniciativa del profesor, sigue siendo la pieza fundamental de la enseñanza, pues es el único momento en donde se hace una exposición coherente y completa de la materia, y donde el profesor tiene mayores posibilidades de influir sobre la comprensión de los conceptos por parte de los alumnos. Es por ello que en [Hale, 1964] se define la lección magistral como "*un tiempo de enseñanza ocupado entera o principalmente en la exposición continua por parte del profesor*". Es en este punto, donde radican fundamentalmente los inconvenientes de la lección teórica; esto es, en la dificultad de conseguir una participación activa del alumno.

Se trata por tanto, de un método pasivo, debido a que el alumno se limita a tomar apuntes, y se preocupa básicamente de que éstos reflejen fielmente la explicación del profesor; lo que supone un doble esfuerzo, pues posteriormente debe invertir gran cantidad de tiempo en asimilar los conceptos que se ha limitado a copiar en clase.

Esta pasividad, a la que se ve sometido normalmente el alumno, baja efectividad en la transmisión de los conocimientos, favorece la repetición, la omisión del sentido crítico, la rutina en la docencia y la ausencia de estímulo para el alumno, ya que generalmente son poco amenas. Al haber un único interlocutor, fluyen rápidas con pocas interrupciones, con lo que los estudiantes quedan abrumados por la cantidad de conocimientos que le son propuestos.

Por otra parte, se acentúa la idea de que el profesor es la única fuente del saber, se expone sólo la visión personal del profesor sobre el tema, creándose una dependencia didáctica total. Como consecuencia, el alumno no adquiere el hábito de manejar bibliografía, ni desarrolla capacidad de síntesis ni de crítica.

Como inconveniente adicional, el profesor no tiene forma de conocer las características individuales de cada alumno, su formación previa. De esta forma se imparten los mismos conocimientos, al mismo ritmo y tiempo a todos los estudiantes por igual, obviando cualquier tipo de tratamiento personalizado o adaptados a las circunstancias concretas de cada estudiante.

Adicionalmente, esta falta de tratamiento personalizado redunda en la dificultad de controlar el proceso de aprendizaje de cada alumno. Esta ausencia de control, que permitiría comprobar de una forma continuada como se asimila el conocimiento, se manifiesta en que la única comprobación posible viene asociada normalmente a un examen, que difiere dicho control a un instante en el que el profesor no puede reaccionar.

Ventajas de la lección magistral

Hay que destacar, como principales ventajas, que es un buen método para introducir al estudiante en los conocimientos fundamentales de una materia. Ofrece al estudiante la posibilidad de disponer, sin demasiado esfuerzo por su parte, de información básica y actualizada sobre el tema, puesto que la labor de recopilación y estructuración recae sobre el profesor.

Así, las lecciones pueden presentar materia que no está aún en la bibliografía genérica de la asignatura. Por ello, se hace necesaria su aplicación en aquellas disciplinas o partes de las mismas, en las que no existe apenas documentación al alcance del alumno.

Cuando por el contrario, la documentación sobre la materia es excesivamente abundante, el alumno en general agradece que, en la lección magistral, el profesor seleccione aquellos textos más recomendables.

Es, además, un método humano que puede ser de gran dinamismo dependiendo de las características del profesor. En este sentido, puede llegar a tener una fuerte capacidad motivadora en cuanto relaciona a unos profesores con una sólida vocación intelectual y a unos alumnos que se están iniciando en ella. Es, por tanto, fuente de una relación personal básica para una acción tutorial posterior.

Por otra parte, para la organización universitaria, es el medio más barato, pues no se necesita una gran dotación de recursos humanos y económicos. Esto se debe a que:

- Normalmente no es necesario más que la pizarra, en algunos casos acompañada por medios audiovisuales. Por tanto, no conlleva los costes de, por ejemplo, una práctica de laboratorio. Además, permite un cierto nivel de masificación que es impensable en otro tipo de técnicas, reduciendo el coste de recursos humanos.
- Permite una exposición más rápida de la materia por número de alumnos que cualquier otro método pedagógico.

Además, según Fernando Lara [Lara, 1997] la clase magistral a veces es más activa de lo que parece, pues el alumno al tomar apuntes traduce a su forma de pensar los contenidos que explica el profesor, si bien es cierto que esta afirmación, sólo se verifica mientras el alumno no haya perdido el hilo de las explicaciones.

Lección Magistral	
Ventajas	**Inconvenientes**
Exposición completa, sistemática y ordenada	Dificultad para conseguir la participación del alumno
El alumno dispone de información básica y actualizada	Acentúa la idea de que el profesor es la única fuente del saber
Puede llegar a tener una fuerte capacidad motivadora	No existe tratamiento personalizado del alumno
No es necesario una gran dotación de personal ni de medios	Dificultad de controlar en proceso de aprendizaje

Tabla 3.1. Ventajas e inconvenientes de la lección magistral

Según los resultados de algunas encuestas [Beard, 1974], los estudiantes aceptan las clases magistrales, cuando son claras y constituyen resúmenes ordenados en los que se destacan los aspectos esenciales de cada tema.

En definitiva, la lección magistral puede ser perfectamente válida siempre que esté, por una parte, bien planteada y preparada, y por otra, se complete con actividades más personalizadas dentro de las limitaciones existentes.

Aspectos influyentes en la calidad de la lección magistral

A pesar de que la lección magistral, como método pedagógico, se ve hoy en día cuestionada tanto por educadores como por estudiantes, desde una perspectiva realista la mayoría de las veces resulta difícil pensar en métodos de enseñanza alternativos allí donde los recursos humanos y materiales disponibles son escasos en relación con el número de alumnos. Por ello, hay que asumir y aceptar que éste seguirá siendo un método muy empleado en la enseñanza universitaria, de lo que surge la necesidad de conocer aquellos aspectos que contribuyen a su excelencia, o que por el contrario acentúan sus carencias.

Un aspecto importante de la lección magistral es su duración. Existen estudios que indican que la atención disminuye a partir de los 40 minutos, precipitándose de forma más intensa a partir de los 60 minutos, si bien, esto está sujeto a diferentes factores, como el horario, clases previas... Por ello, la duración de la lección teórica no debería exceder los 45-50 minutos. Destacan en este sentido los estudios realizados por D. H. Lloyd [Lloyd, 1968] sobre los altibajos en el rendimiento del alumno y del profesor a lo largo de una lección magistral tal y como muestra la Figura 3.2.

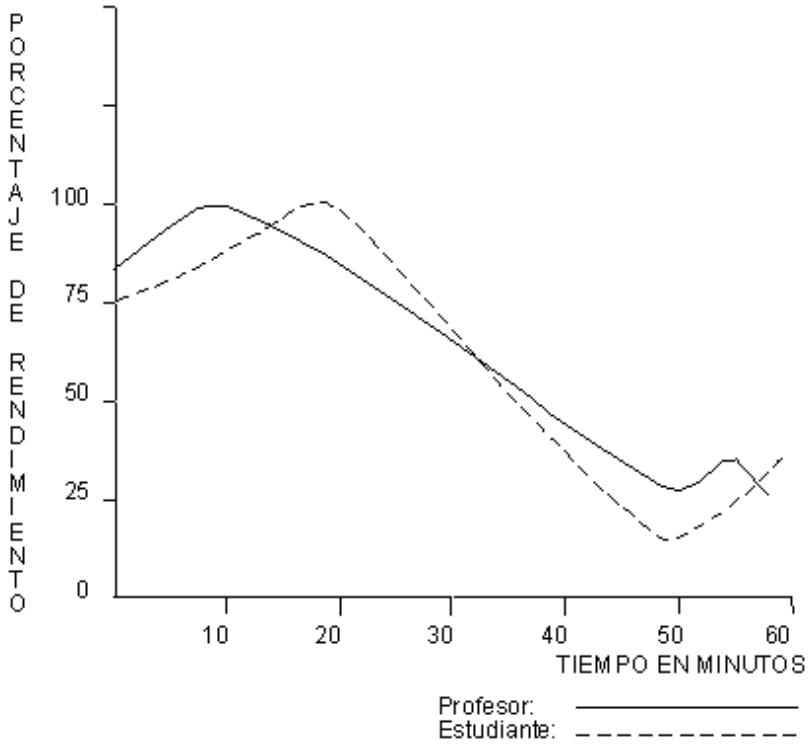


Figura 3.2. Rendimiento de Profesores y Alumnos a lo largo de una Lección Magistral

El relajamiento de la pendiente de estas curvas puede verse parcialmente favorecido por la utilización de técnicas audiovisuales que rompan la uniformidad de la oratoria del profesor. Generalmente los medios de transmisión del mensaje didáctico se dividen en:

- **Medios escritos**: recogen el conocimiento de una disciplina, y sirven para la formación del programa docente del profesor, así como de bibliografía de consulta del alumno.
- **Medios auditivos**: son los basados en la comunicación oral: charlas, clases magistrales, seminarios, unidades docentes en soportes de audio...
- **Medios audiovisuales**: son los relacionados con pizarras y medios proyectados, principalmente transparencias, pantallas de ordenador para retroproyector, cañón...

Según los expertos, el porcentaje de participación del sentido de la vista es del 83% en el proceso de aprendizaje, frente al 11% del oído. El número de datos retenidos en función del medio es mayor en situaciones en que se ve y se oye, que en el caso de ambos por separado, como se puede apreciar en los datos recogidos en la Tabla 3.2.

	Tiempo transcurrido	
Medio	**3 horas**	**3 días**
Oral	70%	10%
Visual	72%	20%
Audiovisual	85%	65%

Tabla 3.2. Duración del recuerdo de los contenidos de una exposición en función del tipo de medio utilizado

Parece conveniente, entonces, utilizar aquellos medios audiovisuales que permitan ilustrar la explicación y agilizar el desarrollo de la clase, sobre todo en temas con gran contenido de esquemas, diagramas y demás representaciones gráficas, que de otro modo pueden ocasionar confusiones, pérdidas de tiempo y concentración por parte de los alumnos y del profesor, al ser dibujadas manualmente sobre la pizarra.

Para ello, es adecuado facilitar con suficiente anterioridad el material utilizado, para que el alumno pueda disponer de él durante la clase. De esta forma, los estudiantes están en condiciones de seguir adecuadamente la explicación, sin que desvíen su atención en trasladar las indicaciones del profesor a sus apuntes, pudiendo completar la documentación aportada con breves anotaciones durante el desarrollo de las clases.

El poseer de antemano la documentación de la clase permite al alumno leerla con anterioridad (utópicamente hablando) con lo que el seguimiento de la clase se hace más

llevadero. Pese a esta ventaja, el profesor ha de vigilar los peligros de suministrar documentación a los alumnos, que fundamentalmente son:

- Ausencia de interés por completar el material y, por tanto, de manejar bibliografía, así como de aprender a mantenerse al día por su cuenta en esa disciplina; pues el alumno tiende a asumir que el profesor en ningún caso va a exigirle desarrollar contenidos más allá de la documentación que éste le aporta. Este punto se hace más notorio cuando la documentación consiste en los llamados *apuntes del profesor*, que cuando, por el contrario, se trata de un conjunto de ilustraciones, esquemas, transparencias y referencias bibliográficas.
- Clases excesivamente veloces, pues el profesor al no verse frenado en sus explicaciones por la velocidad con la que los alumnos toman apuntes y/o copian el contenido de la pizarra, tiende a comprimir gran cantidad de materia en una sola clase. De esta forma se deja poco tiempo para la reflexión y, sobretodo, se fatiga al alumno, quien acaba por desconectar mucho antes del final de la clase. Por ello, una opción interesante en relación con la utilización de medios audiovisuales, consiste en basar el desarrollo de la clase utilizando la pizarra y usando transparencias como medio de apoyo, para presentar diagramas o esquemas.

Finalmente, la estructura de la lección magistral se constituye como un factor altamente influyente en su calidad. El seguimiento de una estructura correcta es un buen comienzo sobre el que construir la lección. Al inicio de la clase es conveniente dar una visión general del tema que permita seguir la exposición con facilidad. Al final de la clase, es aconsejable dar una conclusión que resuma lo expuesto. Si todavía no se ha llegado a dicha conclusión se debe de remarcar hacia donde se pretende llegar y mostrar el camino recorrido.

Aspectos que influyen en la calidad de la lección magistral
Duración de 45-50 minutos
Utilización de medios audiovisuales
Entregar al alumno de forma anticipada el material que se va a utilizar
El material entregado debe ser ilustraciones y esquemas que el alumno pueda completar durante la exposición
Estructura y exposición correcta que suscite el interés del alumno

Tabla 3.3. Aspectos influyentes en la calidad de la lección magistral

Entre el principio y el final debería de discurrir una exposición teórica correcta que suscite en el alumno el interés por el tema, motivándolo en el aprendizaje de la materia; avanzando con razonamientos claros, mostrando la relación entre los conceptos precedentes y/o consecutivos, y

enfocada hacia los temas fundamentales, consiguiendo que el alumno no pierda en ningún momento una visión global de la asignatura.

La correcta exposición

La exposición se erige en el eje fundamental de la lección magistral, por este motivo se ha dedicado un apartado concreto en el que tratar sus principales aspectos.

Para que los alumnos acepten, por tanto, la lección magistral, se hace necesario el correcto desarrollo de la exposición de la misma, para lo que se deben tener en cuenta los siguientes aspectos:

- **Orden**: salvo excepciones, la exposición debe seguir un esquema previamente establecido, aunque sin rigidez. En ningún caso se debe improvisar.
- Se debe intentar exponer los temas de forma **completa, sistemática** y **ordenada**. Cada tema forma una entidad completa pero no autosuficiente, y no puede aislarse de los demás. Por tanto, su exposición debe hacerse dentro del contexto global de la unidad docente a la que van destinados, y por consiguiente, de la asignatura. En este sentido, es necesario que el alumno disponga al principio del curso del temario completo que se va a desarrollar, para facilitarle el seguimiento de las clases, y que le sirvan de ayuda en su posterior estudio, de manera que pueda comprender de forma global la disciplina.
- **Motivación**: se trata de atraer la atención del alumno, creando expectativas respecto a lo que se va a exponer (recursos: alusión a experiencias personales, suscitar problemas, cuestiones, aplicaciones, ejemplos oportunos, material audiovisual...).
- El profesor ha de **apoyarse en aquellos medios que sean apropiados al contenido** y ajustarse al mismo (no divagar). No se debe descuidar aspectos tales como al rigor, la precisión, la claridad y la amenidad en las explicaciones; evitando la monotonía, no sólo en el fondo, sino también en la forma, variando la entonación y ritmo de la exposición, (movimientos, gestos, entonación y ritmo adecuado; establecer pausas para la reflexión y resolver dudas).
- **Objetivos**: es interesante poner en conocimiento de los alumnos los objetivos (a dónde se quiere llegar), su relación con otras materias, límites del conocimiento sobre el tema... Como ya se ha comentado, el inicio y final de la clase es un momento ideal para remarcar los objetivos. No obstante puede ser conveniente refrescar a lo largo de la exposición la perspectiva real del desarrollo de la lección.

- **Resumen**: es importante resumir de vez en cuando los aspectos importantes, recapitular a lo largo de la exposición y, sobre todo, realizar una síntesis que indique al alumno qué es lo más importante del tema tratado. Los resúmenes no sólo han de hacerse al final de cada lección/clase, ya que el resumen le sirve al profesor de recapitulación para recalcar lo que considere importante en cada momento, además de aclarar lo que haya detectado que no se ha entendido sobradamente.
- **Prácticas**: es también importante, siempre que sea posible, ejemplificar la teoría y presentar situaciones prácticas. Evidentemente, se optimiza el método, si se realizan prácticas fuera de clase sobre el tema.
- **Capacidad de comunicación**: mantener una actitud abierta y relajada para conseguir una comunicación con los estudiantes, de forma que les incite a plantear preguntas en clase; invitando al alumno a exponer las dudas que le surjan durante la clase, siempre y cuando no se perjudique el ritmo de la exposición.
- **Observación y comprensión hacia el esfuerzo y capacidad del alumno**: el profesor debe ser capaz de, en todo momento, sondear el seguimiento de sus explicaciones por parte de la mayoría de los alumnos, insistiendo sobre los conceptos fundamentales hasta que queden suficientemente claros. Es preferible incumplir la totalidad de los contenidos finales que se habían puesto como meta en el programa, que por cegarse en mantenerlos, forzar la marcha, huyendo hacia adelante, pensando que "*cuando los alumnos se lo estudien, ya lo entenderán*". Por el contrario, el hecho de que unas explicaciones están construidas a partir de otras anteriores, provoca que la falta de compresión de los conceptos ya explicados haga inútil el esfuerzo del profesor por introducir los nuevos.

Se debe de evitar por norma, que la explicación de los conceptos importantes coincida con el final de la clase, ya que esto conlleva dos peligros:

- Dejar la explicación de dichos conceptos inacabada.
- Acabar la explicación, pero haciéndola coincidir con el momento en que los alumnos están menos receptivos. Esta falta de receptividad se ve motivada fundamentalmente por el cansancio y, en el caso de que los conceptos a explicar estén basados en otros introducidos en la misma clase, en la falta de un período de reflexión/maduración que permita asimilar los nuevos conceptos a partir de los precedentes.

3.5.2.2 Clases de problemas

Un aspecto importante dentro de la enseñanza son las clases de problemas. En ellas, el alumno debe aplicar los conocimientos teóricos adquiridos para resolver problemas o supuestos prácticos, constituyendo un eficaz factor de realimentación de los mismos. Su adecuada inclusión dentro del desarrollo de la materia permite reforzar y aplicar los conceptos expuestos en teoría, y fomentar en el alumno la capacidad de análisis y síntesis.

Por tanto, no parece conveniente establecer una división tajante entre clases teóricas y de problemas, sino entremezclar ambas, de modo que las exposiciones teóricas se alternen con ejercicios ilustrativos o aclaratorios. Asimismo, el profesor puede obtener una información muy valiosa de estas clases, ya que le permiten detectar dificultades de comprensión y aplicación de los conceptos teóricos.

Parece interesante proponer, para afianzar los conocimientos teóricos, un conjunto de problemas, junto con alguna sugerencia de cara a su solución. Estos problemas deben ser resueltos por los alumnos mediante su trabajo personal. De esta forma el alumno completará el proceso de aprendizaje, razonando y enfrentándose a dudas y a conceptos poco claros, que de otro modo no se habrían presentado. Posteriormente, el profesor puede averiguar cuál es el grado de éxito en su solución, y orientar y resolver los más complejos con los alumnos. En caso de dificultades especiales, es aconsejable organizar un seminario para tratar estos problemas.

Los enunciados de los problemas, así como las soluciones aportadas por los alumnos y corregidas por el profesor pueden dejarse disponibles en Internet, si éste último lo estima conveniente.

3.5.2.3 Clases de prácticas

Las prácticas constituyen uno de los complementos ideales e imprescindibles de las clases teóricas y de problemas en la educación universitaria. Esto es más acusado en las disciplinas relacionadas con la Informática.

La realización de las clases prácticas sirve al alumno para adaptar los conceptos teóricos estudiados en clase, en el entorno real de aplicación de éstos, sobre casos concretos. El profesor debe tomar parte activa en la clarificación de las posibles dudas o controversias que surjan, pues es en ese momento cuando el estudiante está más receptivo. Asimismo, debe participar como observador atento a las respuestas de los alumnos, con objeto de conseguir evaluar sobre la marcha la adecuación del programa docente y de la técnica de enseñanza utilizada. Para mejorar el rendimiento es conveniente que el alumno disponga previamente de los supuestos prácticos que debe analizar y desarrollar.

Respecto del desarrollo de las prácticas, se debe indicar primero que, dado el elevado número de alumnos es inevitable dividir al alumnado en grupos de prácticas. En concreto, al tener que desarrollar esta docencia a través de prácticas con ordenador, se tiene que fijar un número máximo de alumnos por ordenador, dato este que, junto con la capacidad del aula, permitan obtener el número de alumnos por grupo.

Se puede pensar que el trabajo de dos y tres personas por ordenador es negativo frente el trabajo individual. Sin embargo, no es menos cierto que este hecho facilita la distensión de la dinámica de la clase, haciendo posible una relación fluida entre el profesor y los alumnos, y entre los propios alumnos, permitiendo la discusión dentro del grupo y el seguimiento de la labor de cada alumno por parte del profesor. Por otra parte, es muy probable que su futura actividad laboral se desarrolle dentro un equipo formado por varios profesionales.

Existen dos planteamientos distintos a la hora de desarrollar las prácticas: *Prácticas Guiadas* y *Prácticas Libres*. En ambos tipos, la experiencia dice que los estudiantes acogen las prácticas con un elevado interés. De cualquier modo, se debe contar, por supuesto, con los medios necesarios para llevarlo a cabo.

Prácticas guiadas

Son aquéllas que son dirigidas directamente y en todo momento por el profesor. Se corresponden a prácticas en las que se introducen entornos de trabajo o utilidades que son nuevas para el alumno.

En este tipo de prácticas es conveniente hacer notar a los alumnos la conveniencia de *parar* al profesor en todo momento en que sean incapaces de ejecutar una determinada acción con éxito en el ordenador; pues de lo contrario, se arriesgan a quedar desconectados del seguimiento de la clase.

Por este motivo, en este tipo de prácticas es contraproducente aumentar el número de terminales por grupo, ya que el profesor tiene que hacer un seguimiento estrecho del avance paso a paso de los alumnos, haciéndose más probable que los alumnos de un determinado puesto se "*atasquen*", cuantos más puestos haya. Cada vez que los alumnos de un puesto tienen un problema en este tipo de prácticas, requieren normalmente la atención exclusiva del profesor para resolverlo; provocando que el resto de alumnos se queden esperando a que el profesor pueda continuar guiando el resto de la práctica.

Prácticas libres

Otro tipo de prácticas son aquéllas en las que el profesor presenta brevemente la práctica, indicando sus objetivos y cómo debe progresarse en la misma, de modo que el alumno, ayudado del guión que se le ha proporcionado con anterioridad, la lleve a cabo. El profesor debe actuar como director, moderador y observador, atento ante las dudas o problemas que surjan, cuidando que los alumnos progresen de forma simultánea.

En cuanto a la organización temporal de las sesiones, será tal que las prácticas siempre sean posteriores a la explicación teórica de los conceptos, y de forma que se puedan realizar íntegramente con una continuidad temporal en temas afines que las haga más provechosas.

Esto tampoco quiere decir que sea conveniente introducir absolutamente todos los conceptos en las clases teóricas previas a las prácticas, pues de lo contrario las prácticas se convierten en una mera verificación por parte del alumno de lo que se le ha enseñado en la teoría, perdiéndose así el dinamismo y la participación que surge cuando intencionadamente el profesor no presenta todos los conocimientos necesarios para resolver dicha práctica.

El profesor de esta forma queda a la espera de que el alumno descubra por sí solo la dificultad de un determinado problema, a sabiendas de que los alumnos, normalmente, fracasarán en el intento de realizarlo por ellos mismos. De esta forma se tiene que, por un lado, el alumno descubre cómo no ha de resolverse el problema y, por otro, se fuerza a que acabe solicitando la ayuda del profesor para poder seguir trabajando. Es en este momento cuando el profesor debe de aprovechar para explicar los conceptos que vengan al caso y que intencionadamente se dejaron sin matizar en la exposición teórica, pues el alumno ha reflexionado sobre el problema y, además, suele estar más receptivo de lo habitual.

En algunos casos puede ser conveniente hacer presentar al alumno un informe detallado con los resultados y las conclusiones obtenidas de cada una de las prácticas. Este informe puede ser presentado de forma individual, a pesar de que las prácticas se hayan realizado en grupo. De este modo, cada estudiante se ejercita en la ordenación sistemática de los conocimientos y aprende a estructurar un trabajo.

Los enunciados de las prácticas, así como los informes de las prácticas corregidas pueden dejarse disponibles en Internet si el profesor lo estima conveniente.

3.5.2.4 Actividades docentes complementarias

Las actividades complementarias contribuyen a mantener el interés del alumno y a favorecer un contacto más directo con el profesor y con el mundo profesional, propiciando un sistema de educación adicional muy útil para su formación.

Seminarios y conferencias

Existen temas que no pueden ser tratados en toda su extensión durante un curso académico, bien por la propia limitación del tiempo asignado a la asignatura, bien porque un adecuado tratamiento de los mismos requiere del concurso de expertos, o bien por la inexistencia de instrumentos o medios especiales. En estos casos parece oportuno la utilización de seminarios o cursos monográficos.

Los seminarios y conferencias son reuniones organizadas con el propósito de incrementar el conocimiento general, completando la formación del alumno en el campo de los conocimientos y de la práctica profesional. Estas actividades son útiles para el fomento de la participación del alumno en la educación.

Asimismo, permiten la exposición de temas con un enfoque diferente al de la clase habitual. En ellos pueden presentarse aspectos muy específicos, avanzados o de interés general, y también una visión clara del mundo laboral.

Los seminarios, en concreto, se pueden plantear desde varios enfoques. En primer lugar, el profesor puede exponer un tema específico y, posteriormente, debatirlo con los alumnos en un ambiente más distendido e informal que el de las clases, intentando que los alumnos expongan sus dudas, comentarios o sugerencias. La discusión puede estar precedida de una conferencia o simplemente de algunas observaciones del individuo que representa el papel de director. Éste con antelación habrá preparado un esquema general para dirigir la discusión hacia determinadas metas.

Otra opción consiste en que sean los propios alumnos los que presenten un tema, previamente preparado, para proceder a continuación a su debate y discusión con el resto de los alumnos y el profesor. Este tipo de actividades sirve para fomentar las facultades expositivas de los alumnos y promover la crítica y la creatividad.

Operando de esta forma, se logran los objetivos que se deben confiar al seminario, que principalmente son:

- **Crear el hábito de investigación científica**: inculcar el espíritu científico, desarrollar en los alumnos la técnica del pensamiento crítico y del pensamiento

original. No obstante, hay que ser cuidadosos en este punto y no olvidar que, en el caso que se discute, el profesor debe de intentar guiar los trabajos hacia aspectos tan concretos como prácticos y evitar que el alumno derive hacia temas demasiado teóricos y/o abstractos que desborden su capacidad actual de análisis. En este mismo sentido, se observa que las preguntas de los alumnos durante el seminario disminuyen drásticamente cuanto menor sea el cariz práctico de los mismos.

- **Aprendizaje de los métodos científicos**: se trata de enseñar a manejar los instrumentos del trabajo intelectual; entre ellos destacan el manejo de bibliografía y la experimentación con los medios existentes.
- **Mejorar las capacidades de expresión escrita y oral**: el alumno tiene que elaborar trabajos escritos y defender sus puntos de vista. El profesor velará:
    1. Por que los trabajos no sean recortes de las fuentes bibliográficas consultadas, y que detrás de ellos exista una auténtica labor de síntesis y de crítica. Los trabajos escritos que sirvan de base a los seminarios, pueden igualmente ser publicados en la página web de la asignatura para que el esfuerzo de los alumnos sea accesible a las promociones venideras.
    2. Por que el alumno realice exposiciones directas, prescindiendo de los elementos que sean innecesarios para comprender la conclusión final de su trabajo.

El papel del profesor en los seminarios se identifica más con el de coordinador y moderador que con el de estrictamente docente. Así, el profesor con su presencia, favorecerá la creación de un clima de confianza, siendo el encargado de mantener la discusión dentro de sus límites, minimizando el debate en asuntos que no tengan importancia, relacionando una discusión con las anteriores y animando a los alumnos e invitados a participar, procurando que la discusión no sea monopolizada por unos pocos. El profesor debe resumir y cerrar cada tema discutido.

De cualquier modo, siempre que se realicen este tipo de trabajos, se debe contar, por supuesto, con los medios necesarios para llevarlo a cabo. Es necesario disponer de una biblioteca bien dotada, acceso a Internet y del equipo especializado en el aula de ordenadores.

Por otra parte, se pueden organizar conferencias con la colaboración de profesores de la propia titulación, o de otras titulaciones, o incluso de otras Universidades, y otras personas del mundo profesional. Éstos últimos suscitan un interés especial, ya que los conferenciantes conocen de cerca las exigencias y peculiaridades del mundo laboral, poniendo en sus enseñanzas un vigor y autenticidad que es muy apreciada por los estudiantes. Esto permite al

alumno conocer puntos de vista diferentes, problemas y últimos adelantos, que le aproximen al entorno en que realizará su ejercicio profesional.

En general, los estudiantes acogen estas actividades docentes con un marcado interés, si bien:

1. Para lograr una planificación correcta de los seminarios y conferencias, hay que tener en cuenta la disponibilidad de tiempo de los alumnos, de acuerdo con sus horarios de clase y otras actividades docentes. Hay que tener presente, por ejemplo, que si bien pueden organizarse seminarios y conferencias al principio de curso, pues se dispone de más tiempo para ello, el nivel de conocimiento en esta época es escaso. No obstante, al principio pueden plantearse aspectos de interés general, para organizarse paulatinamente sesiones sobre aspectos más específicos.
2. Inicialmente los alumnos prefieren este tipo de método frente a los tradicionales. Sin embargo, a medida que se pone en práctica se quejan de la cantidad de tiempo que les puede llegar a absorber. Por tanto, se hace necesario que el profesor calibre en cada caso el nivel de profundidad esperado de los trabajos.

Así, se puede concluir que, aunque este tipo de prácticas docentes pueden desenvolverse durante el curso de los estudios, suelen alcanzar su punto álgido respecto de los titulados que aspiren a perfeccionar su formación mediante la especialización en un campo determinado, probablemente, a través de alguno de los títulos propios o programas de formación continua que contempla la LOU [BOCG, 2001] en su artículo 34.3, o bien realizando los estudios de Tercer Ciclo encaminados al logro del Doctorado.

Visitas y prácticas en instalaciones y centros profesionales

En general sería conveniente llevar a los estudiantes a los centros donde desempeñarán sus labores profesionales. Es interesante que los estudiantes vean de cerca los centros donde desarrollan su trabajo los especialistas en la materia, a fin de conocer de cerca la realidad de las enseñanzas explicadas en la Universidad, y para que adquieran una visión directa y global del funcionamientos de esos centros. Otro beneficio a obtener de este tipo de contactos es el que el empresario conozca la formación de los alumnos con vistas a una posible contratación futura.

Sin embargo, la carga docente que tienen los alumnos hace muy difícil coordinar diferentes visitas a centros profesionales. Además, algunos estudiantes ven estas salidas como jornadas festivas de las que obtienen muy poco provecho.

Adicionalmente, los Planes de Estudios de las titulaciones de Ingeniería Técnica en Informática de Sistemas e Ingeniería Informática articulan otros medios para que los alumnos tomen el pulso a la actividad laboral de forma compaginada con sus estudios:

1. Se pueden conceder **6 créditos** de libre disposición por un mínimo de **180 horas de trabajo**, debidamente justificadas, en empresas u organismos públicos, en tareas de programador o equivalente.
2. Por otro lado los proyectos de fin de carrera (9 créditos obligatorios en el tercer curso del primer ciclo y 6 créditos troncales en el segundo curso de segundo ciclo) pueden desarrollarse en ocasiones en el marco de la colaboración **Universidad–Empresa**, por lo que también pueden servir a este fin.

3.5.2.5 Tutorías

El sistema de tutorías es un aspecto importante en el proceso de enseñanza, pues permite que el alumno pueda contar con la posibilidad de realizar consultas, discutir y esclarecer dificultades surgidas en las clases u otras actividades docentes, fuera del horario normal de éstas. El Real Decreto 898/1985 de 30 de abril, sobre régimen de profesorado universitario, fija en seis a la semana el número de horas de tutoría.

Representa esta técnica otra de las que tradicionalmente se han señalado para impartir la enseñanza universitaria, si bien su aplicación más fiel se reduce principalmente a la Universidad Anglosajona. El método consiste en una reunión periódica del estudiante (solo o en pequeños grupos), con el profesor tutor. En esta reunión, la discusión por medio del diálogo permite el intercambio de ideas, como base para la orientación y el desarrollo de la capacidad del alumno. Se diferencia del seminario, aparte de por el número de participantes, en que no existe excesiva rigidez en el tema y se concede al estudiante más iniciativa y responsabilidad. En definitiva, el alumno aprende por tres vías sucesivas: al hacer el trabajo, al observar los errores cometidos y defender los puntos que considera acertados y, finalmente, al repasar el trabajo completo, corregido y comprobándolo con su primera versión.

En la Universidad Española, dado el desequilibrio existente entre el número de alumnos y profesores, es razonable considerar el sistema como impracticable. Ahora bien, las posibilidades que presenta este método, como complemento de los anteriores, hace considerar como deseables la aplicación del mismo en formas menos estrictas, pero igualmente válidas y tendentes a la, hasta hoy utópica, “enseñanza individualizada”.

Las tutorías sirven para poner en contacto directo al profesor con los alumnos, fomentando la relación entre ambos. Este contacto mutuo debe ser utilizado por ambas partes. Los alumnos

para consultar al profesor todas las dudas surgidas, inquietudes, opiniones, perspectivas profesionales, buscar bibliografía específica... El profesor debe buscar en dicho contacto los elementos de autoevaluación que le permitan detectar el grado de entendimiento y las dificultades que encuentran los alumnos en la materia, detectando los conceptos captados de forma deficiente y que, por tanto, necesiten una revisión del planteamiento expuesto en las clases teóricas o prácticas. En definitiva, el profesor puede y debe observar en las tutorías la marcha general y particular de la asignatura.

La experiencia demuestra que, a pesar de que pueda ser beneficioso para la enseñanza de los alumnos, éstos son reacios a la hora de utilizar las tutorías, aunque las dudas existan. Desgraciadamente, las consultas se reducen a los días previos del examen, que es cuando el estudiante concentra mayores esfuerzos en la asignatura.

Es por ello que el profesor debe de fomentarlas, buscando la actitud participativa de los alumnos, concienciándoles de que el profesor está a su servicio. La preparación de trabajos y seminarios por parte de los alumnos constituyen la forma más eficaz de forzar al alumno a utilizar las tutorías. Si las fechas de entrega de estos trabajos se dosifican convenientemente a lo largo del curso, hasta el alumno más reacio, acaba por pasarse varias veces por el despacho para solventar las dudas que surgen en la elaboración de dichos trabajos. De esta forma se fomenta el trabajo diario de aprendizaje, frente a la extendida actitud de estudiar las materias por bloques en vísperas de los exámenes, que al fin y al cabo constituye la causa última de que las tutorías no se utilicen, o sólo se utilicen en dichas fechas.

Conviene mentalizar a los alumnos de que, para la utilización de este servicio, hay un horario establecido que deben conocer y respetar para que tanto profesores como alumnos puedan programarse sus quehaceres. Una iniciativa en este sentido, consiste en pactar con los alumnos cuáles van a ser las horas de tutoría. Éstas en ningún caso deberían de coincidir con sus horas de clase; y en el caso de que el profesor impartiera clase a grupos con diferentes horarios debería de hacer lo posible por distribuir sus horas de tutoría de forma que cualquier alumno pueda acceder a las mismas sin tener que ausentarse de clase.

No obstante, no ha de tomarse el horario de tutorías como limitación en la iniciativa de los alumnos, por lo que, siempre que sea posible, el profesor debe estar dispuesto para atender cualquier consulta.

3.5.2.6 Internet como vía de comunicación con los alumnos

En el IS'97[9] [Davis et al., 1997], se recomienda la utilización de Internet en la actividad docente. Esta recomendación en principio es muy vaga, pero se ve refrendada por la facilidad que actualmente tienen los alumnos para acceder a Internet, bien desde las propias Escuelas y Facultades, bien desde sus propios domicilios.

La aplicación de los servicios que ofrece a Internet a la docencia universitaria son enormes, destacando el correo electrónico y de una manera especial la WWW [Vetter y Severance, 1997; García et al., 1999a], que se convierte en una herramienta con la que es posible mejorar sustancialmente el proceso educativo [Seal y Przasnyski, 2001].

El correo electrónico ofrece un medio de comunicación asíncrona entre el alumno y el profesor [Nishida et al., 1996; Yu y Yu, 2002], que permite suprimir algunos de los inconvenientes de las tutorías presenciales, especialmente en lo referente a incompatibilidades de horarios, o distancia geográfica entre el alumno y la Universidad, o en la Universidad a Distancia [García et al., 1999a]. También permite el intercambio de ficheros entre las dos partes, *memorias e informes de prácticas* por parte de los alumnos, *material bibliográfico* por parte profesor, por ejemplo en los proyectos de fin de carrera. No obstante, como indica A. Huang [Huang, 2001], el auténtico potencial del correo electrónico como herramienta docente no ha sido todavía reconocido por la mayoría de la comunidad docente. Este autor propone en su artículo un proceso sistemático para la gestión del correo electrónico con los alumnos, que potencia en gran medida las aplicaciones básicas de esta herramienta en el campo docente.

La web ofrece muchas posibilidades: la mera exposición de los programas docentes, revistas en línea [Riser y Gotterbarn, 1998], acceso a almacenes de información (referencias bibliográficas, artículos, informes...) que juegan el rol de bibliotecas digitales [Marchionini y Maurer, 1995], soporte de la información relacionada con un curso, una asignatura o un libro (ya sea de forma pública para todo el mundo, o restringida a una Intranet) [Veraart y Wright, 1996; Paxton, 1996], preparación de cursos de forma conjunta, impartición de cursos de forma no presencial...

El material de carácter docente recopilado y publicado en páginas web es un activo de gran valor para la preparación de cursos o asignaturas [Mercuri, 1998].

Un interesante ejemplo de la utilización avanzada de la red para la docencia es, en el terreno de las bases de datos, el caso de Jeffrey Ullman y Jennifer Widom en la Universidad de

---

[9] El IS'97 es el currículo de ACM para Sistemas de Información (*Information Systems 97*).

Standford[10], en la que, como soporte a sus libros [Ullman y Widom, 1997; Ullman y Widom, 2001; García-Molina et al., 2001], se accede a resúmenes sobre manuales de productos utilizados en las prácticas por los alumnos de esta Universidad, transparencias utilizadas en las exposiciones de las clases y soluciones de exámenes.

La utilización de la potencia que ofrece el servicio web se está explotando en las asignaturas propias del perfil de este Proyecto Docente[11]. Donde, además del programa de cada asignatura, se proponen las prácticas (enlazándose sus soluciones), se facilita material complementario (en forma de artículos, manuales, informes técnicos, problemas resueltos...) y se establece un canal donde anunciar asuntos relacionados con la asignatura (noticias, cambios de fechas, defensas de prácticas...). Además, la organización de este material por cursos académicos ofrece un histórico muy interesante a los alumnos de las siguientes promociones, en forma de ejercicios resueltos y referencias de consulta.

En general, este sistema presenta las siguientes ventajas:

- Permite la confección de material adicional de la asignatura de una forma en la que el alumno está plenamente involucrado. Todo este material que se elabora, tiene el denominador común de no estar en principio al alcance del alumno. En concreto se trata de libros de problemas solucionados, tutoriales sobre herramientas y entornos, artículos o informes sobre temas específicos...
- Desde el punto de vista de los alumnos que hacen los trabajos se favorece la relación profesor-alumno, potenciando el uso de las tutorías más allá del día antes del examen.
- En el caso de los trabajos de alumnos corregidos por el profesor, aquellos alumnos que posteriormente accedan a los mismos, tendrán la opción de, por un lado, ver cómo se resuelven correctamente los ejercicios, y por otro, observar los fallos más comunes en los que se puede incurrir al intentar solucionarlos, obteniendo una experiencia positiva de los fallos de los compañeros.

La desventaja fundamental es la cantidad de tiempo de tutela y corrección de trabajos a llevar a cabo por el profesor, por lo que sólo se puede aplicar a grupos no numerosos (optativas y grupos de voluntarios en las obligatorias y troncales).

[10] http://www-db.stanford.edu/~ullman/fcdb.html

[11] http://tejo.usal.es/~fgarcia/docencia.html

En cierto modo, este método se asemeja a los seminarios en cuanto que también son desarrollados por los alumnos, pero se diferencian fundamentalmente en que la experiencia de los alumnos queda *congelada* en la red a disposición de sus compañeros actuales y de las promociones venideras. Además, no es tan importante la corrección del resultado final del trabajo, como el obtener los fallos más comunes de los alumnos. Este último dato no sólo es interesante para el alumno, sino que permite a la larga dar una idea precisa al profesor de qué puntos no acostumbran a quedar claros, y dónde debe de hacer por tanto mayor hincapié.

Otra posibilidad que ofrece la red es dejar en ella manuales, tutoriales y documentación diversa elaborada por los fabricantes, sobre los productos que se utilizan en las prácticas de las asignaturas. En este sentido conviene asegurar que aquellos elementos que no sean de libre disposición sólo sean accesibles dentro de una Intranet, al no ser legal exponerlos al público en general. Sin embargo, si es legal y necesario que sean accedidos por los alumnos de la titulación.

Como nada es perfecto, el abuso o la delegación excesiva de Internet como base a la docencia también tiene sus peligros en forma de falta de seguridad, deshumanización de la docencia, problemas de derechos de autor... [Neumann, 1998; Mercuri, 1998].

La proliferación del uso de Internet como recurso y herramienta docente ha dado lugar a los denominados espacios virtuales educativos, en el siguiente subapartado se va a presentar este concepto por la importante influencia que ejerce sobre las técnicas docentes, pero también se comentarán algunos de los riesgos que conllevan para la propia comunidad docente.

Espacios virtuales educativos

El auge de Internet en general, y de los servicios web [Berners-Lee et al., 1994] en particular, ha traído consigo la aparición de diversos espacios de trabajo virtuales como metáforas de interacción entre personas pertenecientes a distintos colectivos profesionales que, aun estando diseminados geográficamente, tienen un lugar común de intercambio de información e intereses en dichos espacios virtuales, siendo canalizada la entrada a éstos mediante los tan popularizados portales de Internet.

La enseñanza no es, y no debe ser, ajena a las posibilidades que abren las denominadas Nuevas Tecnologías de la Información y Comunicación (NTIC). El contexto en el que se desarrolla el proceso formativo a comienzos del siglo XXI es muy diferente al que se tenía diez años atrás, siendo las NTIC las responsables y las protagonistas de los cambios sufridos.

Como factores clave de estos cambios cabe citarse [Hare, 2000]:

- El estatus de recurso activo de gran valor que toma la información.
- El aumento de la información en formato digital, así como de usuarios finales consumidores de dicha información.
- La mayor necesidad de la formación continua de los profesionales, que exige modelos de enseñanza y aprendizaje más flexibles.
- El asentamiento de una cultura de la globalización, que rompe barreras físicas y temporales.

Un espacio virtual educativo efectivo debe canalizar tres vertientes básicas de las NTIC. Primeramente, la capacidad de trabajo en red, que permita una localización geográficamente dispersa de los participantes y un sincronismo o un asincronismo de las actividades formativas, según sea necesario. En segundo lugar, facilidades multimedia e hipermedia que permitan contar con información digital heterogénea y representada en diferentes formatos (texto, gráficos, sonido...) pudiéndola relacionar fácilmente. Y por último, una interacción persona-ordenador sencilla, intuitiva, pedagógica y completa, soportada por interfaces de usuario donde se potencie la noción de usabilidad de las mismas, en lugar de buscar la espectacularidad y el encantamiento momentáneo del usuario [García y García, 2002].

Los espacios virtuales educativos construidos en torno a los servicios web no se han limitado exclusivamente a las disciplinas más técnicas o cercanas al mundo de los ordenadores, sino que han calado en todas las disciplinas y a todos los niveles, sirvan como ejemplos [Brown y Neilson, 1996; Chrisman y Harvey, 1998; Lewis, 1998; Veldenz y Dennis, 1998; Paulisse y Polik, 1999; Chalk, 2000; Rosbottom et al., 2000].

La entrada en escena de los espacios educativos virtuales aporta grandes ventajas que complementan positivamente el proceso educativo tradicional. Sin embargo, el trabajo con las NTIC implica cambios en muchas áreas, y también algunos riesgos.

Entre los beneficios cabe citar [García et al., 2002]:

- *Descentralización del proceso educativo*: El seguimiento de las clases desde lugares geográficamente distantes, ya sea de forma síncrona o asíncrona, potencia la enseñanza a distancia, las tutorías no presenciales y los seminarios virtuales.
- *Aumento de la calidad y la accesibilidad de los materiales didácticos*: Los multimedia cambian el concepto del libro tradicional, no buscando suplantarlo sino completarlo. El texto y las fotografías, se ven completadas ahora con datos en

cualquier tipo de formato imaginable (vídeo, animación, sonido...), y localizados en cualquier lugar del mundo.

- *Procesos de aprendizajes personalizados*: Al tener acceso a materiales didácticos de calidad dentro de un proceso educativo asíncrono, el alumno puede progresar según su capacidad y tiempo.
- *Facilidad para el trabajo colaborativo*: El trabajo en equipo se ve facilitado desde el mismo instante en que mejoran y se potencian los medios de comunicación disponibles entre los participantes en una actividad docente.
- *Acceso universal a los recursos*: Permitiendo acceso a recursos localizados de prácticamente cualquier parte del mundo.
- *Capacidad de adaptación o configuración del entorno de trabajo*: El contexto de trabajo del usuario se adapta dependiendo de su nivel, su idioma u otras características, permitiendo un rendimiento más óptimo del interesado.

No obstante, las NTIC presentan también restricciones y riesgos más que desventajas o inconvenientes, normalmente derivados de su mala utilización, de su comprensión equivocada o de las complejidades inherentes al empleo de las NTIC como instrumento de autor. Entre los riesgos cabe citar [García et al., 2002]:

- *Desplazamiento de profesores y/o alumnos del proceso educativo*: Las NTIC obligan tanto a los docentes como a los discentes a un replanteamiento de la naturaleza de la enseñanza y el aprendizaje. Los profesores, en su papel de transmisores de conocimiento, deben aprender a manejar las herramientas que permiten canalizar su conocimiento y experiencia en materiales didácticos asimilables por los alumnos. Los alumnos deben estar abiertos a la utilización de las NTIC y a que las referencias de consulta no se limitan a las notas de clase.
- *Dispersión de la información*: Internet es una fuente inagotable de información, que fácilmente puede llegar a desbordar a cualquiera, y donde la calidad de los materiales accesibles es sumamente variable. Por este motivo, la mera existencia de información no basta para configurar un espacio virtual educativo [Moreno et al., 2000], debe ser información contrastada, clasificada y accesible a través de los medios oportunos.
- *Falta de calidad en los componentes educativos software*: Los servicios educativos ofertados en un espacio virtual no siempre tienen la calidad suficiente para su uso efectivo. La calidad debe mirarse desde las perspectivas técnica y pedagógica, de

forma que los componentes cumplan los requisitos para los que fueron diseñados, pero además lo hagan ayudando al proceso educativo. En este aspecto los mecanismos de interacción ofrecidos por las interfaces de los componentes software son de vital importancia. Una interfaz bien diseñada debe guiar al usuario (ya sea alumno o profesor) en la actividad docente en que se halle inmerso, mientras que una interfaz incorrectamente pensada perderá al usuario en el software, y lo más grave, en el caso de los alumnos, puede inducirles a errores de concepto graves que invaliden su proceso formativo.

Dentro del campo de los espacios virtuales educativos, el autor de este Proyecto Docente e Investigador lleva colaborando desde el año 1998 con el Laboratorio de Diseños Educativos Multimedia y Teleeducación del Instituto de Ciencias de la Educación (IUCE) de la Universidad de Salamanca, en la definición, diseño y construcción de componentes software para espacios virtuales educativos.

De esta experiencia se pueden obtener unas primeras conclusiones sobre los espacios virtuales educativos:

1. La elaboración de material docente presenta varios condicionantes, donde la inteligencia social para la constitución de equipos interdisciplinares es un aspecto con al menos tanto relevancia como estar al tanto de las últimas tendencias tecnológicas. Al menos los ingenieros de software deben colaborar activamente como apoyo a los profesionales de la educación que usan el ordenador como herramienta pedagógica [Holmes, 1999].
2. La creación de contenidos o recursos formativos, mediante el uso de las NTIC, no es exclusiva de la formación a distancia, sino que deben aprovecharse tanto en procesos educativos a distancia o no presenciales como refuerzo en los procesos educativos tradicionales o presenciales. El contexto que ofrece Internet y en especial la web es idóneo para explotar este potencial [Rosbottom, 2001].
3. Una de las mayores preocupaciones es la barrera que las NTIC imponen a multitud de docentes, que se ven en el ostracismo tecnológico y quedan relegados en los procesos formativos actuales. Nuestra intención es recuperarlos como autores de recursos educativos para la teleformación, creando herramientas a su medida, no a la medida de los expertos en Informática y Comunicaciones, con interfaces facilitadoras e integradoras, pedagógicamente elaboradas.

La relegación de docentes frente al importante avance de las NTIC en la Universidad, es un problema del que, en mayor o menor medida, se ven libres las Áreas de Conocimiento

mayormente relacionadas con la Informática (como puede ser el Área de Ciencia de la Computación e Inteligencia Artificial), pero que sufren muchas otras Áreas de Conocimiento en la Universidad Española.

Para hacer frente a este problema, y con el apoyo económico de la Junta de Castilla y León y la Unión Europea a través del Fondo Social Europeo, mediante el Proyecto de Investigación SA002/01 [García, 2002], se ha desarrollado una herramienta de autor para la creación de recursos formativos o libros electrónicos hipermedia, denominada "Facilitador para la composición de documentos electrónicos" [Gil et al., 2001; García y García 2002].

Esta herramienta encapsula toda la complejidad de manejo de las herramientas actuales en las facilidades que él autor necesita, y ofrece como resultado un producto didáctico, hipermedia y multimedia, que se pueda distribuir en diferentes formatos que faciliten al usuario el acceso a la información que contiene (CD-ROM, páginas web, texto impreso...). La base de trabajo es un documento con estructura jerárquica, típicamente accesible a través de un índice, donde cada unidad temática o lexia puede contener texto, material multimedia y enlaces a otras unidades o documentos (ver Figura 3.3).

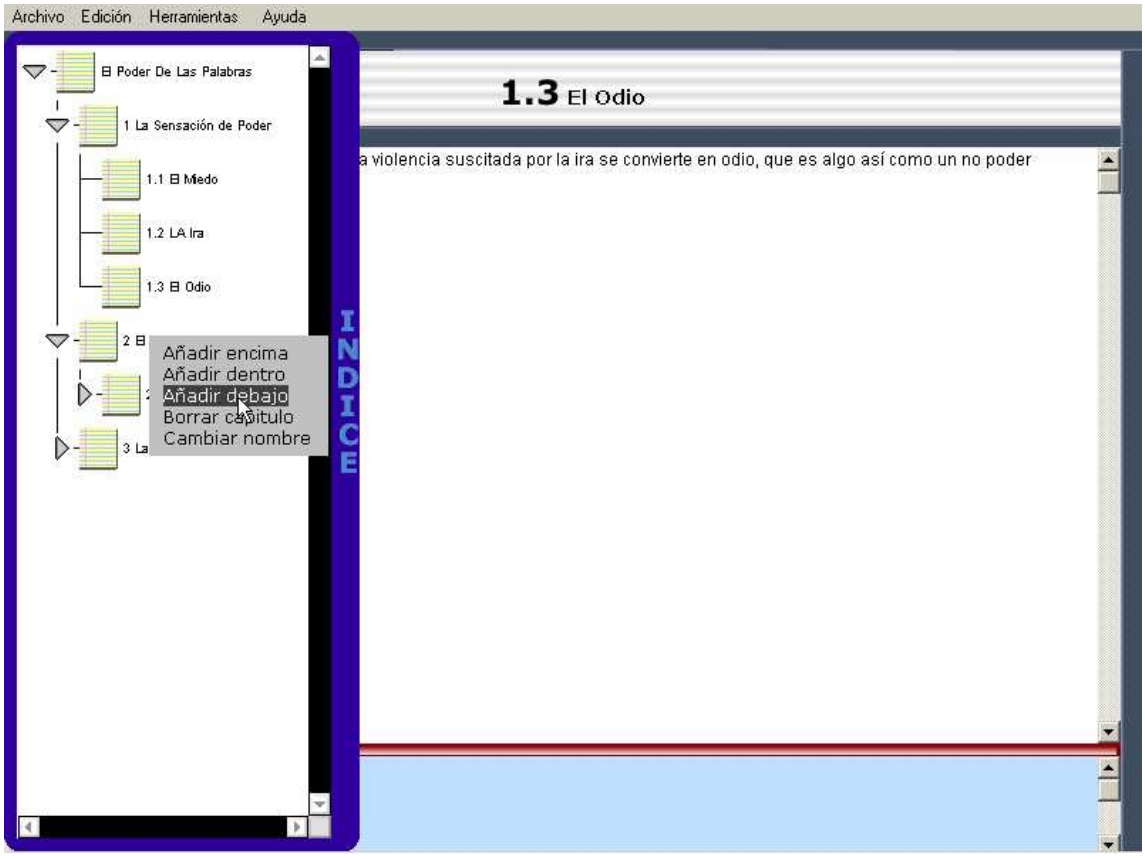


Figura 3.3. Interfaz del Facilitador para la composición de documentos electrónicos

3.5.2.7 Resumen de las actividades docentes comentadas

A modo de resumen en la Tabla 3.4 se muestran las ventajas e inconvenientes de cada una de las técnicas docentes anteriormente explicadas.

Aspecto	1[12]	2	3.1	3.2	4.1	4.2	4.3	4.4
Exposición coherente y completa de la materia	V	I	I	I	I	I	I	I
Participación del alumno	I	V	V	V	V	V		
Estímulo para el alumno	I	V	V	V	V	V		
Hábito de manejar bibliografía	I	V		V	V			V
Desarrollo de la capacidad de síntesis y crítica	I	V			V	V		V
Tratamiento personalizado	I	I	V	V	V		V	
Control del proceso de aprendizaje	I	V	V	V			V	
Dotación de recursos humanos y económicos	V	V	I	I	I	I		I
Aplicación de los conceptos teóricos	I	V	V	V				
Detectar dificultades de comprensión y aplicación de los conceptos teóricos	I	V	V	V			V	
Interés del alumno	I	V	V	V	V	V		V
Organización sistemática de los conocimientos				V	V			
Aprender a estructurar un trabajo				V	V			
Fomentar la capacidad crítica del alumno					V			
Crear el hábito de investigación científica					V			
Actividad profesional y laboral						V		
Confección de material adicional					V			V
Favorecer la relación alumno-profesor			V	V	V		V	V

Tabla 3.4. Resumen de las ventajas e inconvenientes de las actividades docentes

3.6 Sistemas de evaluación

La evaluación es un proceso inseparable del ciclo enseñanza – aprendizaje, que tiene como fin determinar en qué medida se han logrado los objetivos educativos establecidos sobre la base de las tareas profesionales concretas que el estudiante debe ser capaz de realizar al finalizar su aprendizaje. Constituye un aspecto tan esencial como complejo y delicado en el proceso

[12] 1- Clases teóricas, 2- Clases de problemas, 3.1- Prácticas libres, 3.2- Prácticas guiadas, 4.1- Seminarios y conferencias, 4.2- Visitas y prácticas en instalaciones, 4.3- Tutorias, 4.4- Internet

docente. Además, constituye una necesidad legalmente impuesta[13], así como por la propia metodología didáctica, en tanto que elemento clave en el proceso de enseñanza.

La evaluación no se reduce a una medida de la asimilación de la materia a lo largo del curso, sino que debe aportar los datos necesarios para valorar los programas, los métodos didácticos... Así, con respecto al profesor, la evaluación actúa como una realimentación directa destinada a mejorar el proceso educativo.

A través del proceso de evaluación se pretende alcanzar los siguientes objetivos:

- Poder llegar a tener un juicio de valor sobre la realidad del rendimiento del alumno, así como de sus posibilidades reales. Esta información es sumamente útil de cara a medir el grado de aprovechamiento del alumno en el marco de una enseñanza personalizada.
- Puesta de manifiesto de la coherencia o no entre los objetivos y los resultados. Si en este análisis la distancia entre lo deseable y lo real es aceptable, no habrá que introducir ninguna corrección, si, en cambio, las desviaciones no son las deseadas, habrá que tomar las decisiones pertinentes para corregir las deficiencias en los elementos que las hayan provocado.
- Recoger información para la toma de decisiones, no sólo con respecto al funcionamiento del sistema, sino, especialmente, en relación con los elementos que lo componen.

Según A. de la Orden [Orden, 1985], las cuestiones básicas que afectan a la evaluación educativa se centran en los siguientes puntos:

1. Determinación de qué se ha de evaluar y de los procedimientos y formas de evaluación.
2. Establecimiento y formulación de los criterios de evaluación.

[13] El artículo 46.3 de la LOU [BOCG, 2001] prevé que "*Las Universidades establecerán los procedimientos de verificación de los conocimientos de los estudiantes. En las Universidades públicas, el Consejo Social, previo informe del Consejo de Coordinación Universitaria, aprobará las normas que regulen el progreso y la permanencia en la Universidad de los estudiantes, de acuerdo con las características de los respectivos estudios*". En la medida en que no contradigan tales provisiones, la Universidad de Salamanca cuenta con un *Reglamento de exámenes y otros sistemas de evaluación*, aprobado por Junta de Gobierno en fechas 23 y 24 de mayo de 1989, 25 y 26 de octubre del mismo año, modificado parcialmente por Acuerdo de la Comisión Delegada de la Junta de Gobierno de desarrollo reglamentario de 16 de noviembre de 1989 y por Acuerdo de la Junta de Gobierno de 26 de febrero de 1993, donde exhaustivamente se detallan los pormenores del procedimiento y revisión de los mismos.

3. Determinación y clarificación de las decisiones que han de ser adoptadas como resultado de la evaluación.

Para que un sistema de evaluación sea válido, debe reunir las siguientes condiciones:

- Que el alumno conozca con precisión los objetivos formulados.
- Que las pruebas exijan una adecuada y representativa muestra de los contenidos y conductas especificadas en los objetivos.
- Que la evaluación sea fiable y objetiva, de tal forma que el azar o los errores instrumentales tengan un efecto mínimo en los resultados.

La evaluación debe realizarse con un rigor especial, esto es, debe ser lo más objetiva posible, tratando por igual a todos los estudiantes. Además, debe estimarse más la capacidad de aplicación de los conocimientos, que su mera memorización. Así, debe procurar evitarse que las cuestiones planteadas en las pruebas tengan un carácter de *recuerdo aislado* o memorización mecánica, a la vez que deben incluirse aquéllas que sirvan para detectar la capacidad de aplicación, síntesis y generalización de conceptos; evitando así lo que Fernando Lara denomina "*un aprendizaje puramente memorístico que se perderá en cuanto el alumno haga el examen, o tras la caña de después*" [Lara, 1997].

En general, dentro de un proceso de evaluación se admite la necesidad de pruebas iniciales y finales.

Las **pruebas iniciales** tienen carácter de evaluación formativa, no de certificación. Entre ellas se distinguen dos tipos de pruebas diferentes en cuanto a su finalidad, pero que pueden realizarse de forma simultánea. Éstas son:

- ***Prueba del nivel requerido***: Antes de comenzar una enseñanza dada, debe asegurarse que los estudiantes estén en un cierto nivel: *nivel requerido*. Corresponde al profesor definir cuáles son los conocimientos que juzga indispensables para que los estudiantes, que le son confiados, puedan beneficiarse al máximo de la enseñanza que se les dará. Esta prueba permite conocer si todos los estudiantes están al mismo nivel y si éste coincide con el requerido, para que en su defecto se efectúen las modificaciones necesarias de la enseñanza prevista para colocarles a nivel. Si esto no se hace, la calidad disminuye necesariamente. Esta cobertura puede realizarse de diferentes modos: referencias bibliográficas, clases suplementarias a los estudiantes afectados y sólo en último caso, clases previas para todos.

- ***Prueba de nivel de partida***: Su finalidad es conocer, al comienzo de la enseñanza, el nivel al que están los estudiantes respecto al contenido en sí de la asignatura. Solamente conociendo el nivel de punto de partida es posible calcular la ganancia real al final de la enseñanza. Por otra parte, si se detectara que algunos estudiantes estuvieran bastante avanzados respecto a los objetivos previstos, convendría considerar una orientación suplementaria para los mismos; situación que por otra parte no es frecuente.

Las **pruebas finales** siempre tienen carácter de certificación. Si un estudiante juzgado competente obtiene una mala calificación en el examen final, debe reevaluarse la situación y no dejar al examen final una exclusiva misión sancionadora.

La evaluación debe ser un proceso continuo y como tal no puede limitarse a una prueba aislada, con sentido de sanción arbitraria sin tener en cuenta más factores. La denominada *evaluación continua* se convierte así en el método de evaluación más adecuado, ya que tendría en cuenta todos los aspectos de la labor del estudiante (participación en seminarios, trabajos realizados, nivel de participación, resultados en exámenes parciales y/o finales...).

Una evaluación continuada supone la mejor forma de conocer la evolución de los conocimientos del estudiante, así como de constatar el grado de asimilación alcanzado. En la mayoría de los casos se demuestra que este sistema es inviable dentro de la estructura universitaria, debido principalmente a dos aspectos. Por un lado, el gran número de alumnos obliga al profesor a generar, corregir y supervisar una gran cantidad de actividades docentes. Por otro, el alumno, al menos inicialmente, se opone a realizar, preparar y asistir a esas actividades, debido generalmente, a que suponen un gran esfuerzo por su parte.

3.6.1 Técnicas de evaluación

Si se considera que el desarrollo de una tarea profesional comporta una serie de habilidades, que al dirigirse a otra persona implican una cierta capacidad de comunicación, y que para su cumplimiento es necesaria determinada capacidad intelectual y un cierto nivel de conocimientos. Cualquier metodología para la evaluación de estudiantes debe disponer de técnicas destinadas a la evaluación de estos tres campos:

1. Conocimientos o proceso intelectual.
2. Capacidad de comunicación.
3. Habilidades prácticas.

Evidentemente el tipo de instrumentos que proporcionan los datos para emitir juicios de valor sobre cada uno de estos campos es distinto. Ninguna prueba por muy fiable y objetiva que sea puede utilizarse de forma exclusiva. El uso de una mayor variedad de técnicas asegura una evaluación más completa, mientras que el limitarse a una de las pruebas tradicionales orales o escritas, sólo permite determinar la capacidad de memorización o nivel de conocimientos teóricos, pero no la aptitud para la utilización profesional de esos conocimientos.

En la Tabla 3.5 se recogen las técnicas que preferentemente se emplean para la evaluación de los estudiantes en los tres campos referidos.

Seguidamente se procede a la exposición de las características de cada una de ellas, así como a la valoración de sus ventajas e inconvenientes, indicando algunas pautas para su correcta formulación.

	Métodos directos	**Métodos indirectos**

Campo de los conocimientos	No existen	• Pruebas escritas
		  o Preguntas de Respuesta Libre (Redacción)
		  o Preguntas de Respuesta Abierta Corta (PRAC)
		  o Preguntas de Elección Múltiple (PEM)
		  o Situación Problemática
		• Pruebas orales
		  o Exposición
		  o Debate
		  o Entrevista
		• Informes de Observación

Campo de la comunicación y de las habilidades profesionales	Observación de una prueba práctica • Situación real • Entrevista	Proyecto

Tabla 3.5. Técnicas de evaluación más utilizadas

3.6.1.1 Evaluación del campo de los conocimientos

Como puede deducirse de la tabla anterior, el examen, independientemente de la forma que adquiera, se convierte en la forma más utilizada para evaluar los conocimientos del alumno en relación con una determinada materia.

El examen debe poseer la mayor capacidad evaluadora posible, además de servir de realimentación eficaz entre docentes y discentes, para así valorar correctamente la asimilación de las materias tratadas [Crawford y Fekete, 1997]. El examen debe ser planteado de forma que disminuya la incertidumbre derivadas de la prueba en sí, siendo deseable su compleción con

respecto al temario, su eficacia en relación con los objetivos planteados y que el alumno se encuentre informado sobre los conceptos que se le exigen y la forma cómo se va a comprobar este conocimiento [Mora et al., 1995].

En general, se pueden citar una serie de criterios a tener en consideración, en cuanto a exámenes se refiere [Gullbert, 1989]:

- **Validez:** grado de precisión con que el examen utilizado mide verdaderamente aquello para lo que fue diseñado como instrumento de medida.
- **Fiabilidad:** grado de confianza que se puede adscribir a los resultados de un examen. Se trata de la constancia con la que un examen aporta los resultados esperados.
- **Equilibrio:** grado de concordancia entre la proporción de preguntas reservadas a cada uno de los objetivos y su proporción ideal del examen.
- **Equidad:** grado de concordancia entre las preguntas propuestas en el examen y el contenido de las enseñanzas.
- **Discriminación:** cualidad de cada elemento de un método de evaluación, que permite distinguir, con respecto a una variable dada, a los alumnos más preparados de los menos preparados.
- **Tiempo:** es bien conocido que un instrumento de medida será menos fiable si, a causa de disponer de poco tiempo, permite introducir factores no pertinentes como el azar. De todas formas, depende del objetivo que se persiga, a veces el fijar el tiempo estrictamente es un factor importante.

En pro de garantizar la objetividad de los exámenes, parece esencial cuidar al máximo su corrección, la cual debe de ser repasada convenientemente. Asimismo, deben ponerse los medios oportunos para que las revisiones de los exámenes sean conocidas y accesibles por todos los alumnos, haciendo de las mismas una extensión de las tutorías. Este punto permite que el alumno tome conciencia de su responsabilidad directa en la nota obtenida, además de servirle como una experiencia de aprendizaje de sus propios errores.

Pruebas escritas

Aunque la utilización de los exámenes escritos es notablemente superior a la de los orales, no cabe atribuirlo más que a una razón de índole práctica: la facilidad que supone el adaptar el ritmo de evaluación a las posibilidades reales del profesor o incluso, en ciertos tipos de pruebas, la corrección automatizada.

Exámenes de respuesta libre

Este tipo de pruebas se corresponde con los clásicos exámenes donde el alumno redacta su propia respuesta a la pregunta propuesta. Su principal característica es que el alumno debe organizar sus ideas, a la vez que demuestra su capacidad de expresión, seleccionando lo más importante.

En su enunciado debe delimitarse claramente el problema y en ocasiones convendrá precisar la estructura de la respuesta.

La faceta más positiva de este tipo de exámenes es que posibilita apreciar la capacidad para emitir juicios críticos. En su contra tiene dos inconvenientes principales. El primero se refiere al gran tiempo que debe emplearse para su corrección, unido a la dificultad para hacerlo con objetividad, evaluando a todos los alumnos con el mismo criterio, pues intervienen factores externos como el cansancio, el estado de ánimo, la predisposición inicial... Para evitar o al menos reducir este efecto se aconseja:

a) Leer completamente todos los exámenes para tomar una referencia antes de pasar a la corrección propiamente dicha.

b) No realizar la corrección de un examen completo tras otro. Es más adecuado ir corrigiendo pregunta por pregunta todos los exámenes. Así, los mencionados factores externos tienen mayores probabilidades de manifestarse de igual manera para todos los alumnos[14].

El segundo inconveniente es que limita en demasía el número de parcelas cognitivas a juzgar, no siendo, en muchos casos, la selección de preguntas suficientemente representativa. Esto se debe controlar con una adecuada redacción de las preguntas, intentando que tengan una cierta originalidad e interrelacionando diversos conceptos.

Por todo ello, sólo es conveniente utilizar estas pruebas para evaluar un tipo de actuación que otros métodos no puedan medir, como la síntesis de nociones complejas, comparar dos fenómenos, analizar causas, establecer relaciones...

Exámenes de respuesta abierta y corta

Se trata de una serie de preguntas redactadas de tal modo que exigen por respuesta un concepto predeterminado y preciso.

[14] En todo caso conviene revisar de forma global, en un último paso, aquellos exámenes cuya calificación final sea dudosa por estar en la frontera entre una nota y otra.

Es una modificación de la anterior que evita alguno de sus inconvenientes. Por un lado, permite evaluar un área de conocimiento mayor, ya que se pueden plantear un mayor número de cuestiones que contemplen más aspectos del temario y, normalmente, al ser aceptable una sola respuesta, se gana objetividad en la corrección. Existen estudios que indican que la frecuencia de discordancias debidas únicamente a la contradicción entre dos correctores no alcanza el 2%.

Exámenes tipo test

También conocidas como pruebas PEM (Preguntas de Elección Múltiple). Son las denominadas pruebas objetivas[15], que consisten en un cuestionario formado por una serie de preguntas con varias respuestas propuestas, entre las cuales el alumno debe elegir una o varias de ellas.

Entre las principales ventajas de este tipo de pruebas se puede citar las siguientes:

- Evalúan mayor volumen de conocimientos que cualquier otra prueba (siendo factible cubrir la totalidad del temario).
- Permiten graduar la dificultad de la prueba.
- Posibilitan medir procesos intelectuales distintos a la simple memorización.
- Rapidez de corrección, pudiendo realizarse esta de forma automatizada [Mason y Woit, 1998].
- Objetividad.

Sin embargo, tampoco es un método perfecto, presentando los siguientes inconvenientes:

- Su preparación exige mucho tiempo y competencia si se quieren evitar las preguntas arbitrarias, ambiguas o que sólo evalúen el recuerdo.
- Las condiciones teóricas que presentan en sus planteamientos no se corresponden casi nunca con situaciones reales de una forma exacta.
- Resultan costosas si el número de alumnos sobre el que se van a aplicar es reducido.

[15] Bajo este epígrafe pueden incluirse también otro tipo de pruebas de uso menos frecuente por sus mayores limitaciones, entre las que se encuentran las de proposición incompleta, verdadero-falso, ordenación, localización o asociación.

Situación problemática

Este tipo de pruebas permiten evaluar el conocimiento de los procedimientos, técnicas y herramientas aplicables a un supuesto práctico de características predeterminadas por el docente y semidesarrollado por él mismo.

Las preguntas formuladas sobre el caso deben ser tales que permitan la elaboración de las respuestas en un espacio razonable de tiempo. Presenta ventajas sobre otras pruebas escritas pues, aunque teniendo la finalidad de valorar procesos intelectuales, se acerca más al campo de las habilidades prácticas, pero su calificación es también laboriosa y difícil de objetivar.

Pruebas orales

La prueba oral permite al profesor analizar de forma más detallada los conocimientos reales del alumno y su grado de comprensión, junto con su capacidad de estructuración y ordenación de las respuestas, frente a las cuestiones o problemas planteados.

El hecho de que las pruebas orales hayan decaído en los últimos años puede encontrarse quizás en la masificación de la enseñanza que dificulta el contacto personal y directo. Sin embargo, fuera del ámbito docente, el peso que ha cobrado la técnica de la entrevista en la selección de personal por parte de empresas, apunta hacia una nueva valoración del procedimiento.

Además, como indican algunos estudios, los mismos estudiantes que han dado respuestas aceptables a ciertas preguntas en exámenes escritos, demuestran tener grandes *lagunas conceptuales* al tener que defender los mismos conceptos en una entrevista o en una prueba oral [Greening, 1997], derivándose de esto que muchos alumnos basan su estudio en la elaboración de estrategias para pasar con éxito los exámenes, sin preocuparse de realmente comprender las materias objeto de estudio.

Los mayores inconvenientes que presenta cualquier prueba oral pueden resumirse en los siguientes:

- *Estandarización inadecuada*. La prueba a que se somete cada estudiante es diferente, lo que además de romper con la equidad, introduce un factor subjetivo importante en la evaluación.
- *Influencia excesiva de factores sobreañadidos*. Como es la reacción emotiva del alumno y subjetividad del examinador; cada estudiante posee unas características personales que son ajenas a sus conocimientos, como los nervios o su carácter, que

pueden perjudicar de forma notable a unos más que a otros, afectando especialmente a su capacidad de reflexión o análisis.

- *No permiten cubrir un contenido extenso de la materia*.
- *Alto coste en tiempo*. En relación con el valor limitado de las informaciones que aporta.

Pueden diferenciarse tres tipos de pruebas orales:

- **Exposición autónoma de un tema**. Consiste en el desarrollo de un tema sin que exista interacción entre el expositor y evaluador. Presenta el inconveniente de no poderse profundizar en el conocimiento de determinados conceptos cuya ignorancia se soslaya haciendo una alusión superficial. Sin embargo, permite valorar otra serie de aptitudes como capacidad de estructuración, exhaustividad en el tratamiento o reacción ante un auditorio.
- **Debate**. Su utilización es más frecuente como técnica de enseñanza que como medio de evaluación, por los numerosos sesgos de que puede ser objeto. No obstante, permite valorar no sólo la calidad de la información presentada sino también la oportunidad de la misma y capacidad crítica.
- **Entrevista.** Básicamente existen dos modalidades de entrevistas:
    - *Estructurada*, también denominada interrogatorio, en la que el alumno ante una pregunta debe proporcionar una respuesta concreta.
    - *Semiestructurada*, cuyo desarrollo es más coherente y distendido. Las preguntas se relacionan con las respuestas anteriores, ganándose fluidez y versatilidad, siendo factible profundizar en temas concretos o pedir justificaciones, así como pasar de los puntos fuertes a los débiles del candidato.

Informes de observación

Este tipo de pruebas implica la observación directa, repetida y estandarizada de la actividad de los alumnos.

Aunque su ventaja teórica sería el permitir una evaluación global, la carencia en la actualidad de medidas objetivas probadas parece aconsejar como razonable el no utilizar este método de simple observación más que con valor de matiz, por el importante componente de subjetividad que conlleva.

3.6.1.2 Evaluación de las habilidades profesionales y de la capacidad de comunicación

Estos dos campos son tradicionalmente olvidados al plantear la evaluación de los estudiantes a pesar de reconocerse que son los que realmente valoran o prueban la capacidad profesional del alumno en función de su competencia para organizar y utilizar datos y técnicas en situaciones reales.

Se desarrollan conjuntamente por disponerse de los mismos instrumentos de medida para su valoración. Instrumentos que, a pesar de su difícil normalización y objetividad, es imprescindible utilizar en cualquiera de sus posibilidades, si se pretende una evaluación completa de los estudiantes.

Prueba real o simulada

Es un tipo de prueba en la que el estudiante tiene que realizar tareas profesionales, en un medio y condiciones iguales o próximas a aquéllas en las que tendrá que desenvolverse en su vida profesional. En los casos de imposibilidad de realización en situación real puede efectuarse en simulación, mediante, por ejemplo, grabación de vídeo u otro sistema directo de recogida de datos.

Se aconseja su utilización cuando la destreza y/o relación interpersonal son el componente principal del objetivo educativo. Sin embargo, presenta dificultades importantes entre las que están: la colaboración de Organizaciones Empresariales o Administración, y la presencia puntual del examinador en el momento en el que el estudiante hace la demostración de las aptitudes requeridas, pues pueden ser períodos prolongados de tiempo.

Realización de un proyecto

Es un tipo de prueba que permite la observación indirecta de una tarea profesional. En ella el estudiante debe realizar una actividad en un período de tiempo variable, pero extenso, que tendrá como resultado un producto que el docente deberá evaluar. El producto puede ser un documento escrito (informe, investigación) o una obra concreta.

Se aconseja utilizar esta técnica si el componente principal del objetivo educativo es una habilidad técnica o intelectual compleja y si se está interesado más en el producto que en el modo de actuar del estudiante.

El objeto del proyecto puede ser elección del alumno, aunque debe ser el profesor quien determine los requisitos mínimos o estructura del producto final. Puede realizarse individualmente o en grupos aunque no se recomienda más de cuatro o cinco personas por

grupo. A diferencia de la prueba práctica exige el desarrollo secuencial de todas las etapas que deben realizarse, desde la recogida inicial de información hasta el resultado final.

Las dificultades que presenta su realización son: alto coste en personal docente, por el tiempo que se precisa tanto para su tutela como para evaluar su resultado. A esto hay que añadir la necesidad de instaurar una relación de confianza frente al alumno para evitar el fraude [Carter, 1999].

3.6.1.3 Conclusiones sobre los sistemas de evaluación

A la vista de las ventajas e inconvenientes expuestos para las distintas técnicas de evaluación, y dado que no existe ninguna completa en sí misma, únicamente señalar que, con el fin de conseguir una valoración de la capacidad profesional lo más real posible, siempre ha de plantearse la realización de al menos una prueba de evaluación del campo de los conocimientos y otra de los campos de las habilidades profesionales y capacidad de comunicación.

La determinación del tipo de prueba concreta a realizar en cada campo debe considerar el contexto donde se encuentra para poder ser llevada a cabo.

3.6.2 Calificación

La calificación corresponde a la etapa de la evaluación que tiene por finalidad la interpretación de los resultados obtenidos en la aplicación de los instrumentos de medida diseñados. Cuando varios instrumentos diferentes dan un resultado concordante, a pesar de sus imperfecciones, la fiabilidad de la evaluación aumenta.

Antes de iniciar el proceso de calificación es preciso haber decidido si lo que se pretende de la evaluación es que compare individuos entre sí, o que informe sobre lo que los individuos son capaces o no de realizar.

Pruebas de criterios absolutos y relativos

Son conocidas también como pruebas referidas a criterios y pruebas normalizadas respectivamente.

Una prueba de criterios absolutos es aquélla que tiene por finalidad evaluar la actuación de un individuo en relación con un nivel de actuación establecido de antemano basándose en el dominio de un objetivo específico fijado.

Por el contrario las pruebas de criterios relativos tienen por fin el comparar un individuo con otros o con un grupo, sobre la base de la curva de distribución normal de los resultados obtenidos por los estudiantes que han realizado la misma prueba.

Las pruebas de criterios relativos pueden ser válidas para seleccionar un cierto número de individuos, por ejemplo para su admisión a seguir un programa dado. Pero si los objetivos perseguidos son de gran importancia (es decir su dominio o no, tiene consecuencias sobre la colectividad en el ejercicio profesional) es igualmente importante determinar si han sido realmente alcanzados. Esto no puede hacerse comparando entre sí a los estudiantes, sino comparando la actuación del estudiante con la actuación prevista en el objetivo, es decir, mediante pruebas de criterios absolutos.

El sentido final de las pruebas de criterios absolutos es establecer el mínimo aceptable para certificar si un discente llega al mínimo exigido socialmente para el ejercicio con garantías de su labor profesional. Citando a R. F. Mager: "*Poco importan los esfuerzos del estudiante, poco importa que casi haya llegado a la meta... mientras no sea capaz de hacer lo que está obligado a hacer, no se debe certificar que es capaz de hacerlo*" [Mager, 1976].

La evaluación continua debe hacer competir al estudiante consigo mismo y no con otros estudiantes. La lucha debe plantearla contra su ignorancia, pues sería tan injusto el no considerar apto a un estudiante con un nivel de actuación satisfactorio por estar incluido en un grupo de estudiantes particularmente brillante, como peligroso el certificar la aptitud de estudiantes no capacitados, aunque sus resultados sean superiores a la media del grupo al que pertenecen.

Entre las cualidades que debe poseer un sistema de calificación están: *la claridad, la pertinencia, la precisión* y *la objetividad*. Además, para minimizar los factores condicionantes de errores de calificación, conviene tener presentes las siguientes consideraciones o criterios generales:

- Calificación de las respuestas de modo anónimo.
- Si una prueba va a ser corregida por dos o más docentes, antes de la realización de la misma debe haberse acordado el procedimiento de calificación, así como los elementos que debe contener una respuesta completa, realizándose aisladamente la calificación.
- Utilizar un sistema de calificación cuantitativo, según los elementos que se supone deben aparecer en las respuestas. Además, si se considera que para asegurar un nivel de actuación aceptable la respuesta a determinadas preguntas tiene especial

relevancia, el peso atribuido a cada una de ellas puede ser diferente y debe especificarse y comunicarse a los alumnos en el momento de la prueba.

- Cuando la corrección no es automatizada, calificar las respuestas de todos los estudiantes a una misma pregunta, antes de pasar a la siguiente.
- No debe formarse un juicio del candidato a partir de una sola pregunta. Para cada estudiante, el cálculo de la calificación será acumulativo y fundamentado en la lectura de varias respuestas de alumnos diferentes, pues contribuye a una mayor seguridad.
- Cuando la evaluación global se efectúa basándose en pruebas diferentes, debe determinarse y comunicarse previamente a los alumnos el peso que cada una de ellas tendrá en la calificación global.

3.7 Fuentes de nuevos conocimientos: La investigación

La función de la Universidad no se limita a la transmisión de conocimientos para la formación de nuevos titulados, sino que debe preparar a los Doctores que constituirán el futuro personal docente e investigador de la Universidad y Centros de Investigación. Estas tareas no se pueden llevar a cabo adecuadamente si no existe una importante labor investigadora en la Universidad.

Quizás el hito por excelencia que, dentro de la comunidad universitaria, se asocia a la labor investigadora es la **Tesis Doctoral**, cuyo significado se encuentra regulado por el Real Decreto 185/1985, donde se exponen los objetivos del tercer ciclo, a saber:

- Disponer de un marco adecuado para la consecución y transmisión de los avances científicos.
- Formar a los nuevos investigadores y preparar equipos de investigación que puedan afrontar con éxito el reto que suponen las nuevas ciencias, técnicas y metodologías.
- Impulsar la formación del nuevo profesorado.
- Perfeccionar el desarrollo profesional, científico, técnico y artístico de los titulados superiores.

De los objetivos presentados se deduce una íntima relación entre docencia e investigación, que se hace más patente en el caso de la Informática en general, y de la Ingeniería del Software en particular. La obsolescencia de esta disciplina es realmente desconcertante. A diferencia de

campos más asentados, en esta área se puede ver como sus fundamentos son objeto de importantes cambios, de forma que conceptos que hasta hace apenas seis años eran temas de investigación, se incluyen hoy en día en los temarios de las asignaturas. Esto exige un esfuerzo de actualización constante que requiere de la investigación como fuente de nuevos conocimientos, para facilitar así la docencia cuando ésta coincidiera con el área de investigación desarrollada.

El objetivo de aunar docencia e investigación requiere de una planificación departamental y de una decidida ayuda desde la Universidad. El profesorado es la piedra angular, y son necesarias políticas continuadas de formación de este profesorado que no se acaban con la financiación, sino que requieren de una planificación que determine los objetivos de esta formación. Hay que añadir que la legislación hoy en vigor aumenta la autonomía de los Departamentos para confeccionar sus Planes de Estudios, lo que lleva indisolublemente pareja una mayor responsabilidad en las oportunidades de formación que se brindan a los alumnos. Esta responsabilidad pasa por planificar asignaturas que permitan abrir las líneas formativas que se desean reforzar. Esto sólo es posible si se cuenta con un profesorado formado, y aquí, la realización de una Tesis Doctoral es una magnifica oportunidad que no se debe desaprovechar [Díaz, 1999].

No obstante, la labor de investigación de un docente no acaba (o no debiera) con la culminación de su Tesis Doctoral, sino que, aprovechando la experiencia obtenida en la realización de este trabajo, debe canalizar sus esfuerzos de investigación tanto en la formación de nuevos doctores como en el trasvase de los frutos de su investigación a la sociedad. Es en este último punto donde debe fraguarse la conexión de la comunidad universitaria con el tejido industrial y empresarial donde se encuentre inmersa.

3.7.1 La conexión Universidad - Empresa

La Universidad y la Empresa difieren considerablemente sobre los objetivos e intereses de cada una en lo tocante a investigación. Sin embargo, la única forma de unificar esfuerzos es acercar posturas entre ambas entidades y abrir caminos para que pueda existir una transferencia entre la investigación realizada en la Universidad y las necesidades actuales de la Empresa [Glass, 1997].

Esta comunicación no sólo se pide en el campo de la transferencia en el contexto de la investigación. De hecho, cada vez son más las voces que abogan porque la formación curricular de los nuevos titulados, especialmente en carreras de carácter técnico como pueden ser, en

general, las Ingenierías en Informática, no dé la espalda a las necesidades de la Empresa que ha de acoger a éstos al final de sus estudios.

En este sentido, la Ingeniería del Software es una de las disciplinas sobre las que cae una mayor responsabilidad a la hora de preparar a los estudiantes para su incorporación a la vida laboral [Díaz-Herrera y Powell, 1998], [Villarreal y Butler, 1998], [Wohlin y Regnell, 1999].

A menudo se produce la paradoja de que los estudiantes de Ingeniería Informática han sido educados por profesores que probablemente nunca han trabajado como ingenieros informáticos en ningún puesto. Por otro lado, una parte sustancial de la investigación universitaria que se realiza en cualquier país del mundo desarrollado, la constituye la investigación contratada. Mediante ésta, la Empresa encuentra un eficaz apoyo para sus procesos de innovación y desarrollo, y la Universidad aumenta su nivel de calidad en la docencia al aumentar su calidad investigadora y estar en contacto con nuevos procesos o productos útiles. De esta manera, toda la sociedad sale beneficiada.

Por todo ello, la interconexión entre la Empresa y la Universidad ofrece los siguientes beneficios:

- Para la Empresa, la movilización del potencial intelectual y científico de sus técnicos rompiendo la rutina e incrementando la eficacia de su trabajo. Además, podrá influir en los programas de las titulaciones y tener acceso a las innovaciones en métodos, herramientas y tecnología resultado de la investigación conjunta con el mundo universitario [Beckman et al., 1997].
- Para la Universidad, la valoración de su trabajo y un contacto con la vida real mediante problemas concretos. Así mismo, un beneficio económico que posibilite la adquisición de nuevos aparatos, productos y becas para estudiantes, que permita a éstos conocer la problemática del desarrollo del software en un entorno de trabajo real [Dawson y Newsham, 1997; Kuhn, 1998]. También una orientación sobre qué necesita enseñar y cuáles son las necesidades de conocimientos de los futuros ingenieros informáticos.
- Para la sociedad en su conjunto, una mejora en el nivel de efectividad de su Universidad e Industria, que de esta manera pueden ofrecer un mejor servicio a la comunidad.
- En general, un aumento de la reputación y el prestigio de todos los involucrados [Mead et al., 2000].

Una manera de efectuar esta conexión podría ser el intercambio de técnicos altamente cualificados de la Industria y profesores de la Universidad. Los primeros podrían enseñar durante cierto tiempo en la Universidad y los profesores pasar temporadas en la Industria. Todo esto fomentaría un mayor conocimiento mutuo. De hecho, es práctica común en los países más desarrollados, existiendo algún avance en la Universidad Española como puede ser la figura del profesor asociado[16].

Sin embargo, en un área con un marcado carácter tecnológico, la interacción con el entorno industrial es fundamental por cuanto le ofrece una perspectiva directa de las necesidades que se han de resolver y, por tanto, beneficia de forma inmediata a la labor docente que ha de llevar a cabo un profesor universitario.

Algunas de las Universidades más famosas del mundo, tales como Harvard y el MIT, tienen acuerdos con compañías para desarrollar programas de investigación académica en áreas específicas donde se esperan descubrimientos de interés práctico. Otro ejemplo es el desarrollo de la biotecnología en Israel. Explotando la investigación en la Universidad, las compañías israelitas están logrando convertir su país en un líder mundial en dicha área. Ejemplos concretos de la colaboración entre la Universidad y la Empresa en asuntos relacionados con la Ingeniería del Software se pueden encontrar en [Beckman, 1999; Beckman et al., 1999].

No obstante, debe mantenerse el respeto mutuo entre la investigación industrial y universitaria. Cada una tiene su finalidad y su papel que desempeñar, siendo ambas complementarias.

En el caso concreto de la Universidad de Salamanca, la Asociación Universidad Empresa de Salamanca ha sido absorbida por la Fundación General de la Universidad, y cada vez se trabaja más para preparar al alumno según las necesidades de la sociedad.

3.7.2 Los proyectos de final de carrera

Según se contempla en los Planes de Estudios de las titulaciones Ingeniería Técnica de Informática de Sistemas (Tabla 2.17) e Ingeniería Informática (Tabla 2.20) de la Universidad de Salamanca, para la obtención del correspondiente título, los alumnos deben realizar un Proyecto Fin de Carrera. En concreto, para los alumnos de Ingeniería Técnica se trata de una asignatura obligatoria que tiene asignados 9 créditos prácticos, mientras que para los estudios de Ingeniería

[16] A tal efecto, el artículo 53 de la LOU [BOCG, 2001], como ya hiciera también la derogada LRU [MEC, 1983], prevé la figura del profesor asociado, contratado en cada caso "*entre especialistas de reconocida competencia que acrediten ejercer su actividad profesional fuera de la Universidad*".

Informática es una asignatura troncal de 6 créditos prácticos. La realización de estos proyectos puede considerarse a veces como investigación aplicada. Es por ello por lo que se ha considerado la inclusión en este Proyecto Docente de un apartado sobre este tema.

En particular, el Proyecto Fin de Carrera consistirá en un diseño, aplicación, explotación y gestión de sistemas informáticos o en un estudio sobre algún tema o tecnología avanzada que, por su novedad o escasa implantación, no haya sido objeto de un estudio detenido en las asignaturas correspondientes de la carrera.

En general, en estos trabajos es donde mejor se conjugan los objetivos de la Universidad: *docencia* e *investigación*.

Para el alumno, el desarrollo de un proyecto supone, por una parte, la consecución de una formación más específica en un determinado tema y, por otra, la utilización de métodos y medios propios de un entorno profesional. En este sentido, el trabajo constituye la prueba final de madurez antes de pasar al desarrollo de su etapa profesional.

Al docente, le permitirá renovar y ampliar conocimientos, de forma continua, sobre los temas impartidos, así como disponer de un punto de vista práctico de los conceptos teóricos que trata en clase. Este último punto resulta especialmente significativo e importante en una Ingeniería.

Para conseguir el mayor fruto de estos trabajos, es imprescindible que el alumno encuentre un ambiente adecuado tanto en el aspecto científico como en el humano, así como los medios materiales necesarios para su desarrollo. En este sentido, su financiación puede provenir, a parte de la propia Universidad, de la colaboración con la Industria. En la actualidad existe una activa relación Universidad-Empresa materializada en convenios de cooperación educativa que permiten acercar al alumno al mundo laboral y la realización de su Proyecto Fin de Carrera.

Por otra parte, es interesante que los proyectos no se reduzcan a la obtención de resultados teóricos, sino que todos ellos contemplen una parte de experimentación y diseño práctico en función de dichos resultados. Ésta sería la forma de vincular la investigación básica con la aplicada, prestando especial atención a esta última ya que los proyectos se desarrollan dentro de una Ingeniería. Es muy interesante para el universitario tener una experiencia investigadora que complete su formación. Los Proyectos Fin de Carrera, además, pueden constituir un elemento más, de cierta importancia, dentro de la actividad investigadora de los Departamentos.

Los proyectos suponen, por tanto, una oportunidad de enriquecimiento mutuo entre Universidad y alumno, que rompa con la tradicional y negativa idea que se puede tener de una docencia universitaria que solamente esté basada en las clases teórico-prácticas.

El interés por mantener unos criterios de calidad en los proyectos fin de carrera que se realizan en el Departamento de Informática y Automática de la Universidad de Salamanca, ha llevado a la preparación de una guía orientativa para su desarrollo y documentación [García et al., 2000a].

3.8 Aseguramiento de la calidad de la docencia

La calidad es un objetivo fundamental para los directivos de una empresa junto a los dos parámetros clásicos de su gestión: el dinero y el tiempo. El mercado actual es sumamente competitivo; no basta con producir masivamente los productos o servicios, vender es lo importante y sólo se produce un producto cuando se tiene la seguridad de aceptación por parte del cliente. Por tanto, lo realmente importante es satisfacer al cliente, conocer sus necesidades para luego definirlas en forma de requisitos a cumplir.

En la vida cotidiana, la calidad representa las propiedades inherentes a un objeto, de forma que pueda ser comparado con otros objetos de su especie para determinar si es mejor, igual o peor. Calidad es sinónimo de bondad, excelencia o superioridad.

Para la comunidad universitaria, la calidad debe ser un objetivo tan importante como lo es para la Empresa. El profesor de Universidad debe ver en la calidad un camino hacia la excelencia, tanto en su actividad investigadora como en su actividad docente. El antiguo papel del profesor como mero conferenciante y proveedor de conocimiento debe ser reemplazado por un nuevo rol: *el profesor como comunicador, consejero y gestor de la clase* [Null, 1996].

Centrando la atención en la actividad docente, la calidad puede verse como una actividad contractual donde la Universidad es la "*empresa*" y los clientes son varios. El cliente más directo es el estudiante, el producto que se le ofrece debe ser un currículo que cumpla unos requisitos acordes a los objetivos marcados por la titulación elegida. La sociedad es el cliente indirecto de la Universidad, donde el producto que se le ofrece está formado por titulados formados y preparados para introducirse en el mundo real y rendir de acuerdo con las necesidades de esa sociedad. Por último, el propio docente es a la vez mecanismo y cliente de la Universidad, al buscar por un lado la excelencia de conocimiento en una determinada materia y por otro lado la satisfacción personal conseguida cuando los estudiantes salen cumpliendo los requisitos de conocimientos que la Universidad marca y la adecuación que la sociedad demanda.

Para seguir las directrices de la mejora continua hacen falta dos elementos imprescindibles: en primer lugar se requiere de *una apuesta firme por unos servicios de calidad por parte de los órganos institucionales (Departamento, Facultad, Rectorado) que rigen directa o*

indirectamente la actividad docente del profesor y, por otra parte, de *una actitud personal favorable del propio docente en relación con la calidad en la docencia*.

La primera premisa se satisface en la Universidad de Salamanca por la apuesta del equipo rectoral actualmente vigente, y que se refleja en su programa electoral [Berdugo, 1998] bajo un conjunto de medidas destinadas a potenciar los "*programas de calidad de la docencia*", a saber:

- Exigencia en el cumplimiento del calendario docente.
- Introducción de sistemas de tutoría activa.
- Consolidación del programa de adquisición y ampliación de infraestructura para prácticas de laboratorio, así como la ampliación del programa de prácticas de campo.
- Potenciar las prácticas realizadas fuera de la Universidad, concertadas con empresas, instituciones y colegios profesionales.
- Seguir una política de aumento y mejora de la gestión en las asignaturas de libre disposición.
- Introducción de sistemas de evaluación de los alumnos que tengan en cuenta el rendimiento global y garanticen el uso de criterios y objetivos iguales para todos los alumnos en cada materia.
- Organización de sistemas de coordinación, seguimiento y mejora de los programas docentes en cada curso y en cada carrera.
- Programas de estímulo al uso de tecnologías avanzadas en la actividad docente.
- Apoyo a las iniciativas de formación del profesorado.
- Potenciar la participación de toda la comunidad universitaria en programas de evaluación y mejora de la calidad de la enseñanza.

La actitud personal hacia una política de calidad en docencia puede verse reflejada de muy diversas formas, con diferentes actividades realizadas de forma individual o colectiva. Es importante que esta actitud hacia la calidad sea algo voluntario, no impuesto, para que realmente se obtengan los beneficios buscados.

A continuación se van a comentar las actividades más destacadas que, desde la experiencia personal, he llevado a cabo para incidir en la mejora continua de mi labor docente y que se pueden resumir en tres apartados: *el plan de calidad, las tutorías activas* y *las experiencias de evaluación por pares.*

3.8.1 El plan de calidad

La motivación principal para elaborar un plan de calidad de una asignatura es la percepción personal, por parte del profesor responsable (o profesores) de la misma, de que se desea comenzar un proceso de mejora continua en la docencia de una o varias materias. Este proceso de mejora continua debe comenzar con un plan de actuación que establezca como primer objetivo el conocimiento profundo del desarrollo docente de la asignatura o asignaturas elegidas (como experiencia inicial se recomienda comenzar el proceso de mejora continua con una única asignatura).

Una forma adecuada de establecer un mayor grado de control sobre la asignatura es ampliar el documento de planificación docente de la asignatura a un plan de calidad, en el que principalmente se introduzcan dos nuevos aspectos [García et al., 1999c]:

- ***Una lista de objetivos***. Lista que debe verse completada con una serie de líneas de acción para lograr dichos objetivos. Cada una de estas líneas de acción debe quedar completamente definida mediante un marco formado por cuatro entradas:
    - *Cuándo:* Atributo temporal que indique el momento aproximado en el que ha de llevarse a cabo la línea de acción.
    - *Quién:* Faceta que indica los involucrados en la realización y éxito de la línea de acción.
    - *Medios:* Característica que representa los elementos, normalmente materiales, con los que se ha de contar para la culminación de la línea de acción.
    - *Evaluación:* Es imprescindible establecer un medio para evaluar cada una de las líneas de acción establecidas.

  Los objetivos pueden clasificarse en dos categorías ortogonales: por un lado *los objetivos que se pretenden lograr con la asignatura en sí*; y por otro lado *los objetivos personales del propio profesor como docente*.

- ***Una realimentación del proceso de evaluación***. Del análisis de los resultados de la evaluación de cada una de las líneas de acción, se debe obtener una valoración crítica de la asignatura que permita un nuevo planteamiento para el siguiente curso. Esta valoración crítica se puede centrar en la consecución de los tres apartados siguientes:

- *Puntos fuertes*: Se deben detectar las parcelas de la asignatura donde se haya conseguido un mayor impacto sobre los objetivos, con el fin de potenciarlos y mantener su nivel de influencia.
- *Áreas de mejora*: La detección de aquellas partes de la asignatura que más se han alejado de los objetivos, o aquellos objetivos que no han dado el fruto esperado, es un punto esencial en el proceso de mejora continua porque marca las áreas de actuación de cara al nuevo curso, donde la experiencia obtenida y contrastada por los medios de evaluación establecidos es fundamental y, por sí sola, justifica la realización del plan de calidad.
- *Redefinición del plan de calidad para el curso siguiente*: Actividad que coincide con la realización del plan de calidad para el curso siguiente.

Según lo visto se puede afirmar que el proceso de creación de un plan de calidad para una asignatura es un proceso iterativo que se alimenta de la experiencia obtenida en las iteraciones previas, tal y como se muestra en la Figura 3.4.

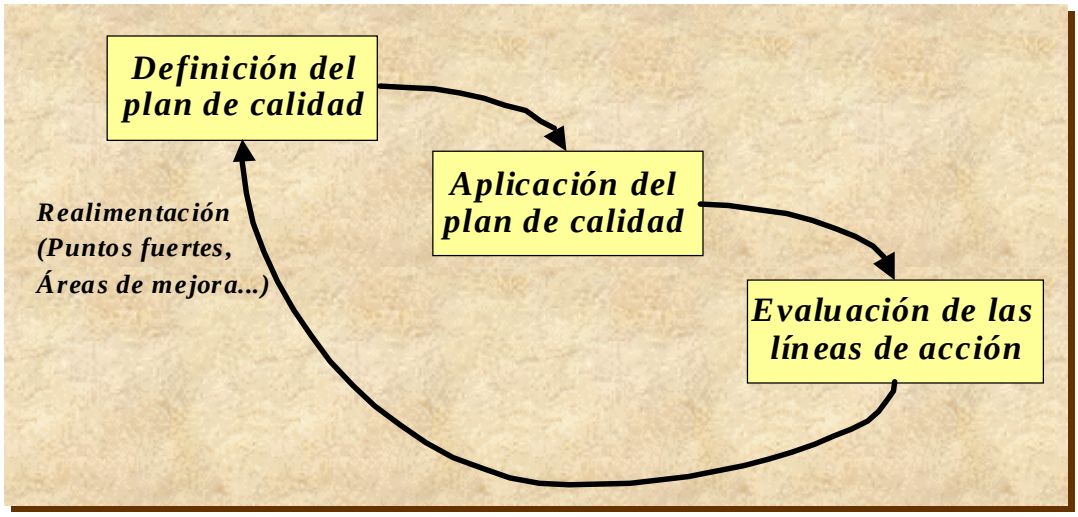


Figura 3.4. Proceso de creación del plan de calidad

El esqueleto básico del plan de calidad queda reflejado en la Figura 3.5.

Portada
Lista de cambios
Tabla de contenidos
1. Introducción
2. Objetivos de la asignatura
3. Objetivos personales
4. Otras actividades
5. Resultado de la evaluación
6. Programa de la asignatura
Anexo. Resumen de los objetivos

Figura 3.5. Esquema de la composición del plan de calidad

3.8.1.1 Experiencias prácticas en la aplicación del plan de calidad

La realización y puesta en práctica de un plan de calidad para la asignatura de Análisis e Ingeniería del Software se abordó por primera vez en el curso 1996-1997 en la Ingeniería Técnica en Informática de Gestión de la Universidad de Burgos [García, 1996].

Esta experiencia piloto nació como una inquietud personal del responsable de la asignatura por establecer un plan de acción que le permitiese obtener unas conclusiones sobre los métodos docentes que se aplicaban en una asignatura tradicionalmente árida, como es la Ingeniería del Software.

Los objetivos iniciales de esta iniciativa fueron principalmente dos: *definir la estructura del plan de calidad* y *establecer los medios de evaluación de las líneas de acción*; en este sentido destaca la confección de un cuestionario de evaluación destinado a los alumnos de la asignatura.

Estas labores iniciales fueron realizadas en colaboración con la Unidad Técnica de Calidad de la Universidad de Burgos, desde donde se guiaron todos los pasos dados.

Los resultados de esta primera experiencia fueron muy positivos, especialmente por el mayor conocimiento y domino que se adquiere de la asignatura impartida, y no se está haciendo referencia exclusivamente a los aspectos cognitivos de la misma. Los datos obtenidos permitieron establecer los puntos fuertes y, lo que era más importante, las áreas de mejora de la asignatura. Cabe destacar como una de las principales áreas de mejora detectadas la necesidad de definir más medios de control/evaluación aparte del cuestionario realizado.

Esta experiencia inicial sirvió para la creación del nuevo plan de calidad para la asignatura de Ingeniería del Software en el curso 1997-1998 [García, 1997a], pero se amplió la iniciativa a otra asignatura, Programación Avanzada [García, 1997b], que era la primera vez que se impartía en el Plan de Estudios de la Ingeniería Técnica en Informática de Gestión en la Universidad de Burgos.

La creación y puesta en marcha de una plan de calidad en una asignatura es siempre interesante y positivo, pero sus beneficios potenciales aumentan en los casos de asignaturas de nueva implantación, porque esta técnica ayuda a detectar los numerosos conflictos y problemas que tienen las asignaturas en sus inicios.

En el curso 1998-1999, se aplicaron las experiencias anteriores en la realización del plan de calidad de la asignatura Ingeniería del Software, pero dentro de la titulación de Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca [García, 1999]. De nuevo, la creación de un plan de calidad para esta asignatura sirvió de gran ayuda para el profesor, por encontrarse esta materia en un momento de transición hacia un nuevo Plan de Estudios que comenzó a impartirse el curso 1999-2000; además dicha transición también estaba condicionada por el comienzo del Segundo Ciclo de la Ingeniería Informática, donde la materia de Ingeniería del Software también está presente.

En el curso académico 1999-2000 se acometió la tarea de la realización de un plan de calidad para el bienio 1999-2001, pero no centrado en una única asignatura, sino que abarcara una unidad docente completa, la **Unidad Docente de Ingeniería del Software y Orientación a Objetos** [García et al., 2000b], que tiene competencias en dos titulaciones universitarias diferentes, aunque relacionadas: la Ingeniería Técnica en Informática de Gestión (Plan de 1997) y la Ingeniería Informática, ambas impartidas en la Universidad de Salamanca. Este plan de calidad es el que actualmente rige todas las asignaturas relacionadas con la materia de Ingeniería del Software en el Departamento de Informática y Automática de la Universidad de Salamanca, y a él se hace constante referencia a la hora de presentar las asignaturas que cubre este Proyecto Docente.

3.8.2 Las tutorías activas

Por acuerdo de la Junta Ordinaria de la Facultad de Ciencias de 9 de diciembre de 1998 se recomienda entre las medidas prioritarias, la implantación de un sistema de tutorías activas para todas las titulaciones de dicha Facultad. Esta iniciativa entra en vigor en el curso 1999-2000 [GAFC-USAL, 1999].

Pueden ser tutores activos los profesores ordinarios, los ayudantes doctores y los asociados a tiempo completo, también doctores, que de forma voluntaria deseen participar en el proyecto y tengan un buen conocimiento del Plan de Estudios de la carrera. Los tutores serán nombrados por la Junta de Facultad e incluidos en la programación docente anual.

La función del tutor es orientar e informar al estudiante en cualquiera de los aspectos académicos relacionados con su estancia en la Universidad, en especial en lo relativo a la organización de su recorrido curricular. En concreto, su tarea de ayuda y consejo se referirá, al menos, a la elección de asignaturas durante el proceso de matricula y a la planificación de los estudios, particularmente en lo tocante a la orientación profesional elegida por el alumno.

Todos los alumnos tienen derecho a la asignación de un tutor, aunque no están obligados a recibir su asesoramiento. Cada tutor puede tutelar a un máximo de quince alumnos. La asignación de alumnos a los tutores se hace en el primer curso de forma aleatoria; asignación que se mantendrá, en principio, durante todo el primer ciclo.

El tutor debe convocar a cada uno de los alumnos que tenga asignados al menos dos veces al año, una cada semestre. Asimismo, si el tutor lo considera conveniente, o cualquiera de los alumnos lo solicitara, pueden celebrarse otras entrevistas en cualquier momento del curso.

3.8.3 Experiencias de evaluación por pares

La evaluación del profesorado es una cuestión de indiscutible actualidad. Continuando por otras vías la línea marcada por la normativa derogada[17], la vigente LOU [BOCG, 2001] se remite al Gobierno (artículo 32), previo informe del Consejo de Coordinación Universitaria, para desarrollar el funcionamiento de la Agencia Nacional de Evaluación de la Calidad y Acreditación. Ésta será la que, en aras de la garantía de la calidad, (artículo 31.1 de la LOU), determine el rendimiento de "*las actividades docentes, investigadoras y de gestión del profesorado universitario*" (artículo 31.2.c).

Hasta la fecha, los artículos 134 a 136 de los Estatutos de la Universidad de Salamanca han regulado los criterios y el procedimiento a seguir para la evaluación ordinaria y extraordinaria del profesorado en esta Universidad. Así, la Memoria de actividades elaborada por el Consejo

[17] Los apartados 3 y 4 del artículo 45 de la LRU [MEC, 1983], recientemente derogada, establecían que los Estatutos de cada Universidad dispondrían los procedimientos necesarios para la evaluación periódica del rendimiento docente y científico del profesorado, imponiendo a los Departamentos la publicación de la Memoria sobre su labor docente e investigadora.

de Departamento, base para la evaluación ordinaria, debía incluir una valoración individualizada del rendimiento docente e investigador de cada uno de los profesores adscritos al mismo.

La opinión de los alumnos puede conocerse mediante su respuesta a cuestionarios. Evidentemente, los resultados de éstos han de ser analizados con suma precaución; su validez científica depende de la configuración de tales cuestionarios y del tratamiento de la información. No hay que olvidar, además, que la actitud del alumno, directamente implicado en la enseñanza y sometido a la evaluación del profesor, está llena de connotaciones subjetivas que pueden afectar a los resultados.

Sin embargo, la evaluación, que es un punto imprescindible en un proceso de mejora continua porque permite conocer el grado en que una actividad o un programa de calidad se ajusta a los objetivos preestablecidos [Balbin, 1999], se suele convertir en la causa más común de reticencia a la implantación de la calidad dentro de un colectivo, tendiendo a confundir el proceso de evaluación como “una caza de brujas” o encontrando en él un carácter sancionador, cuando en realidad, como indica el profesor J. M. Manso [Manso, 1997], lo que se está buscando es la evaluación de las actividades, no de las personas.

Los métodos de evaluación de un proceso de calidad en general, y particularmente en el apartado de la docencia universitaria, se pueden dividir en *métodos de autoevaluación o evaluación interna* y en *métodos de evaluación externa* [CU, 1997].

Los métodos de autoevaluación o evaluación interna son aquéllos que son responsabilidad del individuo o del grupo en que éste está inmerso. Este tipo de métodos de evaluación es muy importante para el seguimiento continuado del proceso de mejora continua que se esté realizando, dado que la cercanía de los recursos necesarios para llevar a cabo la evaluación no requiere de una gran inversión económica ni de tiempo. No obstante, el proceso de mejora continua en el apartado de la calidad se presta a ser refrendado por una evaluación externa al entorno en donde tiene lugar. Los métodos de evaluación externa tienen el inconveniente de que la disponibilidad de los recursos necesarios (principalmente recursos humanos en su papel de evaluadores) para su puesta en marcha requieren de un desembolso económico más grande, así como una mayor planificación entre evaluador(es) y evaluado(s).

A parte del sistema de evaluación del profesorado que sigue la Universidad de Salamanca en su *Plan para la Evaluación y Mejora de la Calidad en la Enseñanza*, personalmente se ha participado en otras experiencias relacionadas con la evaluación, tanto interna como externa.

En este sentido, aunque son varios los métodos para la evaluación de la calidad de la docencia [Carbone y Kaasbrll, 1998], se optó por la utilización del protocolo de evaluación

externa, denominado evaluación por pares, que se describe en [García et al., 1998; García et al., 1999b; García et al., 1999d], para la evaluación externa de la actividad docente en asignaturas del perfil de la plaza a concurso, con unos resultados altamente positivos.

A continuación, y a forma de esquema, se enumeran las fases de que consta el protocolo de evaluación por pares [García et al., 1999d]:

1. Inicio y planificación del proceso de evaluación por pares.
2. Redacción de una memoria sobre la asignatura por parte de su responsable.
3. Estudio de la memoria por parte del evaluador.
4. Entrevista del evaluador con alumnos que hayan cursado la asignatura.
5. Informe provisional del evaluador.
6. Entrevista entre evaluador y evaluado.
7. Informe final del proceso de evaluación.
8. Reunión para la **autoevaluación** del proceso de evaluación.
9. Aplicación de los resultados del proceso de evaluación en la nueva revisión del plan de calidad para la asignatura evaluada.

3.9 Referencias

[Balbin, 1999] Balbin, I. "*Is Your Degree Quality Endorsed?*". In Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 60-63. ACM. 1999.

[Beard, 1974] Beard, R. "*Pedagogía y Didáctica en la Enseñanza Universitaria*". Oikos- Tau, 1974.

[Beckman, 1999] Beckman, K. "*Directory of Industry and University Collaborations with a Focus on Software Engineering Education and Training, Version 7*". Special Report CMU/SEI-99-SR-001, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). February 1999.

[Beckman et al., 1997] Beckman, K., Khajenoori, S., Coulter, N., Mead, N. R. "*Collaborations: Closing the Industry-Academia Gap*". IEEE Software, 14(6):49-57. November/December 1997.

[Beckman et al., 1999] Beckman, K., Lawrence, J., Mead, N., O'Mary, G., Parish, C., Walker, H. "*Industry/University Collaborations: Different Perspectives Heighten Mutual Opportunities*". Software Engineering Institute. http://www.sei.cmu.edu/topics/collaborating/ed/indust-univ-collabs.html. August 1999.

[Berdugo, 1998] Berdugo Gómez de la Torre, I. "*Candidatura a Rector Encabezada por Ignacio Berdugo Gómez de la Torre. Programa Electoral*". Universidad de Salamanca, 1998.

[Berners-Lee et al., 1994] Berners-Lee, T., Cailliau, R., Loutonen, A., Nielsen, H. F., Secret, A. "*The World-Wide Web*". Communications of the ACM, 37(8):76-82. August 1994.

[Bloom, 1956] Bloom, B. "*Taxonomy of Educational Objectives: Handbook I: Cognitive Domain*". David McKay, 1956.

[BOCG, 2001] Boletín Oficial Cortes Generales. "*Ley Orgánica de Universidades*". Congreso de los Diputados. Serie A. Núm. 45-13. Páginas 463-495. 26 de diciembre de 2001.

[Brown y Neilson, 1996] Brown, C. E., Neilson, N. L. "*Enhancing Business Classes with the World Wide Web*". Journal of Education for Business, 71(6):317-324, 1996.

[Carbone y Kaasbrll, 1998] Carbone, A., Kaasbrll, J. J. "*A Survey of Methods Used to Evaluate Computer Science Teaching*". In Proceedings of the 6th Annual Conference on the Teaching of Computing/3rd Annual Conference on Integrating Technology into Computer Science Education on Changing the Delivery of Computer Science Education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 41-45. ACM. 1998.

[Carter, 1999] Carter, J. "*Collaboration or Plagiarism: GAT Happens when Students Work Together*". In Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 52-55. ACM. 1999.

[Crawford y Fekete, 1997] Crawford, K., Fekete, A. "*What Exams Results Really Measure?*". In Proceedings of the Second Australasian Conference on Computer Science Education, ACSE '97. (July 2-4, 1997, The University of Melbourne, Australia). ACM. Pages 185-190. 1997.

[Chalk, 2000] Chalk, P. "*Apprenticeship Learning of Software Engineering Using Webworlds*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITICSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 112-115. ACM Press, 2000.

[Chrisman y Harvey, 1998] Chrisman, N. R., Harvey, F. J. "*Extending the Classroom: Hypermedia-Supported Learning*". Journal of Geography in Higher Education, 22(1):11-20, 1998.

[CU, 1997] Consejo de Universidades. "*Plan Nacional de la Calidad de las Universidades. Guía de Evaluación*". Secretaría General del Consejo de Universidades, 1997.

[Davis et al., 1997] Davis, G. B., Gorgone, J. T., Couger, J. D., Feinstein, D. L., Longenecker, Jr. H. E. (Editors) "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS and AITP, 1997.

[Dawson y Newsham, 1997] Dawson, R., Newsham, R. "*Introducing Software Engineers to the Real World*". IEEE Software, 14(6):37-43. November/December 1997.

[Díaz, 1999] Díaz, O. "*¿Para Qué la Tesis Doctoral?*". Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISDB'99). (Cáceres, 24-25 de noviembre de 1999). Páginas 3-6. 1999.

[Díaz-Herrera y Powell, 1998] Díaz-Herrera, J., Powell, G. M. "*Educating Industrial-Strength Software Engineers*". In Proceedings of the 11th Conference on Software Engineering Education and Training (CSEE&T '98). (February 22-25, 1998. Atlanta, GA – USA). IEEE Computer Society. Pages 139-150. 1998.

[GAFC-USAL, 1999] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1998-1999*". Ediciones Universidad de Salamanca, 1999.

[García, 1996] García Peñalvo, F. J. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1996-1997. 1996.

[García, 1997a] García Peñalvo, F. J. "*Plan de Calidad para la Asignatura Análisis e Ingeniería del Software*". Segundo Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

[García, 1997b] García Peñalvo, F. J. "*Plan de Calidad para la Asignatura Programación Avanzada*". Tercer Curso de la Ingeniería Técnica en Informática de Gestión. Universidad de Burgos. Curso 1997-1998. 1997.

[García, 1999] García Peñalvo, F. J. "*Plan de Calidad para la Asignatura Ingeniería del Software*". Tercer Curso de la Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Curso 1998-1999. 1999.

[García, 2002] García Peñalvo, F. J. "*Herramienta de Autor para el Desarrollo de Material Didáctico Multimedia. Memoria de Resultados del Proyecto SA002/01*". Departamento de Informática y Automática. Universidad de Salamanca. Enero, 2002.

[García et al., 1998] García Peñalvo, F. J., Montero García, E., Arranz Val, P. "*Proceso de Evaluación por Pares. Una Experiencia Práctica*". En las actas de las II Jornadas de Calidad y Universidad: Calidad en la Docencia (Burgos, 10-11 de noviembre de 1998).

[García, et al., 1999a] García Carrasco, J., García del Dujo, Á., López Fernández, R., Mompó Gómez, R., Navazo Suela, Mª A., Pérez Juárez, Mª Á., Redoli Granados, J., Regueras Santos, L. Mª, Rodríguez Pajares, B., Verdú Pérez, Mª J. "*Nuevas Tecnologías y Formación*". PCWEEK. Editorial América Ibérica, Madrid. 1999.

[García et al., 1999b] García Peñalvo, F. J., Montero García, E., Arranz Val, P. "*Protocolo de Evaluación por Pares*". En las actas del VII Congreso Iberoamericano de Educación Superior en Computación – CIESC'99 (Asunción-Paraguay, 29 agosto - 3 septiembre de 1999). Páginas 38-47. 1999.

[García et al., 1999c] García Peñalvo, F. J., Moreno García, Mª N., González Talaván, G., Moreno Montero, Á. Mª. "*Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática*". Actas del Congreso Nacional de Informática Educativa CONIED'99.

Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de noviembre de 1999). Resumen en página 46 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 1999d] García Peñalvo, F. J., Moreno García, Mª N., Montero García, E., Arranz Val, P. "*Evaluación del Profesorado: Un Protocolo de Evaluación por Pares*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de noviembre de 1999). Resumen en página 47 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 2000a] García Peñalvo, F. J., Maudes Raedo, J. M., Piattini Velthuis, M. G., García-Bermejo Giner, J. R., Moreno García, Mª N. "*Proyecto de Final de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación*". Departamento de Informática y Automática de la Universidad de Salamanca. Versión 1.52. http://tejo.usal.es/~fgarcia/doc/pfc.pdf. Marzo, 2000.

[García et al., 2000b] García Peñalvo, F. J., Moreno García, Mª N., García-Bermejo Giner, J. R. y Luis Reboredo, A. de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García et al., 2002] García Peñalvo, F. J., Moreno Montero, Á. Mª, Gil González, A. B., López Fernández, R., García Carrasco, J. "*Espacios Virtuales Educativos como Complemento a las Actividades Formativas Clásicas en el Ámbito de Internet*". Revista de Enseñanza y Tecnología. (En prensa). 2002.

[García y García, 2002] García Peñalvo, F. J., García Carrasco, J. "*Los Espacios Virtuales Educativos en el ámbito de Internet: Un Refuerzo a la Formación Tradicional*". Teoría de la Educación. Educación y Cultura en la Sociedad de la Información, Vol. 3. (En prensa). 2002.

[García-Molina et al., 2001] García-Molina, H., Ullman, J. D., Widom, J. D. "*Database Systems: The Complete Book*". Prentice-Hall, 2001.

[Gil et al., 2001] Gil, A. B., García, F. J., Castilla, E., Vicente, A., Luis, A. de, Theron, R. "*Herramienta para al Generación de un Libro Electrónico Educativo*". Actas de las III Jornadas Multimedia Educativo. J. L. Rodríguez Illera, A. Escofet Roig, B. Gros Salvat, J. Quintana Albalat y M. J. Rubio Hurtado (editores) – (Universitat de Barcelona, España - 25 y 26 de junio de 2001). Resumen en página 56, y ponencia en versión digital (CD-ROM), páginas 277-286. 2001.

[Glass, 1997] Glass, R. L. "*Revisiting the Industry/Academe Communication Chasm*". Communications of the ACM, 40(6): 11-13. June 1997.

[Gómez, 1981] Gómez Pérez, R. Prólogo de la obra de J. Pujol y J. P. Fons. "*Los Métodos de Enseñanza Universitaria*". Eunsa. 1981.

[González, 1996] González Casal, J. "*Estudio de la Profesión Informática en España durante 1995*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (29):13-16. 1996.

[Greening, 1997] Greening, T. "*Examining Student Learning of Computer Science*". In Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San José, CA – USA). Pages 63-66. ACM. 1997.

[Gullbert, 1989] Gullbert, J. J. "*Guía Pedagógica*". Organización Mundial de la Salud. Quinta Edición. Editado por Instituto de Ciencias de la Educación. Universidad de Valladolid. 1989.

[Hale, 1964] Hale. "*Reports of the Committee on University Teaching Methods"*. Londres, University Grants Committee, 1964.

[Hare, 2000] Hare, C. "*New Technologies and Education of Information Professionals*". Teoría de la Educación. Educación y Cultura en la Sociedad de la Información.Vol. 2. 2000.

[Holmes, 1999] Holmes, W. N. "*The Myth of the Educational Computer*". IEEE Computer, 32(9):36-42. September 1999.

[Huang, 2001] Huang, A. "*Innovative Use of Email for Teaching*". Communications of the ACM, 44(11):29-31. November 2001.

[Kuhn, 1998] Kuhn, S. "*The Software Design Studio: An Exploration*". IEEE Software, 15(2):65-71. March/April 1998.

[Lafourcade, 1974] Lafourcade, P. "*Planteamiento, Conducción y Evaluación de la Enseñanza Universitaria*". Kapeluzs, 1974.

[Lara, 1997] Lara Ortega, F. "*Principios de Calidad en la Docencia*". Actas de las I Jornadas sobre Calidad y Universidad. Universidad de Burgos. Noviembre, 1997.

[Lewis, 1998] Lewis, J. D. "*How the Internet Expands Educational Options*". Teaching Excepcional Children, 30(5):34-42, 1998.

[Lloyd, 1968] Lloyd, D. H. "*A Concept of Improvement of Learning Response in the Taught Lesson*". Visual Education, 1968.

[Mager, 1976] Mager, R. F. "*Creación de Actitudes y Aprendizajes*". 2ª edición. Colección Biblioteca del Educador. Editorial Marova. 1976.

[Manso, 1997] Manso Martínez, J. M. "*Docencia en la Universidad: Lo que Es y lo que Debe Ser*". Actas de las I Jornadas sobre Calidad y Universidad – Hacia una Universidad de Calidad. Burgos, 10-13 de Nov. de 1997.

[Marchionini y Maurer, 1995] Marchionini, G., Maurer, H. "*The Roles of Digital Libraries in Teaching and Learning*". Communications of the ACM, 38(4):67-75. April 1995.

[Mason y Woit, 1998] Mason, D. V., Woit, D. M. "*Integrating Technology into Computer Science Examinations*". In Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 140-144. 1998.

[Mead et al., 2000] Mead, N., Unpingco, P., Beckman, K., Walker, H., Parish, C., O'Mary, G. "*Industry/University Collaborations. Different Perspectives Heighten Mutual*

Opportunities". Crosstalk, The Journal of Defense Software Engineering, 13(3):10-15. March 2000.

[MEC, 1983] Ministerio de Educación y Ciencia. "*Ley Orgánica de Reforma Universitaria*". Servicio de Publicaciones del MEC, 1983.

[Mercuri, 1998] Mercuri, R. "*In Search of Academy Integrity*". Communications of the ACM, 40(5):136. May 1998.

[Mora et al., 1995] Mora Núñez, N., Prima Rodríguez, M., González López, R., Crespo Faja, F., Díaz Vázquez, J. E. "*Propuesta de Organización Docente para Asignaturas Troncales de Pocos Créditos, Basada en Principios Constructivistas*". Actas de las III Jornadas Universitarias sobre Innovación Educativa en las Enseñanzas Técnicas (Ferrol – A Coruña. Septiembre, 1995). Tomo II. Páginas 285-292. 1995.

[Moreno et al., 2000] Moreno, Á. Mª., García, F. J., García, J, Alonso, L. "*Componentes Software para Entornos Virtuales de Educación*". En las actas de las Jornadas UNED-2000 Conocimiento, Método y Tecnologías en la Educación a Distancia. Páginas 122-126. 2000.

[Neumann, 1998] Neumann, P. G. "*Risks of E-Education*". Communications of the ACM, 40(10):136. October 1998.

[Nishida et al., 1996] Nishida, T., Saitoh, A., Tsujino, Y., Tokura, N. "*Lecture Supporting System by E-mail and WWW*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 280-284. ACM. 1996.

[Null, 1996] Null, L. "*Applying TQM in the Computer Science Classroom*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 120-124. ACM. 1996.

[Orden, 1985] Orden Hoz, A. de la. "*Modelos de Evaluación Universitaria*". Revista Española de Pedagogía. Nº 169. 1985.

[Paulisse y Polik, 1999] Paulisse, K. W., Polik, W., F. "*Use of WWW Discussion Boards in Chemestry Education*". Journal of Chemical Education, 76(5):704-708, 1999.

[Paxton, 1996] Paxton, J. T. "*Webucation: Using the Web as Classroom Tool*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 285-289. ACM. 1996.

[Pozo, 1982] Pozo Pardo, A. del. "*La Didáctica de Hoy*". Hijos de Santiago Rodríguez, Burgos, 1982.

[Riser y Gotterbarn, 1998] Riser, R., Gotterbarn, D. "*On-line Journal: A tool for enhancing Student Journals*". In Proceedings of the 6th Annual Conference on the Teaching of Computing/3rd Annual Conference on Integrating Technology into Computer Science Education on Changing the Delivery of Computer Science Education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 203-205. ACM. 1998.

[Rodríguez, 1980] Rodríguez Dieguez, J. "*Didáctica General*". Cincel, 1980.

[Rosbottom, 2001] Rosbottom, J. "*Hybrid Learning – A Safe Route into Web-based Open and Distance Learning for the Computer Science Teacher*". In Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education – ITiCSE'01. (June 2001, Canterbury, UK). Pages 89-92. ACM Press. 2001.

[Rosbottom et al., 2000] Rosbottom, J., Crellin, J., Fysh, D. "*A Generic Model for On-line Learning*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITiCSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 108-111. ACM Press, 2000.

[Sanchís y Torralba, 1997] Sanchís Marco, F., Torralba Martínez, J. M. "*Seguimiento del Mercado Laboral como Guía para los Diseños Curriculares. El Caso de las Ingenierías Informáticas*". Revista ALI BASE. Asociación de Doctores, Licenciados e Ingenieros en Informática. (31):18-23. 1997.

[Seal y Przasnyski, 2001] Seal, K. C., Przasnyski, Z. H. "*Using the World Wide Web for Teaching Improvement*". Computers & Education, 36:33-40, 2001.

[Ullman y Widom, 1997] Ullman, J., Widom, J. "*A First Course in Database Systems*". Prentice-Hall, 1997.

[Ullman y Widom, 2001] Ullman, J., Widom, J. "*A First Course in Database Systems*". 2nd edition.Prentice-Hall, 2001.

[UNESCO, 1973] UNESCO. "*Aprender a Ser*". Informe de la Comisión Internacional para el Desarrollo de la Educación. Editorial Alianza, Madrid. 1973.

[Veldenz y Dennis, 1998] Veldenz, H. C., Dennos, J. W. "*The Internet and Education in Surgey*". American Surgeon, 64(9):877-881, 1998.

[Veraart y Wright, 1996] Veraart, V. E., Wright, S. L. "*Supporting Software Engineering Education with Local Web Site*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 275-279. ACM. 1996.

[Vetter y Severance, 1997] Vetter, R. J., Severance, C. "*Web-Based Education Experiences*". IEEE Computer, 30(11):139-141. November 1997.

[Villarreal y Butler, 1998] Villarreal, E. E., Butler, D. "*Giving Computer Science Students a Real-World Experience*". In Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 40-44. 1998.

[Wohlin y Regnell, 1999] Wohlin, C., Regnell, B. "*Achieving Industrial Relevance in Software Engineering Education*". In Proceedings of the 12th Conference on Software Engineering Education and Training (CSEE&T '99). (22-24 March, 1999. New Orleans, Louisiana – USA). IEEE Computer Society. Pages 16-25. 1999.

[Yu y Yu, 2002] Yu, F-Y., Yu, H.-J. J. "*Incorporating E-Mail into the Learning Process: Its Impact on Student Academic Achievement and Attitudes*". Computers & Education, 38:117-126, 2002.

Capítulo 4
Definición del Proyecto Docente

El desarrollo de un Proyecto Docente para una determinada disciplina implica, en primer lugar, la identificación clara y precisa del conjunto de conceptos y conocimientos científico/técnicos que la misma engloba. En el caso de la Ingeniería del Software esta es una tarea que presenta algunas dificultades debido al amplio conjunto de materias que abarca y a la constante evolución a la que se ve sometida como consecuencia de los rápidos cambios que sufre la tecnología del software. Por este motivo, previo al desarrollo de la propuesta concreta de contenido, se presenta una visión global de la Ingeniería del Software como disciplina mediante un análisis de su marco histórico y conceptual. El objetivo no es realizar una exposición exhaustiva, sino mostrar aquellos conceptos, aportaciones y resultados que se consideran relevantes para el contenido de las asignaturas objeto de la propuesta docente. Sin duda, una característica de la Ingeniería del Software como disciplina universitaria, frente a otras disciplinas más establecidas, es el dinamismo con el que cambian sus conceptos y herramientas. Por ello es importante atender a las recomendaciones sobre la enseñanza de la Ingeniería del Software que realizan los organismos y organizaciones internacionales relacionados con la Ciencia de la Computación, los Sistemas de Información y la propia Ingeniería del Software que también se recogen en este capítulo. Atender a este tipo de recomendaciones permite que los conocimientos transmitidos y las habilidades adquiridas por los alumnos respondan a las necesidades profesionales reales y no queden obsoletos en poco tiempo. Finalizamos este capítulo analizando el contexto profesional en el que se van a mover los futuros titulados con el fin de tener presentes estos aspectos en el diseño de las asignaturas y atender de esa forma las demandas de la sociedad.

4.1 Introducción

Es un hecho ampliamente asumido que la Informática es hoy en día un factor social de gran relevancia. El objetivo de las titulaciones donde se circunscribe la plaza objeto de concurso, la Ingeniería Técnica en Informática de Sistemas y la Ingeniería Informática, es la formación de profesionales (ingenieros informáticos) que se puedan incorporar a un mercado laboral, actualmente en plena demanda de estos titulados, con unas garantías plenas de calidad en cuanto a la función que deberán desempeñar en sus puestos de trabajo.

Actualmente, y a consecuencia de la falta de madurez que todavía sufre la Informática, es difícil precisar qué se entiende por un ingeniero informático, ya sea técnico o superior. Esta bruma que rodea a la figura de los informáticos es aprovechada por multitud de personas, ya sean tituladas o no, para acogerse a puestos laborales donde se interacciona de una manera u otra con un computador. Es por este motivo por el que se debe hacer un esfuerzo por diferenciar a los *ingenieros informáticos*[18] de los informáticos en general.

En este sentido, y con un ámbito internacional, en 1998 se formó el Software Engineering Cordinating Committee (SWECC) bajo el auspicio de IEEE-CS y ACM. Su misión es cuidar la evolución de la Ingeniería del Software como una disciplina profesional dentro del mundo de la Informática. Para lo cual se pretende documentar el cuerpo de conocimientos de la disciplina, recomendar un criterio de acreditación de los titulados, desarrollar un modelo de currículo, mantener un código ético y definir un conjunto de estándares. Actualmente, los proyectos que se encuentran en marcha son:

- ***Software Engineering Body of Knowledge (SWEBOK) – Cuerpo de conocimiento de la Ingeniería del Software:*** Tiene como objetivos *clarificar y definir los límites de la ingeniería del software con respecto a otras disciplinas* y *ofrecer los fundamentos para el desarrollo de una propuesta curricular y el material para la certificación de los profesionales.* Este proyecto ha obtenido como resultado la guía con el cuerpo de conocimiento de Ingeniería del Software, que tras pasar por la denominada *Straw Man Version* (versión de hombre de paja) [Bourque et al., 1998], actualmente ha finalizado la *Stone Man Version v0.95* (versión del hombre de piedra) [Abran et al., 2001b]. En este momento se abre un período dos años para la utilización de esta guía, obteniéndose así la

[18] Como ya se ha comentado antes en este proyecto docente, cada vez existe una mayor tendencia en el ámbito internacional a denominar Ingeniería del Software a la titulación que en España equivaldría a la Ingeniería Informática, sin entrar en las diferenciaciones entre titulados de primer o segundo ciclo en nuestro país.

realimentación necesaria para comenzar en el tercer año la *Iron Man Versión* (versión del hombre de hierro).

- ***Software Engineering Education Project – Proyecto de Educación en Ingeniería del Software* [ACM/IEEE-CS, 1999a]*:*** Reúne dos subproyectos, *un modelo de acreditación para programas universitarios* [ACM/IEEE-CS, 1998] y *el plan para el proyecto de educación en Ingeniería del Software (Plan for the Software Engineering Education Project – SWEEP)* [ACM/IEEE-CS, 1999b].

- ***Software Engineering Code of Ethics and Professional Practice – Código de Ética y Práctica Procesional de Ingeniería de Software* [ACM/IEEE-CS, 1999c]*:*** El código ético de la profesión fue aprobado por las asociaciones ACM e IEEE-CS en su versión 5.2 [ACM/IEEE-CS, 1999c][19] como el estándar para la enseñanza y la práctica de la Ingeniería del Software. Este código ha sido desarrollado por un grupo dirigido por Donald Gotterbarn**,** siendo propuesto tras varias versiones[20] y después de revisar los códigos éticos de otras sociedades. El código ético contiene ocho principios relacionados con el comportamiento y las decisiones tomadas por los profesionales. Un breve resumen del mismo se presenta en el Cuadro 4.1.

Tras la presentación de los esfuerzos en pro de la definición de la profesión de *ingeniero informático* o *ingeniero del software* realizados por ACM e IEEE-CS, se vuelve a centrar la atención en el contexto nacional, y más concretamente en el de la Universidad de Salamanca, para definir de una forma más precisa el perfil de los titulados en Ingeniería Técnica en Informática de Sistemas e Ingeniería en Informática; de forma que éstos sean profesionales que *utilicen un sólido conjunto de fundamentos de Ciencias de la Computación para solventar problemas reales, interaccionando con clientes y usuarios, debiendo hacer uso de una capacidad de comunicación oral y escrita correcta y fluida, y actuando siempre de acuerdo al código ético marcado por su profesión.*

[19] En [Dolado, 1999] el Dr. D. Javier Dolado, de la Universidad de San Sebastián, ha traducido el código ético en su versión 5.2 al español.

[20] La versión del código ético más conocida, con anterioridad a la aprobación de la versión 5.2, fue la versión 3.0 [Gotterbarn et al., 1997a], [Gotterbarn et al., 1997b]. Precisamente en [Gotterbarn et al., 1999a] y en [Gotterbarn et al., 1999b] se comenta la versión 5.2 comparándola con la versión 3.0.

Los códigos éticos tienen una función esencial para caracterizar una profesión, y para que una disciplina adquiera el carácter de profesión debe poseer un código de conducta.

Se pueden resumir las principales funciones de los códigos éticos en los siguientes apartados [Bowyer, 1996]:

1) Simbolizar una profesión.
2) Proteger los intereses del grupo.
3) Inspirar buena conducta.
4) Educar a los miembros de la profesión.
5) Disciplinar a sus afiliados.
6) Fomentar las relaciones externas.
7) Enumerar los principios morales básicos.
8) Expresar los ideales a los que se debe aspirar.
9) Mostrar las reglas básicas de comportamiento.
10) Ofrecer guías de comportamiento.
11) Enumerar derechos y responsabilidades.

Los códigos de conducta van más allá de la pura normativa legal, ya que ayudan a guiar el comportamiento en multitud de situaciones para las que no existe referencia legal.

El código propuesto por ACM y IEEE-CS tiene como objetivo documentar las responsabilidades y obligaciones éticas y profesionales, en un intento de educar y aleccionar a los ingenieros del software, a la vez que informar al público sobre las responsabilidades que son importantes para la profesión. Este código ético prima el bienestar y la calidad de vida del público en general, en cuanto a todas las decisiones relacionadas con la Ingeniería del Software [Gotterbarn et al., 1999b].

Los ingenieros informáticos deben responsabilizarse de que al llevar a cabo sus actividades lo hagan con beneficio y respeto a la profesión que ejercen. De acuerdo a sus compromisos con la salud, la seguridad y el bien público, los ingenieros informáticos deben seguir los siguientes ocho principios [Gotterbarn, 1999]:

1. **Sociedad:** Los ingenieros del software actuarán de manera coherente con el interés general.
2. **Cliente y empresario:** Los ingenieros del software deberán actuar de tal modo que se sirvan los mejores intereses para sus clientes y empresarios, y consecuentemente con el interés general.
3. **Producto:** Los ingenieros del software deberán garantizar que sus productos y las modificaciones relacionadas con ellos cumplen los estándares profesionales de mayor nivel que sea posible.
4. **Juicio:** Los ingenieros del software deberán mantener integridad e independencia en su valoración profesional.
5. **Gestión:** Los gestores y líderes en Ingeniería del Software suscribirán y promoverán un enfoque ético a la gestión del desarrollo y el mantenimiento del software.
6. **Profesión:** Los ingenieros del software deberán progresar en la integridad y la reputación de la profesión, de forma coherente con el interés público.
7. **Compañeros:** Los ingenieros del software serán justos y apoyarán a sus compañeros.
8. **Persona:** Los ingenieros del software deberán participar en el aprendizaje continuo de la práctica de su profesión y promoverán un enfoque ético en ella.

Cuadro 4.1. Resumen del código ético de ACM/IEEE-CS, versión 5.2

4.2 El perfil de formación

Las actividades a realizar por la persona que ocupe la plaza a concurso se centran en impartir docencia en materia de Ingeniería del Software en las titulaciones de Ingeniería Técnica en Informática de Sistemas e Ingeniería Informática en la Universidad de Salamanca.

El siguiente paso es identificar qué asignaturas se ajustan al perfil de formación dentro de los Planes de Estudios vigentes en estas titulaciones, Plan de 1997 [BOE, 1997] y Plan de 1999 [BOE, 1999] para la Ingeniería Técnica y la Ingeniería Informática respectivamente.

En el Capítulo 2 ya se presentaron las asignaturas de dichos Planes de Estudios, concretamente, las tablas 2.17 y 2.18 muestran las asignaturas troncales/obligatorias y optativas respectivamente para la Ingeniería Técnica en Informática de Sistemas; mientras que las tablas 2.20 y 2.21 presentaban el mismo tipo de información para la Ingeniería Informática.

Una descripción más detallada de los Planes de Estudios vigentes para ambas titulaciones se puede encontrar en la *Guía Académica de la Facultad de Ciencias* para el Curso 2001-2002 [GAFC-USAL, 2001].

De las asignaturas que se incluyen en estos Planes de Estudios hay varias que podrían considerarse como relacionadas con la materia de Ingeniería del Software, y así se recoge en el Plan de Calidad de la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000]. Pero de una manera directa hay tres asignaturas que se ajustan perfectamente al perfil de la plaza a concurso: **Ingeniería del Software** que se imparte en el primer cuatrimestre del tercer curso de la Ingeniería Técnica en Informática de Sistemas, **Análisis de Sistemas** y **Administración de Proyectos Informáticos**, ambas pertenecientes a la Ingeniería Informática, que con un carácter anual se imparten en el primer y segundo curso respectivamente.

En el caso de la Ingeniería Técnica, el Plan de Estudios vigente (Plan de 1997) comenzó su andadura en el curso 1997/1998 [GAFC-USAL, 1997], impartiéndose la asignatura de Ingeniería del Software por primera vez en el curso académico 1999/2000 [GAFC-USAL, 1999]. No obstante, existe una asignatura equivalente, con las mismas características de nombre, obligatoriedad y carga docente en el Plan de Estudios anterior (Plan de 1992), como se puede comprobar en [GAFC-USAL, 1998].

En la actualidad la asignatura Ingeniería del Software del Plan de 1992 aparece como asignatura sin docencia para aquellos alumnos que, matriculados en el Plan de 1992, no la hayan superado. Además, ha quedado establecido el mecanismo de convalidación oportuno entre las asignaturas del Plan de 1992 y sus homónimas del Plan de 1997.

4.3 Ingeniería del Software

Previamente al desarrollo de la propuesta concreta de los programas de las asignaturas que cubren la materia de Ingeniería del Software en las Ingenierías en Informática en la Universidad

de Salamanca, y que se desarrolla en el Capítulo 5, se presenta una visión global de la Ingeniería del Software como disciplina mediante un análisis de su marco histórico y conceptual. El objetivo no es realizar una exposición exhaustiva, sino mostrar aquellos conceptos, aportaciones y resultados que se consideran relevantes para el contenido de las asignaturas objeto de la propuesta docente. De esta forma los contenidos se adaptarán a las tendencias actuales consolidadas en la teoría, tecnología, práctica y aplicación del software a los sistemas basados en computadores.

El desarrollo de un Proyecto Docente para una determinada disciplina implica, en primer lugar, la identificación clara y precisa del conjunto de conceptos y conocimientos científico/técnicos que la misma engloba. En el caso de la Ingeniería del Software es una tarea que presenta algunas dificultades debido al amplio conjunto de materias que abarca y a la constante evolución a la que se ve sometida como consecuencia de los rápidos cambios que sufre la tecnología del software. Una primera aproximación a esta tarea puede venir dada por la definición de Ingeniería del Software.

4.3.1 Definición de Ingeniería del Software

Antes de hacer un repaso por algunas de las muchas definiciones que de Ingeniería del Software se han dado, se va comentar el origen y polémica que el propio término ha suscitado.

La introducción del término Ingeniería del Software se produce en la primera conferencia sobre Ingeniería del Software patrocinada por la OTAN, celebrada en Garmisch (Alemania) en octubre de 1968 [Naur y Randell, 1969], atribuyéndose la paternidad del término a Fritz Bauer [Randell, 1998].

Fue tal la aceptación de esta conferencia que se consiguió un nuevo patrocinio de la OTAN para una segunda conferencia sobre Ingeniería del Software, que tendría lugar un año más tarde en Roma (Italia) [Buxton y Randell, 1970], con unos resultados menos esperanzadores que los producidos en la primera conferencia. De hecho, no se produjo ninguna petición de que continuara la serie de conferencias de la OTAN, lo cual no influyó para que a partir de entonces se utilizara con gran profusión el nuevo término para describir los trabajos realizados, aunque quizás sin un consenso real sobre su significado.

En el contexto educativo, sin duda alguna, lo que más controversia ha levantado es el propio nombre del término, centrándose la discusión en la pregunta *¿es la Ingeniería del Software realmente una Ingeniería?* [Tomayko, 2000; Bryant, 2000; Lewerentz y Rust, 2000].

Los argumentos que se dan para sustentar una respuesta negativa se pueden resumir en dos categorías. La primera reuniría a aquéllos que se ciñen a la definición literal de Ingeniería dada por algunos diccionarios o sociedades profesionales, argumentando que en estas definiciones se hace mención a productos tangibles derivados del uso efectivo de materiales y fuerzas naturales, mientras que el software ni es tangible, ni utiliza materiales y/o fuerzas naturales para su concepción.

La segunda categoría estaría formada por los que arguyen que una disciplina ingenieril evoluciona desde una profesión, y la profesión relacionada con el software no ha evolucionado lo suficiente para ser considerada una Ingeniería.

Por el contrario, son muchos los que están a favor de la utilización y difusión del término *Ingeniería del Software*, tomando como un estándar de facto la utilización reiterada del término en la bibliografía especializada.

Quizás la defensa más fuerte y adecuada de la Ingeniería del Software como Ingeniería venga de la mano de Mary Shaw que justifica que si tradicionalmente se ha definido Ingeniería como "*la creación de soluciones rentables a problemas prácticos mediante la aplicación del conocimiento científico para la construcción de cosas al servicio de la humanidad*" [Shaw, 1990], entonces el desarrollo del software es un problema ingenieril apropiado, porque involucra "*la creación de soluciones rentables económicamente para problemas prácticos*" [Shaw y Tomayko, 1991].

Una vez hechas estas disquisiciones sobre el término y sus controversias, se va a proceder a exponer una muestra de las numerosas definiciones que de Ingeniería del Software se pueden encontrar en la bibliografía:

> "*Ingeniería del software es el establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en máquinas reales*"
>
> **Fritz Bauer**, *First NATO Software Engineering Conference*, Garmisch (Germany), 1968; en [Buxton et al., 1976][21]

[21] También en [Bauer, 1972].

"*Ingeniería del Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software*"

[Boehm, 1976]

"*Ingeniería del Software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas software*"

[Zelkovitz et al., 1979]

"*La aproximación sistemática al desarrollo, operación, mantenimiento y retirada del software*"

IEEE "*Standard Glossary of Software Engineering Terminology*"

[IEEE, 1983]

"*Es la disciplina tecnológica y de gestión que concierne a la producción y mantenimiento sistemático de productos software que son desarrollados y modificados a tiempo y dentro de los costes estimados*"

[Fairley, 1985]

"*Tratamiento sistemático de todas las fases del ciclo de vida del software. Se refiere a la aplicación de metodologías para el desarrollo del sistema software*"

Asociación Española para la Calidad [AECC, 1986]

"*Construcción de software multi-versión por un equipo de varias personas*"

[Parnas y Weiss, 1987]

"*La aplicación disciplinada de principios, métodos y herramientas de ingeniería, ciencia y matemáticas para la producción económica de software de calidad*"

[Humphrey, 1989]

"*La utilización de metodologías, herramientas y técnicas para resolver los problemas prácticos que surgen en la construcción, desarrollo, soporte y evolución del software*"

Institute for Information Technology, NRC Canada, 1990

"*Una disciplina cuyo objetivo es la producción de software de calidad, que se entrega en plazo, se ajusta al presupuesto y que satisface sus requisitos*"

Vanderbilt University, "Software Engineering"
Aksen Assoc., 1990

"*(1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento del software; es decir, la aplicación de la Ingeniería al Software.*

(2) El estudio de las aproximaciones en (1)"

IEEE Std 610.12-1990 [IEEE, 1999]

"*Disciplina tecnológica y de gestión concerniente a la invención, producción sistemática y mantenimiento de productos software de alta calidad, desarrollados a tiempo y al mínimo coste*"

[Frakes et al., 1991]

"*Las actividades sistemáticas implicadas en el diseño, implantación y prueba de software para optimizar su producción y soporte*"

Canadian Standards Association.
"*CSA Information Technology Vocabulary*"**, 1992**

"*Aplicación de herramientas, métodos y disciplinas para producir y mantener una solución automatizada de un problema real*"

[Blum, 1992]

"*Aquella forma de ingeniería que aplica principios de informática y matemática a la resolución de problemas software de forma eficiente en cuanto al coste*"

[Humphrey, 1993]

"*Aplicación de principios científicos para la transformación ordenada de un problema en una solución software funcional, así como en el consiguiente mantenimiento del software hasta el final de su vida útil*"

[Davis, 1993]

"*Es la aplicación de herramientas, métodos y disciplinas de forma eficiente en cuanto al coste, para producir y mantener una solución a un problema de procesamiento real automatizado parcial o totalmente por el software*"

[Horan, 1995]

> "*La aplicación de métodos y conocimiento científico para crear soluciones prácticas y rentables para el diseño, construcción, operación y mantenimiento del software y los productos asociados, al servicio de las personas*"
>
> Adaptado de la definición de Ingeniería de
> **Mary Shaw y David Garlan** en **[Shaw y Garlan, 1996]**

En lugar de con definiciones escuetas, la *Sociedad Británica de Informática* y el *Instituto de Ingeniería del Software* presentan su visión de lo qué es la Ingeniería del Software con descripciones más elaboradas. La primera de ellas es la siguiente [BCS, 1989]:

"*La Ingeniería del Software requiere la comprensión y aplicación de principios de ingeniería, habilidades de diseño, buenas prácticas de gestión, fundamentos de la Ciencia de la Computación y formalismos matemáticos. Es tarea de la Ingeniería del Software juntar estas áreas de trabajo tan dispares y utilizarlas en las fases de obtención de los requisitos, especificación, diseño, verificación, implementación, prueba, documentación y mantenimiento de sistemas software complejos y de gran tamaño. El ingeniero del software debe cumplir el papel del arquitecto del sistema complejo, tomando en cuenta las necesidades y requisitos del usuario, la viabilidad, el coste, la calidad, la confianza, la seguridad y las restricciones temporales. La necesidad de ajustar la importancia relativa de estos factores de acuerdo a la naturaleza del sistema y de su aplicación confiere una fuerte dimensión ética a las tareas del ingeniero del software, sobre quien pueda depender la seguridad y bienestar de otros, y para quien, como en medicina o en derecho, un sentido de moralidad profesional se requiere para su trabajo.*

El ingeniero del software debe ser capaz de estimar el coste y la duración del proceso de desarrollo del software, así como determinar la consecución de corrección y confianza. Tales medidas y estimaciones pueden involucrar conocimientos de conceptos financieros y de gestión, al mismo nivel que el manejo de los fundamentos matemáticos. Se necesita el uso preciso de las notaciones formales y de las palabras para expresarlas con el grado de precisión requerido a otros ingenieros y a clientes formados. En la mayoría de las circunstancias las hebras técnicas, teóricas y de gestión de un proyecto de Ingeniería del Software no pueden separarse unas de las otras.

Para construir grandes productos y conseguir una alta productividad, el ingeniero requiere el uso de herramientas software de desarrollo y de elementos reutilizables que garanticen su subsiguiente modificación y mantenimiento con seguridad.

La actividad profesional del ingeniero del software abarca el rango de tareas involucradas en el ciclo de vida de un sistema software. La obtención de requisitos, especificación, diseño, verificación y construcción son tareas críticas para conseguir la calidad del producto y son todas ellas responsabilidad del ingeniero del software.

Dado que el software determina el comportamiento de un autómata, el ingeniero del software necesita tener conocimientos de hardware digital y de comunicaciones. Aunque la Ingeniería del Software como disciplina puede ser calificada como independiente del área de aplicación, su realización debe ser en el contexto de aplicaciones específicas. El ingeniero del software debe, por tanto, ser capaz de colaborar con otros profesionales que le brindarán capacidades complementarias en la labor de especificar, diseñar y construir sistemas hardware-software que se ajusten a las necesidades del cliente, haga uso de las soluciones hardware y software en una óptima combinación y ofrezca una interfaz de usuario con una calidad adecuada.

La mayoría del software se construye en equipo, frecuentemente con equipos interdisciplinarios. La habilidad para trabajar cerca los unos de los otros es esencial.

Algunos de los métodos y de las herramientas intelectuales de la Ingeniería del Software están en proceso de desarrollo y se espera que tengan que cambiar de forma rápida. Los ingenieros del software, por tanto, necesitan tener unos buenos fundamentos teóricos que les sirvan de base par aprender y usar nuevos métodos en el futuro, y la mentalidad que les permita actualizar de forma permanente los conocimientos que necesitan para su labor profesional"

El Instituto de Ingeniería del Software propone la siguiente definición [Ford, 1990]:

1. Definición central:
    - *Ingeniería es la aplicación sistemática de conocimiento científico para la creación y construcción de soluciones rentables a problemas prácticos al servicio de la humanidad.*
    - *La Ingeniería del Software es la forma de ingeniería que aplica principios propios de la Ciencia de la Computación y Matemáticas para conseguir soluciones rentables a problemas software.*
2. Elaboraciones e interpretaciones:
    - *La creación y construcción del software debe incluir el mantenimiento. Debe cubrirse el ciclo de vida del software completo.*

- *La rentabilidad implica no sólo dinero, sino tiempo, calendario y recursos humanos. También implica obtener buenos valores por los recursos invertidos; incluyendo la calidad cuando las medidas se consideren oportunas.*
- *La Ingeniería del Software no se limita a aplicar sólo principios de la Ciencia de la Computación y las Matemáticas, sino cualquier principio del que pueda sacar ventaja.*
- *La Ingeniería del Software necesita contar con principios y técnicas de gestión para llevar a cabo sus actividades de desarrollo.*

3. Distinción entre el uso actual del término "Ingeniería del Software" y la definición que se adecua a la misión del SEI:
   - *Actualmente, el término "Ingeniería del Software" tiene múltiples conjuntos de significados conflictivos y pobremente entendidos, que van desde la programación a la gestión del diseño del sistema.*
   - *Actualmente, el término "Ingeniería del Software" es más una aspiración que una descripción.*

Parece claro que hay un consenso en que el desarrollo del software necesita una base rigurosa que encuentra en la Ingeniería, e indirectamente en la Ciencia y en las Matemáticas. Pero concretamente, en lo referente a la palabra *Ingeniería*, hay diferentes opiniones, así J. McDermid destaca los fundamentos de Ciencia y Matemáticas [McDermid, 1991]; R. S. Pressman, en clara alusión a F. Bauer, habla simplemente de principios de ingeniería, que pueden incorporar principios teóricos y prácticos [Pressman, 2000]; Ian Sommerville incluye "teorías, métodos y herramientas" [Sommerville, 2001], aunque indica que la Ingeniería del Software es diferente a otras formas de Ingeniería debido a la propia naturaleza del software [Beizer, 2000]; Hoare en la década de los setenta cita los componentes clave de la Ingeniería, que según él son lo suficientemente valiosos y relevantes como para ser imitados por los desarrolladores de software, concretamente identifica cuatro aspectos de la Ingeniería que cubren tanto los elementos teóricos como los prácticos de la Ingeniería del Software [Hoare, 1975]: profesionalismo, vigilancia, conocimiento teórico y herramientas; A. Wasserman sugiere que existen ocho nociones fundamentales en la Ingeniería del Software que forman la base para una disciplina efectiva [Wasserman, 1996]: abstracción, métodos y notaciones de análisis y diseño, prototipado de la interfaz de usuario, modularidad y arquitectura del software, proceso y ciclo de vida, reutilización, métricas, y herramientas y entornos integrados.

No sólo hace falta decir lo qué es la Ingeniería del Software, sino que es conveniente recalcar lo qué no es; así la Ingeniería del Software no es el diseño de programas que se implementan en otras áreas ingenieriles, ni es simplemente una forma de programar más

organizada que la que prevalece entre aficionados, principiantes o personas con falta de educación y entrenamiento específico.

Esta variedad de definiciones refleja las diferentes concepciones existentes sobre la Ingeniería del Software. A modo de ejemplo, esta diferencia de alcance y concepción de la Ingeniería del Software como disciplina se aprecia revisando los contenidos del libro de Ian Sommerville [Sommerville, 1985], donde la Ingeniería del Software está limitada al desarrollo de la programación mientras que el libro de Roger S. Pressman [Pressman, 1987] de esa misma época ya considera el análisis y diseño de sistemas. En su quinta edición Sommerville [Sommerville, 1996] amplía su concepto de Ingeniería del Software a aquellas actividades relacionadas con la especificación, desarrollo, gestión y evolución del software, no incorporando ningún capítulo específico sobre la práctica o los lenguajes de programación. En la sexta edición [Sommerville, 2001], última edición hasta la fecha, se ha reorganizado en siete partes, se ha hecho un mayor énfasis en la orientación a objetos, ahora los fragmentos de código se expresan en Java [SUN, 2002] y los modelos objeto en UML (*Unified Modeling Language*) [OMG, 2001b].

El concepto de Ingeniería del Software surge de la distinción entre programación de pequeños proyectos (*programming in the small*) y programación de grandes proyectos (*programming in the large*) y el reconocimiento de que la Ingeniería del Software está relacionada con esta última. Este primer concepto fue rápidamente ampliado para incorporar a la Ingeniería del Software todas aquellas tareas relacionadas con la automatización de los Sistemas de Información y con la Ingeniería de Sistemas en general.

El enfoque de la Ingeniería del Software que se adopta en este Proyecto Docente es el relacionado con los problemas que se presentan en el desarrollo de grandes sistemas software bajo la perspectiva de los sistemas de información a los que dan servicio. Esto viene motivado por el hecho de que aquellos enfoques de la Ingeniería del Software más relacionados con el diseño y la programación de módulos o componentes software están asignados a otras asignaturas de los Planes de Estudios como **Programación**, **Laboratorio de Programación**, **Estructuras de Datos** o **Programación Orientada a Objetos**.

4.3.2 Marco histórico de la Ingeniería del Software

El contexto en el que se han desarrollado las técnicas del software está íntimamente ligado a la evolución solapada de los sistemas informáticos y de la programación, cuyos hitos o avances más importantes han configurado las eras o etapas que se detallan a continuación. No se intenta hacer una clasificación cronológica exacta ya que los desarrollos han estado muy solapados e

incluso ideas que surgieron en una fecha determinada no serian aceptadas ampliamente quizá hasta 10 años después [Goldberg, 1986; Pressman, 1992].

Ninguna de estas eras puede decirse que hayan terminado formalmente, sobre todo si se hace caso del estudio [Redwine, 1985] que afirma que el tiempo necesario para aceptar una idea que implique un cambio tecnológico es de 15 a 20 años. Esta afirmación ha sido corroborada más recientemente por otros muchos autores, como Klaus Dittrich del grupo del conocido "*manifiesto de Atkinson*" [Atkinson et al., 1989].

Durante la **primera era** (1.950-1.96x), la programación de ordenadores se concibe más como un arte que como una técnica sistematizada y compleja. En esta época la importancia se centra en el hardware, sometido a un cambio continuo, considerando al software como un *añadido*. El software se desarrolla utilizando únicamente la intuición, con escasos métodos sistematizados y prácticamente sin ningún control en su desarrollo. La mayoría de los sistemas utilizaban una orientación *batch* (excepciones a esto son el sistema de reserva de *American Airlines* y los sistemas americanos de defensa en tiempo real, como SAGE). El ordenador estaba dedicado a la ejecución de un programa simple que a su vez resolvía una situación específica.

En la **segunda era** (1.96x-197x) la multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos en la interacción hombre-máquina. Los sistemas en tiempo real podían recoger, analizar y transformar datos procedentes de múltiples orígenes, controlando procesos y produciendo resultados en milisegundos en vez de en minutos. Los avances en las máquinas de memoria secundaria *on-line* llevaron a la primera generación de sistemas de gestión de bases de datos.

La segunda era también se caracterizó por el uso de *productos software* o paquetes de software y por el comienzo de las "*software house*" o "*casas de servicios*". El software empezó a ser desarrollado pensando en una distribución más amplia y para un mercado multidisciplinar.

Una de las primeras contribuciones a la Ingeniería del Software, en el sentido de mejorar la fiabilidad, dirección y productividad en el desarrollo del software, fue el artículo de E. W. Dijkstra "*Go to Statement Considered Harmful*", [Dijkstra, 1968a] que acentuaba los beneficios de utilizar los conceptos de la programación estructurada. Otra contribución importante de esta época fue la obra de Weinberg "*The Psychology of Computer Programming*" [Weinberg, 1971]; este clásico de la literatura del software introdujo las ideas de "*programación sin ego*", nombres mnemónicos de variables y en general la necesidad de la claridad y un buen estilo en la programación. Aunque quizás uno de los artículos de mayor influencia de la época fuera el de D. L. Parnas sobre la ocultación de la información [Parnas, 1972].

Según fue creciendo el número de sistemas basados en ordenador las bibliotecas de software se fueron extendiendo. Los proyectos produjeron cientos de miles de instrucciones fuente. Pero los problemas también empezaron a aparecer: todos estos sistemas informáticos y todas estas instrucciones fuente tenían que ser mantenidos para corregir errores "*oscuros*" (detectados tardíamente), adaptarse a los cambios en los requisitos de los usuarios o adaptarse al nuevo hardware que adquirían las organizaciones. El esfuerzo necesario para el mantenimiento de los sistemas informáticos, y sobre todo del software asociado, comenzó a absorber recursos de forma alarmante y, peor aún, la naturaleza personalizada y la ausencia o escasez de técnicas generales de diseño y análisis, hacía que muchos de estos sistemas fuesen prácticamente inmantenibles: había empezado una **crisis del software**, reconocida por primera vez oficialmente en la reunión de la OTAN de 1968 [Naur y Randell, 1969; Boehm, 1976; Goldberg, 1986; Randell, 1998].

En la **tercera era** (1.97x-1.98x) aparecen los sistemas distribuidos - varios computadores, realizando cada uno sus funciones concurrentemente y comunicándose unos con otros - dando lugar a un incremento importante de la complejidad de los sistemas informáticos. Esta época se caracteriza por el uso generalizado del microprocesador que, gracias al abaratamiento y aumento de potencia de éstos, permite la realización de funciones complejas a un coste excepcionalmente bajo.

Durante esta era, la crisis del software se acentuó, detectándose que aproximadamente el 50% de los presupuestos de los centros de procesamiento de datos se dedica a mantenimiento [Goldberg, 1986], de forma que la productividad en el desarrollo de nuevos sistemas se vio notablemente dañada. La intensificación de la crisis provoca como respuesta una mayor toma en consideración de la necesidad de un proceso de Ingeniería para el desarrollo del software. Como consecuencia de la importancia que adquiere la Ingeniería del Software hacen su aparición las primeras metodologías, destacando las propuestas por Jackson [Jackson, 1975], Warnier [Warnier, 1974] y DeMarco [DeMarco, 1979] por ser las que mayor difusión y utilización alcanzan. Por otra parte, este acercamiento del proceso de construcción del software a los procesos de ingeniería clásicos, conduce a la aplicación de técnicas de gestión de proyectos como PERT y CPM a los proyectos de desarrollo de software.

La consecuencia que se extrae de esta época es que es mejor utilizar alguna metodología disciplinada, no importa cual, que no utilizar ninguna [Basili, 1991].

Una **cuarta era** (1.98x-...) ha venido caracterizada por la introducción de sistemas de sobremesa, y la adopción de tecnologías y herramientas que proporcionan el soporte necesario para mejorar la calidad y la productividad en el desarrollo del software. Entre ellas se pueden

destacar: las herramientas CASE, los entornos de programación, el prototipado rápido (usado independientemente o con el ciclo de vida tradicional), la tecnología orientada a objetos, la reutilización sistemática del software, y los lenguajes de cuarta generación (4GL).

Es también en este período cuando empiezan a darse a conocer las aproximaciones formales al desarrollo de software a través de especificaciones algebraicas y lenguajes de especificación ejecutables [Goguen y Meseguer, 1988], como OBJ [Goguen et al., 1992] o ACT ONE [Ehrig y Mahr, 1985], técnicas y métodos orientados a modelos como Z [Spivey, 1989; Diller, 1990] o VDM (*Vienna Definition Method*) [Jones, 1991] y álgebras de procesos como CSP (*Communicating Sequential Processes*) [Hoare, 1985], aunque algunas de estas técnicas son muy anteriores (como VDM, desarrollado por los laboratorios de investigación de IBM de Viena, en 1973).

En los últimos años, han cobrado una gran importancia los modelos de proceso y la preocupación por la mejora en la calidad tanto del producto software como del proceso para mejorarlo. De estos modelos se puede destacar la norma ISO-9000, en su aplicación a la construcción de software [Layman, 1994; Schmauch, 1994], la iniciativa "BOOTSTRAP" [Lebsanft y Synspace, 1994], que consiste en un método para analizar y evaluar cuantitativamente determinados atributos de calidad del proceso (la evaluación permite conseguir un perfil detallado acerca de la calidad de la organización donde se aplica); el conocido modelo de madurez y capacidad de la Universidad Carnegie-Mellon (Pittsburg, PA - USA), CMM (*Capability Maturity Model*) [Paulk et al., 1993a; Paulk et al., 1993b] o el método SPICE (*Software Process Improvement & Capability Evaluation*) [Dorling, 1993; Konrad et al., 1995] que tiene como objetivo convertirse en un estándar ISO mundial que integre a las iniciativas anteriores.

Los sistemas basados en microprocesadores de 32 ó 64 bits, la computación paralela, el desarrollo de sistemas de Inteligencia Artificial (todavía hoy no excesivamente extendidos en el mercado) y las nuevas tecnologías (láser, fibra óptica...) en las comunicaciones (Internet), junto con herramientas de programación visual, desarrollo y ejecución de aplicaciones remotas, están conduciendo a **la transición hacia una quinta era**. En Ingeniería del Software se espera que esta nueva era venga marcada por el desarrollo y perfeccionamiento de los entornos de programación y herramientas integradas de apoyo a metodologías, por la continuación y mejora de las técnicas asociadas al prototipado [Davis, 1982; Gomaa, 1983; Davis, 1992] y a la reusabilidad del software [Marqués, 1995; García, 2000], por las técnicas de Ingeniería Web (*Web Engineering*) [Pressman, 2000, c.29; Ginige y Murugesan, 2001; IEEE-CS, 2001a; IEEE-CS, 2001b; WebE, 2001] y por la aplicación de técnicas de "Ingeniería del Conocimiento" al

desarrollo de software, especialmente como apoyo a la Ingeniería de Requisitos en la captura, estructuración y reutilización del conocimiento (requisitos) en dominios de aplicación [Sutcliffe y Maiden, 1998].

Estas eras tienen una correspondencia bastante inmediata en lo que podía denominarse la historia de la educación en Ingeniería del Software. Así, James E. Tomayko [Tomayko, 1998] identifica tres períodos significativos: la época de cursos de aislados y de libre contenido (antes de 1978), la época de los primeros programas para graduados (1978-1988), y por último una época de rápida proliferación de programas influidos por el trabajo realizado en el SEI (*Software Engineering Institute*).

4.3.3 Marco conceptual de la Ingeniería del Software

En este capítulo se ha señalado que el enfoque de la Ingeniería del Software que aquí se presenta es el relacionado con los Sistemas de Información a los que da servicio. Bajo esta perspectiva, un sistema software es una parte de un sistema mayor que lo engloba como componente. La Ingeniería del Software, por lo tanto, será solamente una parte del diseño del sistema en la que los requisitos del software han de ajustarse a los requisitos del resto de los elementos que constituyen este sistema. Por este motivo, el ingeniero del software ha de estar implicado en el desarrollo de los requisitos del sistema completo, comprendiendo el dominio de actividad en su totalidad.

Atendiendo a esta concepción de la Ingeniería del Software, es importante remarcar el papel que desempeña la Teoría General de Sistemas como antecedente conceptual en el que se apoya la teoría sobre los Sistemas de Información a los que la Ingeniería del Software intenta aportar soluciones. En el marco de la Teoría General de Sistemas, el análisis de sistemas tiene como objetivo general la comprensión de los sistemas complejos para abordar su modificación de forma que se mejore el funcionamiento interno para hacerlo más eficiente, para modificar sus metas... Las modificaciones pueden consistir en el desarrollo de un subsistema nuevo, en la agregación de nuevos componentes, en la incorporación de nuevas transformaciones... En general, el análisis de sistemas establece los siguientes pasos a seguir:

1. **Definición del problema**. En este paso se identifican los elementos de insatisfacción, los posibles cambios en las entradas y/o salidas al sistema y los objetivos del análisis del sistema.

2. **Comprensión y definición del sistema**. En este paso se identifica y descompone el sistema jerárquicamente en sus partes constituyentes o subsistemas junto con las relaciones existentes entre los mismos.
3. **Elaboración de alternativas**. En este paso se estudian las diferentes alternativas existentes para la modificación y mejora del sistema, atendiendo a los costes y perspectivas de realización.
4. **Elección de una de las alternativas definidas en el paso anterior.**
5. **Puesta en práctica de la solución elegida.**
6. **Evaluación del impacto de los cambios introducidos en el sistema.**

Muchas de las técnicas y métodos actuales de la Ingeniería del Software intentan dar respuesta a este tipo de cuestiones.

Los sistemas, de los que el software forma parte, se denominan sistemas informáticos o sistemas de computadora. En este sentido Roger S. Pressman [Pressman, 1997] considera que la Ingeniería del Software ocurre como consecuencia de un proceso denominado Ingeniería de Sistemas de Computadora. La Ingeniería de Sistemas de Computadora se concentra en el análisis, diseño y organización de los elementos en un sistema que pueden ser un producto, un servicio o una tecnología para la transformación de información o el control de información. De igual forma, este autor denomina al *Proceso de Ingeniería del Software* como **Ingeniería de la Información** cuando el contexto de trabajo de Ingeniería se enfoca a una empresa, y lo denomina **Ingeniería de Producto** cuando el objetivo es construir un producto. El término genérico de Ingeniería de Sistemas es el que utiliza para unificar estos dos tipos de ingenierías. La Ingeniería de Sistemas establece, por lo tanto, el papel que ha de asignarse al software y los enlaces que unen al software con otros elementos de un sistema basado en computadora.

Desde una perspectiva más general, J. L. Le Moigne [Le Moigne, 1973] concibe los sistemas formados por tres subsistemas interrelacionados: *el de decisión, el de información* y *el físico*. El sistema de decisión procede a la regulación y control del sistema físico decidiendo su comportamiento en función de los objetivos marcados. El sistema físico transforma un flujo físico de entradas en un flujo físico de salidas. En interconexión entre el sistema físico y el sistema de gestión se encuentra el sistema de información. El sistema de información está compuesto por diversos elementos encargados de almacenar y tratar las informaciones relativas al sistema físico a fin de ponerlas a disposición del sistema de gestión. El Sistema Automatizado de Información (SAI) es un subsistema del sistema de información en el que todas las

transformaciones significativas de información son efectuadas por máquinas de tratamiento automático de las informaciones.

Basándose en las ideas anteriores, se considera a la Ingeniería del Software como la disciplina que se ocupa de las actividades relacionadas con los sistemas informáticos o sistemas de información en los que el software desempeña un papel relevante. Estos sistemas de información han de ser fiables, es decir, que su realización se lleve a cabo de forma correcta conforme a unos estándares de calidad y, además, que su desarrollo se realice en el tiempo y coste establecidos. Este último aspecto es crucial, y existe una gran variedad de informes, publicaciones y datos que avalan la gran dependencia que las organizaciones tienen hoy en día de los sistemas software.

El término software de calidad aparece reiteradamente como un fin de la Ingeniería del Software. Una valoración general de la calidad de un sistema software requiere la identificación de atributos comunes que pueden esperarse de un buen producto de ingeniería. Asumiendo que el software proporciona la funcionalidad requerida, hay una serie de atributos que se deberían encontrar. Según I. Sommerville [Sommerville, 2001] existe un amplio rango de potenciales atributos de calidad del software a considerar (seguridad, protección, fiabilidad, flexibilidad, robustez, comprensión, experimentación, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia y aprendizaje), no siendo posible, en general, optimizarlos todos para un sistema software, por lo que una parte importante de la planificación de calidad es la selección de los atributos de calidad de mayor importancia, así como el establecimiento de las vías para alcanzarlos. Por su parte Mynatt [Mynatt, 1990] establece que la calidad de un producto software se mide en función de determinados aspectos que, básicamente, son distintos para el promotor, los usuarios y los encargados de mantenimiento. El promotor o cliente del proyecto, como entidad encargada de los costes, exigirá que el producto software aumente la productividad de su organización, tenga un bajo coste, sea eficiente, fiable y flexible. Los usuarios exigirán que el sistema les proporcione la funcionalidad adecuada, pero también fiabilidad, eficiencia, y que sea fácil de aprender, de usar y de recordar. Por último, a los encargados del mantenimiento les interesará que el sistema contenga el menor número de errores cuando se instale por primera vez, que tenga una buena documentación, que sea fiable y que esté bien diseñado. Por su parte, Bertrand Meyer [Meyer, 1997] distingue entre factores de calidad externos (tales como corrección, robustez, extensibilidad, reutilización, compatibilidad, eficiencia, portabilidad, facilidad de uso, funcionalidad y oportunidad) e internos (modularidad, legibilidad...), si bien los factores externos son los que importan en última instancia, debido a que son los únicos percibidos por el usuario, son los factores internos los que aseguran el cumplimiento de los primeros. En cualquier caso, la optimización de todos estos aspectos es

difícil ya que algunos son excluyentes. Además, la relación entre los costes y estos atributos no es lineal, y pequeñas mejoras o variaciones en cualquiera de ellos pueden ser demasiado caras. Boehm [Boehm, 1981] afirma que aplicar con éxito la Ingeniería del Software requiere una continua resolución de una variedad de metas importantes, pero conflictivas en la mayoría de sus casos.

En muchas ocasiones la Ingeniería del Software se ha querido limitar al desarrollo de grandes proyectos informáticos. Sin embargo, tal y como afirma Barry B. Boehm "*se hacen planos para una casa tanto si esta es grande como si es pequeña*". La filosofía que subyace en esta frase, es que cualquier desarrollo de software ha de seguir un proceso. Como afirma Roger S. Pressman [Pressman, 1997]: "*El fundamento de la Ingeniería del Software es la capa proceso. El proceso de la Ingeniería del Software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la Ingeniería del Software*". El proceso define un marco de trabajo en el que se establece el control de gestión de los proyectos software y el contexto en el que se aplican los métodos técnicos y se producen los resultados del trabajo.

En el proceso de construcción de sistemas informáticos se pueden distinguir dos fases genéricas, independientemente del paradigma de ingeniería elegido: ***la definición*** y ***el desarrollo***.

Durante la fase de *definición* se identifican los requisitos claves del sistema y del software. Durante la misma se desarrollan un **Análisis de Sistemas**, en el que se define el papel de cada elemento en el sistema automatizado de información, incluyendo el que jugará el software, y un **Análisis de Requisitos** en el que se especifican todos los requisitos de usuario que el sistema tiene que satisfacer. Esta fase está orientada al **QUÉ**: *qué información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, qué ligaduras de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto.* En la fase de definición existe un paso complementario que consiste en la planificación del proyecto software, en el que se asignan los recursos, se estiman los costes y se planifican las tareas y el trabajo.

Por el contrario la fase de *desarrollo* está orientada al **CÓMO**, y el primer paso de esta fase corresponde al **Diseño del Software**. En el diseño del software *se trasladan los requisitos del software a un conjunto de representaciones que describen la estructura de datos, arquitectura del software y procedimientos algorítmicos y que permiten la construcción física de dicho software*. Los otros dos pasos de la fase de desarrollo corresponden a la **Codificación** y a la **Prueba del Software**.

Además de las fases de definición y desarrollo del software, existe una tercera fase dentro de la construcción de los sistemas que corresponde a la fase de **Mantenimiento** de los mismos. Esta fase se enfoca *a los cambios asociados con la corrección de errores, con las adaptaciones requeridas por la evolución del entorno del software y las modificaciones debidas a los cambios de requisitos del usuario para mejorar el sistema*.

La fase de definición es crucial en el desarrollo de un sistema software pues en ella quedan establecidas y determinadas explícitamente las necesidades y limitaciones del usuario y del sistema. La especificación de requisitos ha de realizarse antes de que la construcción del sistema de comienzo, pues de lo contrario podría ocurrir que las necesidades se simplifiquen completamente, las limitaciones se olviden o las interdependencias se pasen por alto durante la fase de desarrollo.

Para la comprensión y validación del sistema en estudio, se elaboran modelos. Estos modelos han de separar las especificaciones conceptuales y lógicas (¿qué ha de cumplir el sistema?), de las especificaciones físicas, (¿cómo lo hará dependiendo de los recursos hardware y software?). Un concepto esencial en el desarrollo de este proceso es el de "*nivel de abstracción*". Aplicando este concepto, el desarrollo de un sistema de información consiste en definir una jerarquía apropiada de niveles de abstracción. Cada nivel produce un modelo del sistema que se describe mediante un lenguaje apropiado. El desarrollo comienza con niveles de abstracción altos, poco detallados, y termina con los de máximo detalle que sirven como base para la construcción directa del sistema ejecutable.

Una última consideración a tener en cuenta en el proceso de construcción de sistemas informáticos, es que la operatividad del producto final depende en gran medida de su conocimiento. Esto se consigue con la elaboración detallada y precisa de la documentación de apoyo: *documentación de sistema, manual de usuario, instrucciones de instalación, guías de entrenamiento, manual de operación...*

El uso de una **metodología** permite el dominio del proceso descrito, definición, desarrollo, implementación y mantenimiento, lo que asegurará el éxito de los proyectos informáticos. En general, una metodología es "*el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*" [DRAE, 1995]. Se puede decir que una metodología es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería del Software. A su vez, un método, es un procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla. Es un conjunto de técnicas, herramientas y tareas que, de acuerdo a un enfoque metodológico, se aplican para la resolución de un problema.

Desde el punto de vista específico de la Ingeniería del Software, la metodología describe como se organiza un proyecto, el orden en el que la mayoría de las actividades tienen que realizarse y los enlaces entre ellas, indicando asimismo cómo tienen que realizarse algunas tareas proporcionando las herramientas concretas e intelectuales. En concreto, se puede definir **metodología de Ingeniería del Software** como "*un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas*" [Rumbaugh et al, 1991].

En la actualidad se pueden distinguir seis escuelas principales de pensamiento en relación con las técnicas y métodos de desarrollo de Ingeniería del Software:

1. **Orientadas a procesos**: Si se parte de que la Ingeniería del Software se fundamenta en el modelo básico **entrada/proceso/salida** de un sistema[22]; de forma que los datos se introducen en el sistema y éste responde ante ellos transformándolos para obtener salidas. Estas metodologías se enfocan fundamentalmente en la parte de proceso y, por esto, se describen como un enfoque de desarrollo de software orientado al proceso. Utilizan un enfoque de descomposición descendente para evaluar los procesos del espacio del problema y los flujos de datos con los que están conectados. Este tipo de metodologías se desarrolló a lo largo de los años 70. Los creadores de este tipo de métodos fueron Edward Yourdon y Larry Constantine [Yourdon y Constantine, 1975; Yourdon and Constantine, 1979; Yourdon and Constantine, 1989; Yourdon, 1989]; Tom DeMarco [DeMarco, 1979]; Gane y Sarson [Gane y Sarson, 1977; Gane y Sarson, 1979]. Representantes de éste grupo son las metodologías de análisis y diseño estructurado como **Merise** [Tardieu et al., 1986], **YSM** (*Yourdon Systems Method*) [Yourdon Inc., 1993], **SSADM** (*Structured Systems Analysis and Design Method*) [Ashworth y Goodland, 1990] o **METRICA v.2.1** [MAP, 1995] y en parte **METRICA v3.0** [MAP, 2001a; MAP, 2001b].

2. **Orientadas a datos**: Al contrario que en el caso anterior, estas metodologías se centran más la parte de **entrada/salida** dentro del modelo básico **entrada/proceso/salida.** En estas metodologías las actividades de análisis comienzan evaluando en primer lugar los datos y sus interrelaciones para determinar la arquitectura de datos subyacente. Cuando esta arquitectura está definida, se definen las salidas a producir y los procesos y entradas necesarios para

[22] Este modelo básico es utilizado por todas las metodologías estructuradas.

obtenerlas. Ejemplos representativos de este grupo son los métodos **JSP** (*Jackson Structured Programming*) y **JSD** (*Jackson Structured Design*) [Jackson, 1975; Jackson, 1983; Cameron, 1989], la construcción lógica de programas **LCP** (*Logical Construction Program*) de [Warnier, 1974] y el **DESD** (Desarrollo de Sistemas Estructurados de Datos), también conocido como metodología **Warnier-Orr**, [Orr, 1977; Orr, 1981].

3. **Orientadas a estados y transiciones**: Estas metodologías están dirigidas a la especificación de sistemas en tiempo real y sistemas que tienen que reaccionar continuamente a estímulos internos y externos (eventos o sucesos). Las extensiones de las metodologías de análisis y diseño estructurado de Ward y Mellor [Ward y Mellor, 1985] y de Hatley y Pirbhai [Hatley y Pirbhai, 1987] son dos buenos ejemplos de estas metodologías.

4. **Diseño basado en el conocimiento**: Es una aproximación que se encuentra aún en una fase temprana de desarrollo. Utiliza técnicas y conceptos de Inteligencia Artificial para especificar y generar sistemas de información. El método **KADS** (*Knowledge Acquisition and Development Systems*) [Wielinga et al. 1991] y la metodología **IDEAL** [Gómez et al., 1998] son ejemplos de esta categoría.

5. **Orientadas a objetos**: Estas metodologías se fundamentan en la integración de los dos aspectos de los sistemas de información: datos y procesos. En este paradigma un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes. El objeto encapsula datos y operaciones. Este enfoque permite un modelado más natural del mundo real y facilita enormemente la reusabilidad. Algunos representantes de este grupo son las metodologías **OOA/D** de Grady Booch [Booch, 1994], **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991; Rumbaugh, 1996], **OOSE** (*Object Oriented Software Engineering*) [Jacobson et al., 1993], **FUSION** propuesta por [Coleman et al., 1994], **MOSES** de [Henderson-Sellers y Edwards, 1994a; Henderson-Sellers y Edwards, 1994b], **UP** (*Unified Process*) [Jacobson et al., 1999] o, en parte, **METRICA v3.0** [MAP, 2001a; MAP, 2001b].

6. **Basadas en métodos formales**: Estas metodologías implican una revolución en los procedimientos de desarrollo, ya que a diferencia de todas las anteriores, estas técnicas se basan en teorías matemáticas que permiten una verdadera aproximación científica y rigurosa al desarrollo de sistemas de información y software asociado. Ejemplo de este tipo de metodologías puede ser **OO-Method** [Pastor et al., 1997],

que está basada en el lenguaje de especificación formal OASIS (*Open and Active Specification of Information Systems*) [Pastor y Ramos, 1995; Letelier et al., 1998; Sánchez et al., 1998].

Existen otras clasificaciones de las metodologías, por ejemplo en [Piattini et al., 1996] éstas se clasifican en función de tres parámetros *el enfoque*, *el tipo de sistema* y *la formalidad*.

4.3.4 Ingeniería del Software y Orientación a Objetos

La Orientación a Objetos forma parte de la Ingeniería del Software como un paradigma de desarrollo con su propio marco conceptual que involucra terminología, métodos, modelos, procedimientos, técnicas, prácticas y procesos.

Los métodos de la Orientación a Objetos han sido reconocidos en el ámbito de las tecnologías de la información, como la mejor filosofía para abordar la reutilización y la extensibilidad del software.

En una primera aproximación se puede decir que el término *orientado a objetos* significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como un comportamiento. Esto se opone frontalmente a la programación convencional, donde las estructuras de datos y la funcionalidad sólo se relacionan débilmente.

Tradicionalmente se asocia la Orientación a Objetos con la Programación Orientada a Objetos, es decir centrando la atención en los lenguajes de programación. Cierto es que los lenguajes de programación orientados a objetos son útiles para eliminar algunas restricciones propias de los lenguajes de programación tradicionales. Sin embargo, enfatizar la fase de codificación supone un retroceso de la Ingeniería del Software al priorizar los mecanismos de implementación frente al proceso de pensamiento subyacente al cual sirven de base [Rumbaugh et al., 1991]. Así, este paradigma se extiende a todas las fases del ciclo de vida con un modelo común subyacente a todas estas fases: *el modelo objeto*.

Se entiende, entonces, por *desarrollo orientado a objetos* la forma de pensar acerca del software basándose en abstracciones del mundo real; donde la palabra desarrollo hace alusión directa al bloque inicial de fases del ciclo de vida del software: *análisis, diseño* e *implementación.*

Aparecen así los conceptos de **Programación Orientada a Objetos** (POO), **Diseño Orientado a Objetos** (DOO) y **Análisis Orientado a Objetos** (AOO).

> "*La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia*"
>
> **Grady Booch**, [Booch, 1994]

Cabe destacar tres partes importantes en esta definición:

1. La POO utiliza *objetos*, no algoritmos, como sus bloques lógicos de construcción fundamentales.
2. Cada objeto es una *instancia* de alguna *clase*.
3. Las clases están relacionadas con otras clases por medio de relaciones de *herencia*.

Mientras que los métodos de programación ponen el énfasis en el uso correcto y efectivo de mecanismos particulares del lenguaje de programación que se utiliza, los métodos de diseño enfatizan la estructuración correcta y efectiva de un sistema complejo, definiéndose como sigue:

> "*El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para descubrir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña*"
>
> **Grady Booch**, [Booch, 1994]

Se destacan dos aspectos de esta definición:

1. El DOO da lugar a una descomposición orientada a objetos.
2. Utiliza diversas notaciones para expresar diferentes modelos del diseño lógico (*estructura de clases y objetos*) y físico (*arquitectura de módulos y procesos*) de un sistema, además de aspectos estáticos y dinámicos del sistema.

El modelo de objetos ha influido en las fases iniciales del ciclo de vida del desarrollo del software. El análisis orientado a objetos enfatiza la construcción de modelos del mundo real, utilizando una visión del mundo orientada a objetos, pudiéndose definir como:

> "*El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de las clases y los objetos que se encuentran en el vocabulario del dominio del problema*"
>
> **Grady Booch**, [Booch, 1994]

Toda la Orientación a Objetos gira en torno a dos conceptos fundamentales: *el objeto* y *la clase*. Ambos son dos conceptos estrechamente relacionados, pero que no deben confundirse nunca [Holland et al., 1997]. Para terminar esta introducción se van a presentar algunas de las definiciones que, tanto de objeto como de clase, pueden encontrarse en la bibliografía especializada.

Un objeto modela una parte de la realidad. Con el concepto de objeto, se modela la permanencia e identidad de conceptos percibidos. Así un objeto puede definirse como:

> "*Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema*"
>
> **[Smith and Tockey, 1988]**

> "*Una colección de operaciones que comparten un estado*"
>
> **Peter Wegner,** [Wegner, 1990]

> "*Concepto, abstracción, o cosa con frontera y significado débil, perteneciente al problema que se trata; instancia de una clase*"
>
> **[Rumbaugh et al., 1991]**

> "*Un objeto es un encapsulado de un conjunto de operaciones o métodos que pueden ser invocados externamente y de un estado que puede recordar los efectos de los métodos*"
>
> **[Blair et al., 1991]**

> "*Entidad conceptual que es identificable, tiene características que comparten un estado interno y tiene unas operaciones que pueden cambiar el estado del sistema local, y que también pueden solicitar operaciones de objetos relacionados*"
>
> **[Champeaux et al., 1993]**

> "Un objeto tiene estado, comportamiento e identidad; la estructura y el comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables"
>
> **Grady Booch,** [Booch, 1994]

> "*Un concepto, abstracción o cosa que puede ser individualmente identificada y tiene significado en una aplicación. Un objeto es una instancia de una clase*"
>
> **[Blaha y Premerlani, 1998]**

> "*Una entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento es representado por sus operaciones, métodos y máquinas de estados. Un objeto es una instancia de una clase*"
>
> **[OMG, 2001b]**

Por su parte las clases sirven como plantillas para la creación de objetos, especificando un comportamiento a todas sus instancias. Se puede definir como:

> "*Una clase es una plantilla desde la que sus objetos pueden ser creados. Contiene la definición de los descriptores de estado y los métodos de los objetos*"
>
> **[Blair et al., 1991]**

> "*Es un conjunto de objetos que comparten una estructura común y un comportamiento común*"
>
> **Grady Booch,** [Booch, 1994]

> "*Descripción abstracta de los datos y del comportamiento de una colección de objetos similares*"
>
> **Timothy Budd**, [Budd, 1991]

> "*Descripción de un grupo de objetos con propiedades similares, comportamientos comunes, interrelaciones comunes y semántica común*"
>
> **[Rumbaugh et al., 1991]**

> "*Una clase es un tipo abstracto de datos equipado con una posible implementación*"
>
> **Bertrand Meyer,** [Meyer, 1997]

4.3.4.1 Marco histórico de la Orientación a Objetos

Los pasos por los que ha pasado la evolución histórica de la Orientación a Objeto han sido similares a los de otros métodos de desarrollo; esto es, en primer lugar el énfasis estuvo centrado en las técnicas de programación, para posteriormente ir centrando la atención en las

fases de desarrollo de mayor nivel de abstracción, diseño y análisis, y actualmente en los procesos de negocio [Pancake, 1995].

Lo que no ha sido tan normal ha sido su evolución temporal, pues tras los primeros pasos dados a finales de la década de los sesenta, sufrió una parada significativa en su evolución hasta la segunda mitad de la década de los ochenta[23], que resurge con tremenda fuerza, especialmente a partir de la primera conferencia OOPSLA en 1986, dando lugar a una gran diversidad, que podría calificarse de caótica en algunas parcelas (como por ejemplo los métodos de análisis y diseño). En los últimos años de la década de los noventa se produjo una cierta madurez en la tecnología de objetos, caminando hacia la unificación y los primeros estándares.

El término objeto surge de forma independiente en varios campos de la informática, casi simultáneamente a principios de los setenta, para referirse a nociones que eran diferentes en su apariencia pero relacionadas entre sí. Todas estas nociones se inventaron para manejar la complejidad de los sistemas software de tal forma que los objetos representaban componentes de un sistema descompuesto modularmente o bien unidades modulares de representación del conocimiento [Yonezawa y Tokoro, 1987].

Levy añade que los siguientes acontecimientos contribuyeron a la evolución de los conceptos orientados a objetos [Levy, 1984]:

- Avances en la arquitectura de los computadores, incluyendo los sistemas de capacidades y el apoyo en hardware para conceptos de sistemas operativos.
- Avances en los lenguajes de programación, como se demostró en Simula [Dahl et al., 1970; Dahl y Hoare, 1972; Birtwistle et al., 1973; SIS, 1987], Smalltalk [Goldberg y Robson, 1983; Goldberg, 1985; Lalonde y Pugh, 1990; Lalonde y Pugh, 1990], CLU [Liskov, 1993] y Ada [Ada95-Web].
- Avances en metodología de la programación, incluyendo la modularización y la ocultación de la información [Parnas, 1972].

A esta lista de contribuciones podrían añadirse las tres siguientes [Booch, 1994]:

- Avances en los modelos de bases datos.
- Investigación en Inteligencia Artificial.
- Avances en Filosofía y Ciencia Cognitiva.

El concepto de un objeto tuvo sus inicios en el hardware con la aparición de arquitecturas basadas en descriptores y, posteriormente, arquitecturas basadas en capacidades [Ramamoorthy y Sheu, 1988]. Estas arquitecturas representaron una ruptura con las arquitecturas clásicas de Von Neumann, surgiendo como consecuencia de los intentos realizados para eliminar el hueco existente entre las abstracciones de alto nivel de los lenguajes de programación y las abstracciones de bajo nivel de la propia máquina. Según sus inventores estas arquitecturas ofrecen ventajas del tipo: mejor detección de errores, mejora de la eficiencia de ejecución, menos tipos de instrucciones, compilación más sencilla y reducción de los requisitos de almacenamiento. Algunos computadores con arquitectura orientada a objetos fueron el Burroughs 5000, el Plessey 250, el Intel 432, el IBM System/38 o el Rational R10000.

Muy relacionados con las arquitecturas orientadas a objetos están los sistemas operativos orientados a objetos. El trabajo de Dijkstra con el sistema de multiprogramación **THE** fue el primero que introdujo la idea de construir sistemas como máquinas de estados en capas [Dijkstra, 1968b]. Otros sistemas operativos pioneros en la tecnología de objetos fueron **UCLA Secure UNIX** (para el PDP 11/45 y 11/70), **StarOS** y **Medusa** (para Cm* de CMU) o el **iMAX** (para el Intel 432). Sistemas operativos actuales, como **Microsoft Windows NT**, parecen seguir el camino de los objetos.

Sin duda alguna la contribución más importante al modelo de objetos estriba en los denominados lenguajes de programación basados en objetos y orientados a objetos. Las ideas fundamentales de clase y objeto aparecieron por primera vez en el lenguaje Simula 67 [SIS, 1987].

En 1972 David L. Parnas introduce la idea de ocultación de la información [Parnas, 1972], y en la década de los setenta varios investigadores, destacando Liskov y Zilles [Liskov y Zilles, 1977], Guttag [Guttag, 1980] y Mary Shaw [Shaw, 1984], fueron pioneros en el desarrollo de mecanismos de tipos de datos abstractos. Hoare contribuyó a esos desarrollos con su propuesta de una teoría de tipos y subtipos.

Aunque la tecnología de Bases de Datos ha evolucionado un tanto independientemente de la Ingeniería del Software, también ha contribuido al modelo de objetos [Atkinson y Buneman, 1987], especialmente mediante las ideas de la aproximación entidad-relación al modelado de datos [Rumbaugh, 1987; Rumbaugh, 1988]. En el modelo entidad-relación, propuesto

[23] Como dato significativo, entre junio de 1978 y diciembre de 1985, sólo aparecen nueve artículos – de unos cuatrocientos – en ACM SIGPLAN Notices, mencionando la tecnología orientada a objetos de una forma significativa [Sharp et al., 2000].

inicialmente por Peter Chen [Chen, 1976], el mundo se modela en términos de sus entidades, los atributos de éstas y las relaciones entre esas entidades.

En el campo de la Inteligencia Artificial, los avances en representación del conocimiento han contribuido a una comprensión de las abstracciones orientadas a objetos. En 1975, M. Minsky propuso por primera vez una teoría de marcos para representar objetos del mundo real tal y como los perciben los sistemas de reconocimiento de imágenes y el lenguaje natural [Barr y Feigenbaum, 1981]. Desde entonces se han utilizado los marcos como fundamento arquitectónico para diversos sistemas inteligentes.

Por último la Filosofía y la Ciencia Cognitiva han contribuido al avance del modelo de objetos. La idea de que el mundo podía verse en términos de objetos o procesos procede de los filósofos griegos, más concretamente de la Teoría de las Ideas desarrollada por Platón, a partir de las enseñanzas de Sócrates, y estudiada y ampliada por Aristóteles (en la Tabla 4.1 se tiene una equivalencia entre los conceptos de la Teoría de las Ideas y la Orientación a Objetos, para una mayor información consultar [Alhir, 1998; Rayside y Campbell, 2000]). Así se puede decir que la Orientación a Objetos se acerca más al enfoque Aristotélico-Tomista[24], frente al enfoque Kantiano de los métodos estructurados. Asimismo, en el siglo XVII se tiene a Descartes defendiendo que los seres humanos aplican de forma natural una visión orientada a objetos del mundo [Stillings et al., 1987]. Ya en el siglo XX, Rand amplía estos conceptos en su filosofía de la epistemología objetivista [Rand, 1979].

TEORÍA DE LAS IDEAS	PARADIGMA OBJETUAL
Método Socrático (la recolección)	Abstracción (como forma de obtener conocimiento)
Ideas, universalidad y sustancias secundarias	Clases (y encapsulación)
Cosas, particulares y sustancias primarias	Objetos (y encapsulación)
Procedimiento de captura y división y el principio de uno sobre muchos	Herencia (y polimorfismo)
El argumento del tercer hombre	Abstracción (como marco conceptual para el modelado que involucra múltiples niveles de abstracción)

Tabla 4.1. Correspondencia entre la Teoría de las Ideas y la Orientación a Objetos

[24] En la realidad no existen datos o procesos independientes. Santo Tomás de Aquino sostiene que el número, considerado abstractamente, no existe fuera de la mente humana; "*la verdad consiste en la adecuación del entendimiento con las cosas*" (Suma Teológica. I, c.16, a.3.).

Aristóteles indica que "*no se pueden separar, a los objetos en movimiento, por ejemplo*" puesto que "*hay una multitud de accidentes que son esenciales a las cosas, en tanto que cada uno de ellos reside esencialmente en ellas*" (Metafísica. 7ª edición, Madrid, Espasa-Calpe, 1972, p.284.).

Al igual que sucede en el paradigma estructurado, la tecnología de objetos debe dar cobertura a todo el ciclo de desarrollo de los sistemas software, es decir, al análisis, al diseño y a la implementación; aunque se debe resaltar que estas fases se solapan y presentan unas formas más difuminadas que en el desarrollo estructurado.

Cuando a mediados de la década de los ochenta resurge la Orientación a Objetos, comienzan a surgir multitud de propuestas metodológicas con una orientación objetual, sólo en el período comprendido entre 1989 y 1994 el número de lenguajes de modelado orientados a objetos pasa de 10 a más de 50. Precisamente esta diversidad de métodos y notaciones ha sido una de las mayores barreras con que se encontraron las empresas y los departamentos de informática para la adopción de la Orientación a Objetos, conociéndose a este fenómeno como la guerra de los métodos [García y Pardo, 1998].

La cantidad de métodos que surgen en esta primera *hornada* se denominan metodologías o métodos de primera generación entre los que cabe destacar **Synthesis** [Bailin, 1989]**, RDD** (*Responsibility Driven Design*) [Wirfs-Brock et al., 1990], **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991], **OOD** (*Object Oriented Design*) [Booch, 1991], **OOSE** (*Object-Oriented Software Engineering*) – **Objectory** [Jacobson et al., 1993], o la metodología de **Shlaer y Mellor** [Shlaer and Mellor, 1992].

Hacia la mitad de década de los noventa aparecen en escena las nuevas versiones de los métodos más consolidados, así como nuevas incorporaciones que surgen como una acumulación de las características más destacadas de los métodos tradicionales junto con algunos añadidos: las metodologías o métodos de segunda generación. Pertenecientes a esta generación se tiene, entre otros, a **OMT-2** [Rumbaugh, 1996], **OOram** [Reenskaug et al., 1996], **OOA/D** (*Object-Oriented Analysis and Design*) [Booch, 1994], **Fusion** [Coleman et al., 1994]**, Moses** (*Methodology for Object-Oriented Software Engineering of Systems*) [Henderson-Sellers y Edwards, 1994a; Henderson-Sellers y Edwards, 1994b]**, Syntropy** [Cook y Daniels, 1994] o **Medea** [Piattini, 1994].

Ante esta situación de diversidad, en la parte final de la década de los noventa se empiezan a escuchar los primeros rumores sobre la necesidad de una unificación y una estandarización. Esto ha dado lugar a una tercera generación de metodologías, de entre las que destacan **UP** (*Unified Process*) [Jacobson et al., 1999], **OPEN** (*Object-Oriented Process, Environment and Notation*) [Graham et al., 1997; Henderson-Sellers et al., 1998; Firesmith et al., 1998] o **Catalysis** [D'Souza and Wills, 1999].

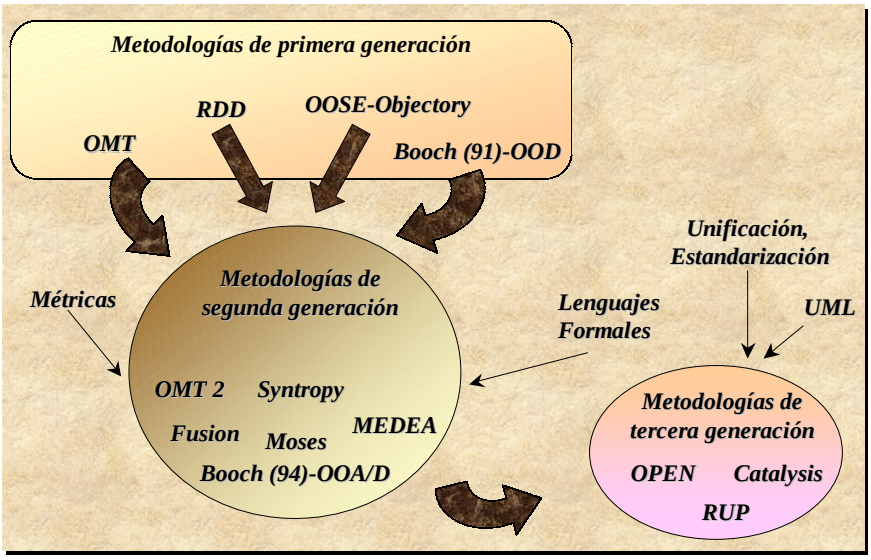


Figura 4.1. Generaciones de las metodologías orientadas a objetos

En la Figura 4.1 se presenta un esquema de la evolución sufrida por las metodologías y métodos de diseño orientados al objeto.

El catalizador principal de esta unificación ha sido sin lugar la aparición en escena del lenguaje de modelado UML (*Unified Modeling Language*) de Rational Software Corporation, que fue adoptado el 17 de noviembre de 1997 como lenguaje de modelado estándar por el OMG (*Object Management Group*) en su versión 1.1 [Rational et al., 1997]. OMG ha propuesto la especificación de UML para su estandarización internacional por parte de ISO (*International Organization for Standardization*).

UML se desarrolla en el seno de Rational Software Corporation con el apoyo de diversos colaboradores a lo largo de su historia, convirtiéndose en el elemento unificador de los lenguajes de modelado de los métodos **OOA/D** de Grady Booch [Booch, 94], **OMT-2** de James Rumbaugh [Rumbaugh, 1996] y **OOSE** de Ivar Jacobson [Jacobson et al., 1993].

Grady Booch, director científico de Rational Software Corporation desde prácticamente su creación en 1980, empieza a planificar su estrategia a favor de la unificación de métodos. Así, para la comunidad de los métodos orientados al objeto la gran noticia en el OOPSLA'94 fue que James Rumbaugh había abandonado General Electric para unirse a Grady Booch en Rational Software Corporation para fusionar sus métodos. De esta manera el desarrollo de UML comienza en octubre de 1994, cuando Booch y Rumbaugh empiezan a trabajar para unificar los dos métodos que más repercusión habían alcanzado en la escena del ADOO, OOA/D y OMT. Como primer fruto de esta colaboración aparece en octubre de 1995 la primera versión pública de la descripción de la unión de sus métodos. Esta versión se presenta en el OOPSLA'95 con el nombre de Método Unificado versión 0.8 [Booch y Rumbaugh, 1995].

El siguiente paso en el proceso de unificación se produce a finales de 1995 cuando Rational compra a la empresa Objectory, y su fundador, el prestigioso Ivar Jacobson, se une a Rational como vicepresidente de Ingeniería de Negocio para acoplar su metodología OOSE al método unificado de Booch y Rumbaugh. Tras la incorporación de Jacobson a Rational, se les conoce a Booch, Rumbaugh y Jacobson por *los tres amigos*.

El primer paso que se da después de la incorporación de Jacobson es la creación de un lenguaje de modelado unificado debido principalmente a dos razones. En primer lugar el lenguaje de modelado permite que los tres métodos puedan evolucionar hacia un punto común de forma independiente. Por otro lado, la unificación de la semántica y de la notación les permite colocarse en un lugar de privilegio en el mercado de los métodos orientados al objeto. Debe tenerse en cuenta que la mayoría de la gente que dice utilizar un método orientado al objeto realmente lo que usa es la notación asociada a dicho método, olvidándose de los procesos asociados al método, así pues, contar con una notación gráfica universal era fundamental para la unificación de sus métodos, y de hecho la notación de UML se convierte en el estándar de facto de las notaciones en tecnología de objetos antes de su aceptación como estándar oficial.

La primera versión de este lenguaje de modelado aparece en junio de 1996 con el nombre de UML 0.9 [Booch et al., 1996a]. En septiembre de este mismo año aparece la versión 0.91 de UML [Booch et al., 1996b].

Durante 1996 invitan a otros expertos a colaborar con ellos, entre los colaboradores se encuentran nombres muy ilustres y de reconocido prestigio dentro de la comunidad de la Ingeniería del Software y más particularmente de la Orientación al Objeto. Citarlos a todos sería largo, pero como muestra se pueden mencionar a personajes de tanto renombre como *Peter Coad, Derek Coleman, Ward Cunningham, David Ambly, Eric Gamma, David Harel, Richard Helm, Ralph Johnson, Stephen Mellor, Bertrand Meyer, Jim Odell, Kenny Rubin, sally Shlaer, JohnVlissides, Paul Ward, Rebbeca Wirfs-Brock, Edward Yourdon* entre otros.

Además, se crea OTUG (*Object Technology Users Group*) como foro de discusión; una lista de correo en la que se discute y se comparten ideas sobre la tecnología de objetos, teniendo como trasfondo a UML.

Tras la salida a la luz de UML la comunidad de ADOO queda divida en dos grupos principales, aquellos que se unen a la estela de UML y aquéllos que forman una “coalición anti UML”. Esta situación la aprovecha OMG para en 1996 crear el OOAD SIG (*Object-Oriented Analysis and Design Special Interest Group*) que se encargara de canalizar los esfuerzos para conseguir un estándar. De esta forma se organiza la *OMG Task Force RFP* (*Request For Proposal*), es decir una petición de propuestas para la creación de un estándar del lenguaje de

modelado para los métodos de ADOO. Esta petición de propuestas sugería un lenguaje de modelado que contara con un metamodelo, una notación, una sintaxis y una semántica.

Rational va a responder a esta petición de propuesta con la versión 1.0 de UML en enero de 1997 [Booch et al., 1997a; Booch et al, 1997b]. Esta versión está avalada por un conjunto de empresas de mucho prestigio dentro del mundo de la informática: Digital Equipment Corporation, HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, y Unisys.

Como era de esperar, la respuesta de Rational no fue la única, OMG recibió en enero de 1997 las respuestas de IBM & ObjecTime [Cook et al., 1997], Platinum Technology [Rivas et al., 1997], Ptech, Taskon & Reich Technologies y Softeam.

Durante los primeros meses de 1997 se hicieron predicciones de todo tipo y para todos los gustos. La notación de UML se había convertido en un estándar de facto apoyado por empresas de especial relevancia en el sector informático, lo cual colocaba a UML en una posición de privilegio, pero su oscura semántica y su metamodelo parecían no estar a la altura de otras propuestas, lo cual hacía peligrar su adopción como estándar por OMG en detrimento del resto de las propuestas. Así todo apuntaba a dos posibles soluciones. La primera de ellas, y quizás la peor, daba como resultado dos estándares: UML de Rational y OML de Platinum Technology. La segunda de ella apuntaba por una solución mixta que contemplase aspectos de todas las propuestas, siendo la que se llevó a cabo; así se unen al equipo liderado por Rational el resto de los equipos que enviaron propuestas, dando lugar a la versión 1.1 de UML [Rational et al., 1997], que fue enviada a OMG el 1 de septiembre de 1997, contribuyendo todos con sus ideas a la generación de una respuesta única.

Esta propuesta es aceptada por OMG como estándar de lenguaje de modelado el 17 de noviembre de 1997, pasándose a denominar al lenguaje de modelado OMG UML 1.1. En julio de 1998 aparece la versión OMG UML 1.2 [OMG, 1998], presentando sólo cambios editoriales. Casi un año más tarde, en junio de 1999 aparece OMG UML 1.3 [OMG, 1999] con algunos cambios significativos, especialmente en lo tocante a la semántica. La versión OMG UML 1.4 [OMG, 2001b] aparece como versión definitiva en septiembre de 2001 presentando cambios menores, siendo ésta la versión en vigor a fecha de hoy. El camino seguido hasta la versión actual de OMG UML se puede resumir en la Figura 4.2.

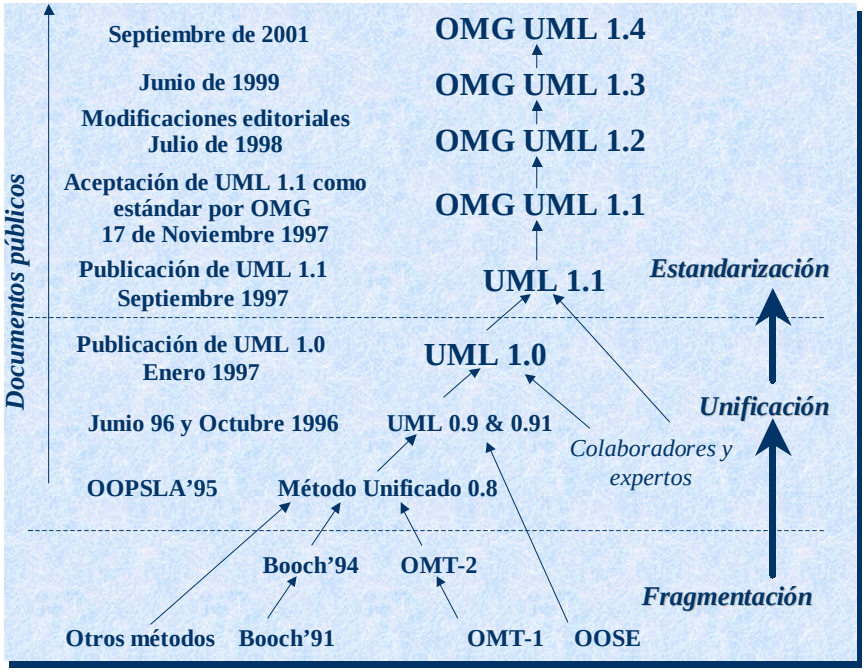


Figura 4.2. Evolución de UML

En los más de cuatro años que han pasado desde que UML fuera aceptado por OMG este lenguaje de modelado se ha convertido en el estándar de facto para especificar artefactos software. Durante este tiempo las modificaciones que ha sufrido UML han sido pequeñas en comparación con la evolución de los métodos que lo han adoptado como lenguaje de modelado. Por este motivo se está preparando una nueva versión que incorporará mayores cambios, y que recibirá el nombre de UML 2.0 [Kobryn, 1999; Kobryn, 2002], siendo el 2002 el año en que se prevé que vea la luz.

Consecuentemente, OMG ha puesto en marcha cuatro RFP para UML 2.0. Una RFP sobre la infraestructura de UML, en la que se busca reestructurar las construcciones básicas de UML y mejorar su configurabilidad [OMG, 2000a]. Una RFP dedicada a la superestructura de UML 2.0 para mejorar elementos avanzados tales como componentes, actividades e interacciones [OMG, 2000b]. Una RFP sobre OCL (*Object Constraint Language*) en pro de incrementar la precisión y expresividad de este lenguaje [OMG, 2000c]. Y por último una RFP para el intercambio de diagramas [OMG, 2001a].

Las referencias básicas de UML las constituyen los libros escritos por los autores principales del mismo [Rumbaugh et al., 1999; Booch et al., 1999; Jacobson et al., 1999], donde los dos primeros están dedicados por completo al lenguaje de modelado, mientras que el tercero se dedica al proceso unificado.

4.3.4.2 Marco conceptual de la Orientación a Objetos

Los sistemas y el pensamiento orientado a objetos se asientan sobre una colección de conceptos que, en su mayor parte, aparecen en diferentes campos y desarrollos de las Ciencias de la Informática con anterioridad a que la Orientación a Objetos fuera un paradigma aceptado como tal.

Desde que los primeros principios de la Orientación a Objetos fueran establecidos en el campo de los lenguajes de programación, fundamentalmente con Simula y Smalltalk, una gran diversidad de campos de la computación han utilizado la tecnología de objetos en el desarrollo de sus teorías; esto ha provocado la existencia de un número de polisemias y sinónimos en relación con el conjunto central de los conceptos inherentes a la Orientación a Objetos. En [Wirfs-Brock y Johnson, 1990] se hace un esfuerzo por presentar los conceptos básicos del paradigma, clarificando su significado con respecto a diferentes interpretaciones.

Para solucionar esta confusión conceptual se establece un *idioma común* que permita la comunicación entre equipos de desarrollo de software con tecnología de objetos. El conjunto de conceptos y propiedades, que configuran este *idioma común* es lo que se conoce como **Modelo Objeto**, y que puede definirse como sigue: "*Un Modelo Objeto es un marco de referencia conceptual, en el que se establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los sistemas software soportados por la tecnología orientada al objeto. Este conjunto básico de conceptos deberá incluir como mínimo, los de abstracción, encapsulación, jerarquía y modularidad; y deberá considerar el sistema de información como un conjunto de entidades conceptuales modeladas como objetos e interactuando entre ellas*" [Marqués, 1995].

Desde la perspectiva de la evolución de los modelos objetos, puede considerarse que el primero de ellos, reconocido como tal, es el propuesto por Anita K. Jones [Jones, 1987] en el año 1978, definiendo el modelo objeto como un concepto y una herramienta, que proporciona las líneas directrices para caracterizar las entidades abstractas, en los términos en los que se conciben.

A partir de esta primera propuesta formal de modelo objeto, y ante la ausencia de un modelo común de referencia y de estándares universalmente aceptados, para estos modelos, se ha asistido a una proliferación de propuestas de modelos objetos [Sernadas et al., 1989; Ward, 1989; Castellanos et al., 1991; Bertino y Martino, 1993; Marqués, 1995] entre otros. Además, se puede constatar que la mayoría de los trabajos en la Orientación a Objetos definen de forma implícita o explícita su propio modelo objeto, como por ejemplo el de UML [OMG, 2001b] o el de OML [Firesmith et al., 1998].

Según Grady Booch [Booch, 1994] todo modelo objeto está formado por cuatro elementos fundamentales: *abstracción, encapsulamiento, modularidad* y *jerarquía*; y por tres elementos secundarios: *tipos, concurrencia* y *persistencia*. "Fundamentales" significa que un modelo que carezca de cualquiera de estos elementos no es orientado a objetos. Con "secundarios" quiere decirse que cada uno de ellos es una parte útil del modelo objeto, pero no esencial.

La *abstracción* se utiliza para combatir la complejidad. Surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias. Una abstracción se centra en la visión externa de un objeto, sirviendo para separar el comportamiento esencial de un objeto de su implantación.

Se puede definir abstracción como: "*Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes*" [Graham, 1994].

Mientras que la abstracción ayuda a las personas a pensar sobre lo que están haciendo, el encapsulamiento permite que los cambios realizados en los programas sean fiables con el menor esfuerzo. El encapsulamiento genera una cápsula, una barrera conceptual sobre la información y servicios de un objeto, haciendo que estos permanezcan juntos. El encapsulamiento es un medio para conseguir el principio de ocultación de la información [Parnas, 1972]. Se puede definir encapsulamiento como: "*Un principio de estado que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto*" [Piattini et al., 1996].

La modularidad permite la fragmentación de un programa en componentes individuales, lo que puede reducir la complejidad en algún grado. Dicha fragmentación crea una serie de fronteras bien definidas y documentadas dentro del programa. Estas interfaces tienen una importancia incalculable para la comprensión del programa. La modularidad se puede definir como: "*La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados*" [Booch, 1994].

La abstracción es un concepto sumamente útil, pero demasiado extenso, excepto para los casos triviales. El encapsulamiento y la modularidad son medios para manejar las abstracciones. Con frecuencia, un conjunto de abstracciones forma una jerarquía. La jerarquía se puede definir como: "*La jerarquía es una clasificación u ordenación de abstracciones*" [Booch, 1994].

Los objetos se comunican a través del *paso de mensajes*. Esto elimina la duplicación de datos, garantizando que no se propaguen los efectos de los cambios en las estructuras de datos encapsuladas dentro del objeto sobre otras partes del sistema. Así, un objeto accede a otro

enviándole un mensaje, cuando esto ocurre, el receptor ejecuta el método correspondiente al mensaje. Un mensaje consiste en un nombre de un método más cualquier argumento adicional. El conjunto de mensajes a los que un objeto responde caracteriza su comportamiento. Se define mensaje como: "*Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento*" [OMG, 2001b].

El concepto de tipo se deriva de las teorías sobre los tipos abstractos de datos. Se incluye el tipo como elemento separado de modelado de objetos porque el concepto de tipo pone énfasis en el significado de la abstracción en un sentido muy distinto. Se puede definir *tipo* como: "Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas" [Booch, 1994].

Un concepto muy ligado a la teoría de tipos es el *polimorfismo*, que indica que un solo nombre puede denotar objetos de muchas clases diferentes que se relacionan por una superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto de operaciones. Lo opuesto al polimorfismo es el *monomorfismo*, que se encuentra en todos los lenguajes con comprobación estricta de tipos y ligadura estática. Existe el polimorfismo denominado de *inclusión* cuando interactúan las características de la herencia y el enlace dinámico. Es una de las características más potentes de los lenguajes de programación orientados a objetos después de su capacidad para soportar la abstracción, y es lo que distingue a la programación basada en tipos abstractos de datos. Se puede definir polimorfismo como: "*La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases*" [Graham, 1994].

Un sistema que implique concurrencia puede tener muchos hilos de control (algunos transitorios y otros permanentes durante toda la ejecución del sistema). Mientras que la Programación Orientada a Objetos se centra en la abstracción de datos, encapsulamiento y la herencia, la concurrencia se centra en la abstracción de procesos y la sincronización; los objetos unifican los dos puntos de vista anteriores. Se define la concurrencia como: "*Concurrencia es la propiedad que distingue un objeto activo de uno que no está activo*" [Booch, 1994].

Todo objeto software ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo. Así pues, se puede definir la *persistencia* como: "*La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la*

posición del objeto varía con respecto al espacio de direcciones en el que fue creado)" [Booch, 1994].

4.3.5 El cuerpo de conocimiento de la Ingeniería del Software

Atendiendo a todo lo expuesto, parece imprescindible incluir en la asignatura de Ingeniería del Software los conceptos relacionados con los actores implicados en el desarrollo de proyectos, aspectos del desarrollo y la calidad de los productos finales, así como los procedimientos, herramientas y métodos de trabajo a disposición del analista para poder construir el software de calidad que el usuario necesita. Teniendo en cuenta que entre los enfoques metodológicos mencionados anteriormente la orientación a objetos es una de las líneas más prometedoras y que las orientadas a procesos y las orientadas a estados y transiciones siguen siendo las más utilizadas y difundidas en la actualidad, estos serán los enfoques metodológicos que se incluirán en los programas de las asignaturas presentados en este Proyecto Docente.

Una vez que se ha introducido cual es el marco histórico y conceptual de la materia, se va a perfilar de una manera más concreta el conjunto de conocimientos que entran dentro del área de influencia de la Ingeniería del Software: *su cuerpo de conocimiento*.

Una de las tareas básicas para la definición de una profesión es el establecimiento del conjunto de conocimientos que el profesional debe poseer para el adecuado ejercicio de su labor profesional. Este cuerpo de conocimiento es fundamental para constituir el resto de los elementos que conformarán la profesión, esto es, una propuesta curricular y una política de certificación de los estudios y de los profesionales.

Ante esta necesidad se han propuesto diferentes alternativas, aunque afortunadamente sólo la propuesta de IEEE-CS parece ser la que ha canalizado los resultados, y se ha convertido en la referencia al cuerpo de conocimiento de la Ingeniería del Software.

A continuación se va a describir este cuerpo de conocimiento coordinado por IEEE-CS, aunque se presentarán someramente, a modo informativo, algunas otras propuestas que nacieron casi en paralelo a la de IEEE-CS, pero que quedaron rápidamente olvidadas o absorbidas por ésta.

4.3.5.1 SWEBOK propuesto por IEEE-CS

De las diferentes propuestas para la creación de un cuerpo de conocimiento de la Ingeniería del Software, el proyecto SWEBOK (*Software Engineering Body of Knowledge*) [Abran et al., 2001c], coordinado por IEEE-CS y respaldado por otras importantes compañías y

organizaciones científicas como ACM, Mitre, Boeing o Racional Software Coorporation entre otras, es el que ha acabado acatándose como estándar internacional, aunque a fecha de hoy todavía no ha concluido, habiéndose finalizado en el año 2001 la segunda fase, versión del hombre de piedra [Abran et al., 2001b]. Queda pendiente la tercera y última fase de este proyecto, que se conoce como la versión del hombre de hierro, y que dará comienzo tras un período no inferior a dos años de utilización y maduración de la versión del hombre de piedra, posibilitando así la inclusión de una retroalimentación en el proyecto.

Las previsiones iniciales sobre la duración del proyecto se han visto bastante alteradas, pues en un principio se pensó que el proyecto estaría terminado completamente en el año 2001, tal y como se muestra en la Figura 4.3.

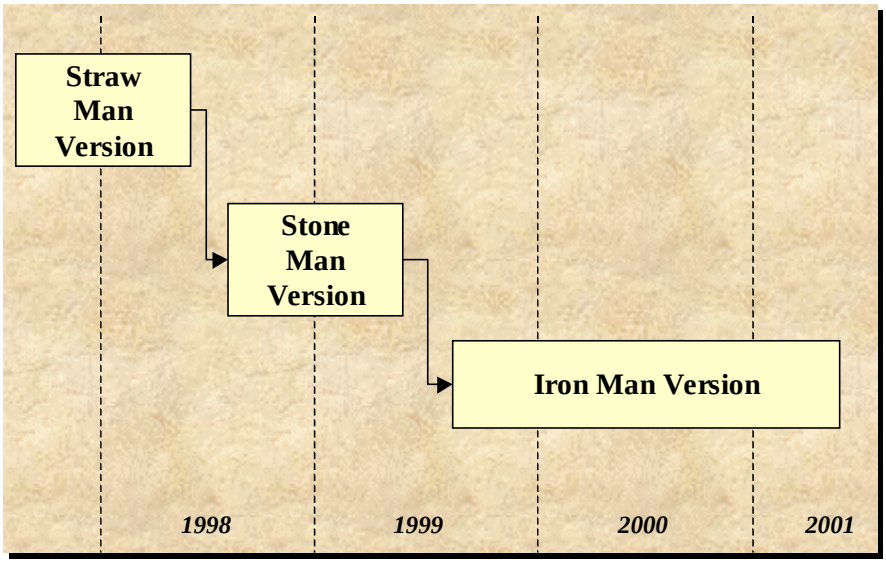


Figura 4.3. Duración inicialmente prevista para el proyecto SWEBOK [Dupuis et al., 1999]

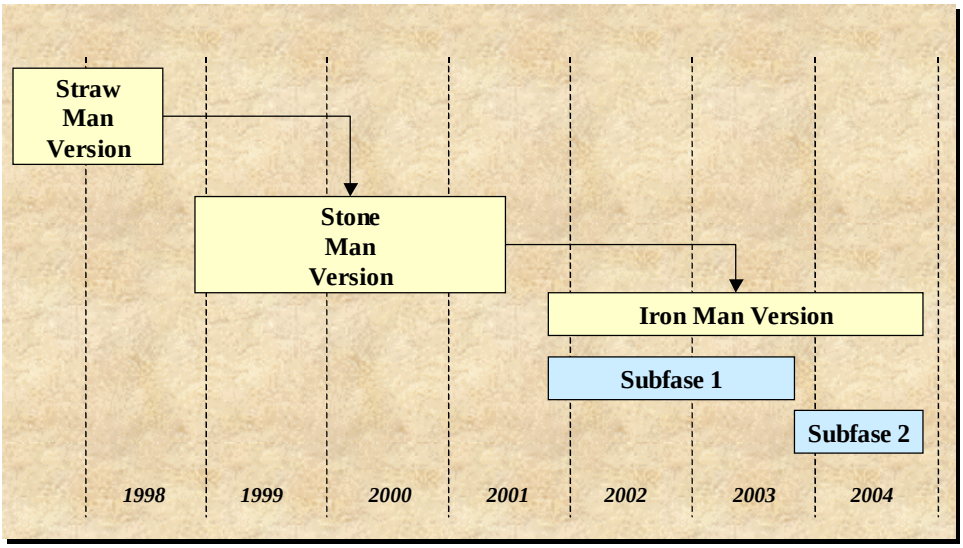


Figura 4.4. Duración prevista del proyecto SWEBOK al finalizar la versión del hombre de piedra (figura basada en [Dupuis et al., 2000])

Sin embargo, la realidad se desarrolló de forma diferente. Así mientras que la versión del hombre de paja [Bourque et al., 1998] concluyó en el tiempo previsto, septiembre de 1998, en la fecha aproximada de conclusión de la versión del hombre de piedra, finales de 1999, sólo se había terminado el primer ciclo de revisión de la misma por un grupo reducido de expertos, dando lugar a la versión 0.5 [Abran et al., 1999]. El segundo ciclo de revisión de la versión del hombre de piedra por un conjunto de usuarios seleccionados concluye en abril de 2000 con la versión 0.7 [Abran et al., 2000]. El tercer ciclo de revisión es llevado a cabo por la comunidad de Ingeniería del Software y concluye en febrero de 2001 con la versión 0.9 [Abran et al., 2001a], que aparece como versión final 0.95 de libre distribución en mayo de 2001 [Abran et al., 2001b] y como libro publicado por IEEE-CS en diciembre de 2001 [Abran et al., 2001c]. La versión del hombre de hierro cuenta de dos subfases, la primera de ellas con una duración estimada no inferior a los dos años monitorizará la utilización de la versión disponible en Internet [Abran et al., 2001b], para que al tercer año se comience el desarrollo propiamente dicho de la versión del hombre de hierro en la segunda subfase de ésta. El proceso seguido hasta la fecha en el desarrollo del SWEBOK se resume en la Figura 4.4.

Los objetivos de este proyecto se resumen en los siguientes cinco puntos [Bourque et al., 1999a; Abran et al., 2001b]:

1. Promover una vista consistente de la Ingeniería del Software para todo el mundo.
2. Clarificar el lugar, y establecer los límites, de la Ingeniería del Software con respecto a otras disciplinas, tales como la Ciencia de la Computación, la Gestión de Proyectos, la Ingeniería de Computadores o las Matemáticas.
3. Caracterizar los contenidos de la Ingeniería del Software como disciplina.
4. Ofrecer un acceso al cuerpo de conocimiento de la Ingeniería del Software.
5. Ofrecer las bases para el desarrollo de una propuesta curricular y una política de certificación, relacionadas ambas con la Ingeniería del Software.

El producto que ha resultado de este proyecto no es el cuerpo de conocimiento en sí, sino una guía de él. El conocimiento ya existe, lo que se busca es el consenso para determinar el subconjunto de conceptos esenciales que caracterizan a la Ingeniería del Software.

La guía que, como ya se ha mencionado, actualmente ha concluido de su segunda fase (versión del hombre de piedra), se divide en diez áreas de conocimiento, cada una de las cuales se estudia en un capítulo independiente de la guía y cuenta con una serie de especialistas responsables. Tanto las áreas como los especialistas para la versión actual quedan reflejados en la Tabla 4.2.

Área de Conocimiento	Especialistas
Requisitos del software	**P. Sawyer y G. Kotonya** (*Lancaster University, Reino Unido*)
Diseño del software	**G. Tremblay** (*Université du Québec à Montreal, Canadá*)
Construcción del software	**T. Bollinger** (*The MITRE Corporation, USA*), **P. Gabrini y L. Martin** (*Université du Québec à Montreal, Canadá*)
Prueba del Software	**A. Bertolino** (*National Research Council, Italia*)
Mantenimiento del software	**T. M. Pigoski** (*Techsoft, Inc., USA*)
Gestión de la configuración del software	**J. A. Scott y D. Nisse** (*Lawrence Livermore Laboratory, USA*)
Gestión de la Ingeniería del Software	**S. G. MacDonell y A. R. Gray** (*University of Otago, Nueva Zelanda*)
Proceso de Ingeniería del Software	**K. El Emam** (*National Research Council, Canadá*)
Herramientas y métodos de la Ingeniería del Software	**D. Carrington** (*The University of Queensland, Australia*)
Calidad del software	**D. Wallace y L. Reeker** (*National Institute of Standards and Technology, USA*)

Tabla 4.2. Las áreas de conocimiento del SWEBOK y sus responsables [Abran et al., 2001b]

El número de áreas de conocimiento no ha sido siempre de diez, ni han tenido siempre el mismo nombre. En la *Straw Man Version* [Bourque et al., 1998] fueron quince las áreas de conocimiento consideradas; éstas se obtuvieron del estudio de libros de texto sobre Ingeniería del Software, del estudio de programas de cursos universitarios y *masters* en Ingeniería del Software, y especialmente el ciclo de vida ISO/IEC 12207[25] [ISO/IEC, 1995] fue considerado como una entrada principal para esta versión de la guía, marcando las bases y el vocabulario para la clasificación de los diferentes temas relacionados con el ciclo de vida.

Además, se han considerado siete disciplinas relacionadas con la Ingeniería del Software: Ciencias cognitivas y factores humanos, Ingeniería de computadores, Ciencia de la Computación, Gestión y Ciencia de la gestión, Matemáticas, Gestión de proyectos e Ingeniería de Sistemas.

El proyecto SWEBOK especifica las unidades de conocimiento, así como los temas pertenecientes a dichas áreas de conocimiento, que se considerará el conocimiento esencial que debe poseer un ingeniero del software. Éstos deben también poseer ciertos conocimientos de las disciplinas relacionadas, pero no es cometido del SWEBOK especificar estos conocimientos.

La guía del SWEBOK se organiza de forma jerárquica, descomponiendo cada área de conocimiento en un conjunto de temas con nombres fácilmente reconocibles, como se puede

[25] Adoptado por IEEE/EIA y por ISO/IEC como un estándar.

apreciar en la Figura 4.5 y en la Figura 4.6. Una breve descripción de cada una de las diez áreas de conocimiento se presenta en el Cuadro 4.2.

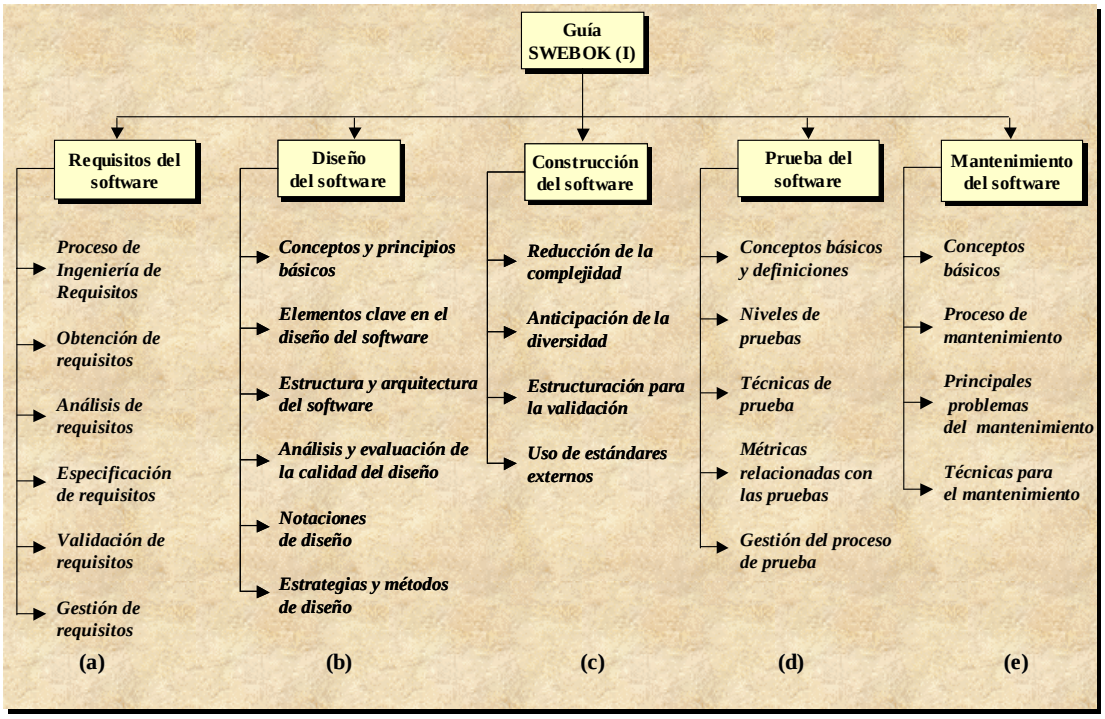


Figura 4.5. Estructura de la guía SWEBOK (parte I) [Abran et al., 2001b]

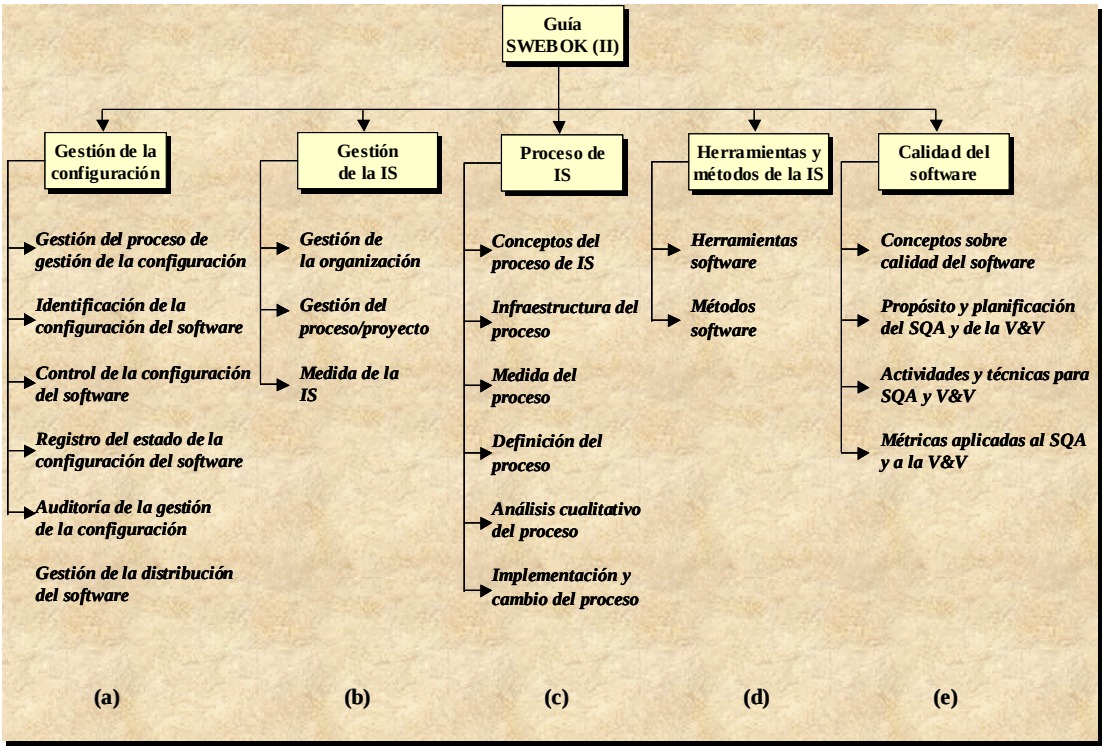


Figura 4.6. Estructura de la guía SWEBOK (parte II) [Abran et al., 2001b]

En la Tabla 4.3 se recoge la equivalencia entre las áreas de conocimiento propuestas en las dos versiones de la guía SWEBOK existentes hasta la fecha.

Requisitos del software: Se divide en seis subáreas que se corresponden aproximadamente con las actividades del proceso que se desarrollan iterativa y concurrentemente, más que de forma secuencial. La parcela del *proceso de ingeniería de requisitos* presenta e introduce las otras cuatro. La subárea de *obtención de requisitos* cubre la captura, descubrimiento o adquisición de éstos. La parte de *análisis* trata de solventar los conflictos entre los requisitos, así como la frontera del sistema. La especificación se refiere a la estructura, calidad y capacidad de verificación del documento de requisitos. La *validación de requisitos* busca conflictos, omisiones y ambigüedades; a la vez que asegura que los requisitos siguen un estándar de calidad. Por último, la subárea de *gestión de requisitos* permite mantener los requisitos presentes en todo el ciclo de vida, adaptándolos a los cambios sufridos por el proyecto.

Diseño del software: Transforma los requisitos, típicamente expresados en términos del dominio del problema, en descripciones que explican cómo resolver el problema. Describe cómo el sistema se descompone y se organiza en componentes, describiendo las interfaces entre ellos.

Construcción del software: Es el acto fundamental de la Ingeniería del Software. Requiere el establecimiento de un diálogo entre el ingeniero y el computador, entre representantes de dos mundos totalmente diferentes. Para establecer los temas de esta unidad de conocimiento se adoptan dos vistas complementarias de la construcción del software. La primera de ellas establece las tres interfaces principales para la creación de software: *lingüista, matemática* y *visual*. La segunda establece que, para cada uno de los estilos, los temas a tratar se organicen de acuerdo a cuatro principios de organización: *reducción de la complejidad, anticipación de la diversidad, estructuración de la validación* y *uso de estándares externos*.

Prueba del software: Consiste en la verificación dinámica del comportamiento de los programas ante un conjunto finito de casos de prueba.

Mantenimiento del software: Estudia los procesos relacionados con la modificación de un producto software después de su entrega, para corregir faltas, mejorar su rendimiento u otros atributos, o adaptar el producto a otro entorno.

Gestión de la configuración del software: Es la disciplina que identifica la configuración de puntos discretos en el tiempo para lograr un control sistemático de sus cambios y mantener la integridad y la trazabilidad a través del ciclo de vida del sistema.

Gestión de la Ingeniería del Software: Une la gestión del proceso y la parte de métricas del proceso. La gestión del proceso casa con la noción de "*gestión a la larga*", esto es, trata de la organización de las fases del ciclo de vida. La parte de métricas aborda *la medida de los objetivos del programa, la medida de la selección, la recolección de datos* y *el desarrollo de modelos*.

Proceso de la Ingeniería del Software: Cubre la definición, implementación, medida, gestión, cambio y mejora de los procesos software.

Herramientas y métodos de la Ingeniería del Software: Describe dos subáreas que discurren de forma horizontal a través de otras áreas de conocimiento: *las herramientas software* y *los métodos de desarrollo*.

Calidad del software: Discute sobre el aseguramiento de la calidad que todo producto resultado de un proceso de Ingeniería del Software debe tener.

Cuadro 4.2. Descripción de las áreas de conocimiento del SWEBOK en la *Stone Man Version*

Área de Conocimiento en la *Stone Man Version*	Áreas de Conocimiento correspondientes en la *Straw Man Version*	Notas
Requisitos del software	Análisis de requisitos[♣]	
Diseño del software	Diseño detallado[♣]	Se decidió cubrir todo el diseño desde el principio, no distinguiendo entre diseño arquitectónico y diseño detallado. Esta distinción aparece en el estándar 12207
Construcción del software	Codificación[♣]	
Prueba del Software	Prueba[♣]	
Mantenimiento del software	Proceso de mantenimiento[♣]	
Gestión de la configuración del software	Gestión de la configuración[♣]	
Gestión de la Ingeniería del Software	Proceso de gestión[♣] Medida/Métricas	La definición de este proceso en el estándar 12207 hace mención del uso de datos cuantificables para la toma de decisiones, por eso la parte de *métricas* se ha agrupado con la de *proceso de gestión*
Proceso de Ingeniería del Software	Proceso de mejora[♣]	
Herramientas y métodos de la Ingeniería del Software	Métodos de desarrollo (orientados al objeto, formales, prototipado) Entornos de desarrollo de software	Los entornos de desarrollo (herramientas) y la reutilización se consideraron los dos elementos principales del proceso de infraestructura. Los métodos se incluyeron porque con frecuencia las herramientas se construyen para implementar métodos determinados
Calidad del software	Aseguramiento de la calidad[♣] Verificación y validación[♣] Confiabilidad del software	Todas las áreas relacionadas con la calidad se han agrupado en una sola área de conocimiento
	Presentación y definición de la Ingeniería del Software	Se incluyó en la *Straw Man Version* porque aparecía en todos los libros de texto sobre Ingeniería del Software, y se necesita algún tipo de introducción en la *Stone Man Version*, pero no ha sido incluida en ninguna área como tal.

Tabla 4.3. Equivalencia entre la lista de áreas de conocimiento de la *Stone Man Version* y de la *Straw Man Version* de la guía SWEBOK [Bourque et al., 1999b]

[♣] Área de la *Straw Man Version* basada en el estándar ISO/IEC 12207.

En la Tabla 4.4 se establece la correspondencia entre las áreas de conocimiento propuestas en la *Stone Man Version* v0.95 de la guía SWEBOK [Abran et al., 2001b] y el estándar ISO/IEC 12207 [ISO/IEC, 1995].

Área de Conocimiento de la *Stone Man Version*	Estándar ISO/IEC 12207	
Requisitos del software	Análisis de requisitos	PROCESOS PRINCIPALES
Diseño del software	Diseño arquitectónico Diseño detallado	
Construcción del software	Codificación Integración	
Prueba del Software	Prueba Instalación Soporte a la aceptación Proceso de operación Explotación del software Soporte operativo a los usuarios	
Mantenimiento del software	Proceso de mantenimiento	
Gestión de la configuración del software	Gestión de la configuración	PROCESOS DE SOPORTE
Calidad del software	Aseguramiento de la calidad Verificación y validación Revisión conjunta Auditoría	
Gestión de la Ingeniería del Software	Procesos de gestión	PROCESOS DE LA ORGANIZACIÓN
Herramientas y métodos de la Ingeniería del Software	Proceso de infraestructura	
Proceso de Ingeniería del Software	Proceso de mejora	

Tabla 4.4. Correspondencia entre las áreas de conocimiento de la guía SWEBOK *Stone Man Version* y el estándar ISO/IEC 12207 [Bourque et al., 1999b]

4.3.5.2 SWE-BOK propuesto para la FAA

Este cuerpo de conocimiento (denotado por las siglas SWE-BOK) es el resultado de un trabajo patrocinado por la FAA (*Federal Aviation Administration*) de EEUU como parte de un proyecto para mejorar los procesos de adquisición, desarrollo y mantenimiento del software de dicha entidad.

Este cuerpo de conocimiento pretende contribuir al trabajo que está realizando el SWECC (*Software Engineering Coordination Committee*) bajo el patrocinio de ACM e IEEE-CS para el desarrollo de la Ingeniería del Software y su madurez como disciplina.

Este cuerpo de conocimiento se recoge en [Hilburn et al., 1999], donde el término *conocimiento* se utiliza para describir el espectro completo del contenido de la disciplina: *información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos* y *bibliografía*.

Este cuerpo de conocimiento se estructura en tres niveles de abstracción: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*, para lograr así un balance entre la simplicidad y la claridad y el nivel apropiado de detalle en la descripción del conocimiento. En el Cuadro 4.3 se recogen las definiciones que se manejan en esta estructuración, mientras que en la Figura 4.7 se pueden apreciar estos niveles de abstracción de forma gráfica.

Conocimiento: Término utilizado para describir el espectro completo de los contenidos de la disciplina: *información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos* y *bibliografía*.

Cuerpo de conocimiento: Descripción jerárquica del conocimiento sobre la Ingeniería del Software que organiza y estructura el conocimiento en tres niveles de jerarquía: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*.

Categoría de conocimiento: Una subdisciplina de la Ingeniería del Software que es generalmente reconocida como una parte significativa de este cuerpo de conocimiento de la Ingeniería del Software. Son elementos estructurales de alto nivel, utilizados para organizar, clasificar y describir el conocimiento sobre Ingeniería del Software. Cada una de ellas está compuesta por un conjunto de áreas de conocimiento.

Área de conocimiento: Una subdivisión de una categoría de conocimiento que representa el conocimiento de la Ingeniería del Software que está lógicamente cohesionado y relacionado con la categoría de conocimiento mediante la herencia o la agregación. Cada una de ellas está compuesta de un conjunto de unidades de conocimiento.

Unidad de conocimiento: Una subdivisión de un área de conocimiento que representa un componente básico del cuerpo de conocimiento de la Ingeniería del Software que tiene una descripción explícita. Para el propósito de este cuerpo de conocimiento, cada una de estas unidades es atómica; esto es, no se subdivide en elementos más básicos.

Cuadro 4.3. Definiciones manejadas en el SWE-BOK de la FAA [Hilburn et al., 1999]

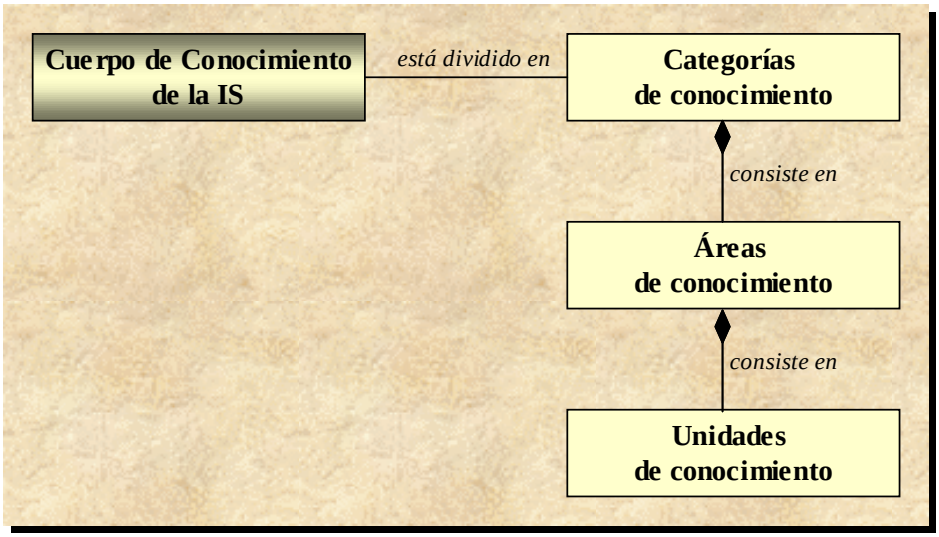


Figura 4.7. Niveles de abstracción en la arquitectura del SWE-BOK de la FAA [Hilburn et al., 1999]

Este cuerpo de conocimiento está formado por cuatro categorías de conocimiento: *Fundamentos de Informática, Ingeniería del Producto Software, Gestión del Software* y *Dominios Software*. En las siguientes tablas se resume este cuerpo de conocimiento.

1. FUNDAMENTOS DE INFORMÁTICA	
Descripción:	Esta categoría concierne al conocimiento, conceptos, teoría, principios, métodos, propiedades y aplicaciones de Informática que forman la base para el desarrollo de software y de la disciplina de la Ingeniería del Software.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
1.1 Algoritmos y estructuras de datos	1.1.1 Estructuras de datos básicas 1.1.2 Diseño de algoritmos 1.1.3 Análisis de algoritmos
1.2 Arquitectura del computador	1.2.1 Sistemas digitales 1.2.2 Organización de un sistema computacional 1.2.3 Arquitecturas alternativas 1.2.4 Comunicaciones y redes
1.3 Fundamentos matemáticos	1.3.1 Lógica matemática y prueba de sistemas 1.3.2 Estructuras matemáticas discretas 1.3.3 Sistemas formales 1.3.4 Combinatoria 1.3.5 Probabilidad y estadística
1.4 Sistemas operativos	1.4.1 Fundamentos de sistemas operativos 1.4.2 Gestión de procesos 1.4.3 Gestión de memoria 1.4.4 Seguridad y protección 1.4.5 Sistemas distribuidos y de tiempo real
1.5 Lenguajes de programación	1.5.1 Teoría de lenguajes de programación 1.5.2 Paradigmas de programación 1.5.3 Diseño e implementación de lenguajes de programación

Tabla 4.5. Categoría de Fundamentos de Informática

2. INGENIERÍA DEL PRODUCTO SOFTWARE	
Descripción:	Esta categoría se refiere a un conjunto de actividades bien definidas e integradas para producir productos software consistentes. Incluye actividades técnicas, que involucran la documentación de estos productos y el mantenimiento de la traza y la consistencia entre ellos. También se refiere al control de la transición entre las diferentes fases del ciclo de vida del software, así como a las actividades que se necesitan para ofrecer productos software de alta calidad a los clientes.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
2.1 Ingeniería de requisitos del software	2.1.1 Obtención de requisitos 2.1.2 Análisis de requisitos 2.1.3 Especificación de requisitos
2.2 Diseño del software	2.2.1 Diseño arquitectónico 2.2.2 Especificación abstracta 2.2.3 Diseño de la interfaz 2.2.4 Diseño de las estructuras de datos 2.2.5 Diseño de algoritmos
2.3 Codificación del software	2.3.1 Implementación de código 2.3.2 Reutilización de código 2.3.3 Estándares de codificación y documentación
2.4 Prueba del software	2.4.1 Pruebas de unidad 2.4.2 Pruebas de integración 2.4.3 Pruebas del sistema 2.4.4 Pruebas de rendimiento 2.4.5 Pruebas de aceptación 2.4.6 Pruebas de instalación 2.4.7 Documentación de la prueba
2.5 Explotación y mantenimiento	2.5.1 Instalación y explotación del software 2.5.2 Operaciones de mantenimiento del software 2.5.3 Proceso del mantenimiento del software 2.5.4 Gestión del mantenimiento del software 2.5.5 Reingeniería del software

Tabla 4.6. Categoría de Ingeniería del Producto Software

3. GESTIÓN DEL SOFTWARE	
Descripción:	Esta categoría casa con los conceptos, métodos y técnicas para la gestión de los productos y proyectos software. Incluye actividades relacionadas con la gestión de proyectos, gestión de riesgos, calidad del software y gestión de la configuración.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
3.1 Gestión del proyecto software	3.1.1 Planificación del proyecto 3.1.2 Organización del proyecto 3.1.3 Estimación del proyecto 3.1.4 Calendario del proyecto 3.1.5 Control del proyecto
3.2 Gestión de riesgos del software	3.2.1 Análisis del riesgo 3.2.2 Planificación de la gestión del riesgo 3.2.3 Monitorización del riesgo
3.3 Gestión de la calidad del software	3.3.1 Aseguramiento de la calidad del software 3.3.2 Verificación y validación 3.3.3 Métricas del software 3.3.4 Sistemas dependientes
3.4 Gestión de la configuración software	3.4.1 Identificación de la configuración del software 3.4.2 Control de la configuración del software 3.4.3 Auditoría de la configuración del software 3.4.4 Contabilidad del estado de la configuración del software
3.5 Gestión del proceso software	3.5.1 Gestión cuantitativa del proceso del software 3.5.2 Mejora del proceso del software 3.5.3 Evaluación del proceso software 3.5.4 Automatización del proceso software 3.5.5 Ingeniería del proceso software
3.6 Adquisición del software	3.6.1 Gestión de la obtención 3.6.2 Planificación de la adquisición 3.6.3 Gestión del rendimiento

Tabla 4.7. Categoría de Gestión del Software

4. DOMINIOS SOFTWARE	
Descripción:	Esta categoría tiene que ver con el conocimiento de dominios específicos que involucran la utilización y aplicación de la Ingeniería del Software.
ÁREAS DE CONOCIMIENTO	
4.1 Inteligencia artificial	
4.2 Sistemas de bases de datos	
4.3 Interacción hombre-máquina	
4.4 Computación numérica y simbólica	
4.5 Simulación por computadora	
4.6 Sistemas de tiempo real	

Tabla 4.8. Categoría de Dominios Software

4.3.5.3 SE-BOK propuesto por el WGSEET

El fin último del **WGSEET** (*Working Group on Software Engineering Education and Training*) es desarrollar un modelo de currículo para la Ingeniería del Software que pueda ser aplicado en todo, o en parte, en el desarrollo de programas educativos especializados en Ingeniería del Software. Esta propuesta se recoge en [Bagert et al., 1999] (y de forma más esquemática en [Hilburn et al., 1998]).

Como parte importante de este proyecto está la definición de un cuerpo de conocimiento de la Ingeniería del Software que sirva de base a la propuesta curricular. Este cuerpo de conocimiento (denotado por SE-BOK en este trabajo) se organiza en cuatro áreas de conocimiento: *el área central, el área de fundamentos, el área de conceptos recurrentes* y *el área de soporte*, áreas que se detallan en la Tabla 4.9.

SE-BOK [Bagert et al., 1999]	
ÁREAS DE CONOCIMIENTO	**COMPONENTES DE CONOCIMIENTO**
Área Central – incluye aquellos componentes que definen la esencia de la Ingeniería del Software	• Requisitos del software • Diseño del software • Construcción del software • Gestión de proyectos software • Evolución del software
Área de Fundamentos – incluye aquellos componentes que sirve de base a las áreas central y de conceptos recurrentes	• Fundamentos de Informática • Factores humanos • Dominios de aplicación
Área de Conceptos Recurrentes – son elementos que discurren por todos los componentes de conocimiento del área central	• Ética y profesionalismo • Procesos software • Calidad del software • Modelado de software • Métricas del software • Herramientas y entornos • Documentación
Área de Soporte – incluye otros campos de estudio que ofrecen los conocimientos necesarios para completar la educación de los ingenieros del software	• Educación general • Matemáticas • Ciencias naturales • Ciencias sociales • Empresariales • Ingeniería • ...

Tabla 4.9. SE-BOK propuesto por el WGSEET

4.3.5.4 Comparativa

De los diferentes cuerpos de conocimiento que se han presentado, el propuesto por IEEE-CS es el que cuenta con el respaldo internacional y, al igual que el propuesto por el WGSEET, tiene entre sus cometidos servir de base para la definición de una propuesta curricular para la disciplina de la Ingeniería del Software.

El propuesto para la FAA es el que abarca unos contenidos más amplios, buscando definir y evaluar las competencias software que se necesitan en una organización con unas dependencias altas de los sistemas software.

El SWEBOK es la única propuesta que establece la frontera de la Ingeniería del Software identificando otras disciplinas relacionadas, ya que las otras propuestas de una u otra forma introducen en el cuerpo de conocimiento temas relacionados con otras disciplinas. Una de las intersecciones que más se repite es con la parte de Gestión de Proyectos, que por otra parte tiene su propio cuerpo de conocimiento [Duncan, 1996].

4.3.6 La enseñanza de la Ingeniería del Software

Tras la revisión anterior sobre los conceptos relacionados con la disciplina de la Ingeniería del Software, se van a efectuar algunas consideraciones de tipo general sobre la enseñanza de la Ingeniería del Software.

Sin duda, una característica de la Ingeniería del Software como disciplina universitaria, frente a otras disciplinas más establecidas, es el dinamismo con el que cambian sus conceptos y herramientas. Por ello es importante atender a las recomendaciones sobre la enseñanza de la Ingeniería del Software que realizan los organismos y organizaciones internacionales relacionados con la Ciencia de la Computación, los Sistemas de Información y la propia Ingeniería del Software. Atender a este tipo de recomendaciones permite que los conocimientos transmitidos y las habilidades adquiridas por los alumnos respondan a las necesidades profesionales reales y no queden obsoletos en poco tiempo.

Del análisis realizado en el apartado anterior se puede deducir que la Ingeniería del Software consiste en un gran número de actividades interrelacionadas que resultan muy difíciles de conjugar bajo un único epígrafe. Por ello, y sobre todo con fines educativos, es imprescindible introducir una cierta organización en esos contenidos, que permita la definición de un programa educativo consistente y ordenado.

En este sentido en [Ardis y Ford, 1989; Ford y Gibbs, 1989; Ford, 1991a] se identifican para la formación en Ingeniería del Software los puntos de vista del proceso de ingeniería y de

los productos resultantes. A continuación se exponen brevemente las características más importantes de esta doble visión en formación en Ingeniería del Software.

❖ **Punto de vista del proceso**

El proceso de la Ingeniería del Software incluye un amplio rango de actividades realizadas por los ingenieros de software; pero a lo largo de este rango muchos aspectos de estas actividades son similares. Los elementos del proceso pueden considerarse en dos dimensiones: *la actividad* y *el aspecto*.

- *Actividad*. Las actividades del proceso se dividen en cuatro grupos: *desarrollo, control, dirección* y *operaciones*.
    - ***Actividades de Desarrollo.*** Creación o producción del software de los componentes del sistema, incluyendo análisis de requisitos, especificación, diseño, implementación y prueba.
    - ***Actividades de Control***. Estas actividades están más relacionadas con el control del desarrollo que con la producción del software. Las dos actividades principales del control hacen referencia a la evolución del software y a la calidad del software. En este apartado se consideran las actividades relacionadas con la dirección de la configuración, el control de los cambios, el mantenimiento, el control de la calidad, las pruebas, la evaluación, la verificación y la validación.
    - ***Actividades de Dirección***. Implica la dirección ejecutiva, administrativa y supervisora de un proyecto de software, incluyendo las actividades técnicas que soportan el proceso ejecutivo de decisión. Las actividades que se pueden considerar aquí son las de planificación del proyecto, localización de recursos, organización de equipos de trabajo, estimación de costes y aspectos legales.
    - ***Actividades de Operación***. Relacionado con el uso de los sistemas de software. Estas actividades incluyen formación del personal en el uso del sistema, planificación para la entrega e instalación de sistemas, cambio desde el sistema antiguo (manual o automático) al nuevo, operación del software y retirada del sistema.
- *Aspecto*. Las actividades de la Ingeniería se dividen tradicionalmente en actividades analíticas y sintéticas. Se consideran seis aspectos de estas actividades para recoger

esta distinción: *abstracción, representación, métodos, herramientas, medición* y *comunicaciones*.

- ***Abstracción.*** Incluye los principios fundamentales y los modelos formales (Modelos del proceso de desarrollo del software, máquinas de estados finitos y redes de Petri, modelo COCOMO...).
- ***Representación***. Incluye notaciones y lenguajes (Ada, Tablas de Decisión, diagramas de flujo, PERT).
- ***Métodos.*** Incluye métodos formales, prácticas actuales y metodologías (OOD, Programación estructurada...).
- ***Herramientas***. Incluye los conjuntos de herramientas software individuales e integradas (e implícitamente los sistemas hardware donde se ejecutan). Pueden mencionarse las de propósito general (correo electrónico y proceso de textos), las herramientas relativas al diseño e implementación (compiladores y editores sensitivos a la sintaxis) y las herramientas de control de proyectos.
- ***Medición***. Los aspectos de medición incluyen análisis de medidas y evaluación de los productos y el proceso software, así como el impacto del software en la organización; en esta categoría hay que incluir las métricas y los estándares. Esta área merece ser tenida en cuenta en la formación ya que los ingenieros de software, al igual que los ingenieros de los campos tradicionales, necesitan conocer qué medir, cómo medirlo y cómo utilizar los resultados para analizar y evaluar cómo progresan los procesos y productos.
- ***Comunicación***. La comunicación es el último aspecto. Todas las actividades de los ingenieros de software implican comunicación tanto oral como escrita, así como producción de documentación. Un ingeniero de software debe tener unas buenas habilidades en las técnicas generales de comunicación y una comprensión de las formas apropiadas de documentación para cada actividad [Levine et al., 1991; Michael, 2000; Pollock, 2001].

❖ **Punto de vista del producto**

A menudo es conveniente discutir las actividades y aspectos en el contexto de una determinada clase de sistema de software; por ejemplo la programación concurrente y la secuencial tienen características diferenciadoras. Así, se añaden dos nuevas dimensiones a la estructura

organizacional del contenido curricular: *las clases de sistemas software* y *los requisitos del sistema*.

- *Clases de sistemas software*. De las distintas clases que pueden ser consideradas, un grupo se define en función de las relaciones del sistema con su entorno, y sus elementos (partes) están descritos por términos tales como procesamiento por lotes, interactivo, reactivo, tiempo real. Otro grupo tiene elementos descritos en términos tales como distribuido, concurrente, o red. Otro está definido en función de las características internas, tales como orientado a tablas, orientado a procesos o basado en conocimientos. También, se incluyen áreas de aplicación genéricas o específicas, sistemas de aviónica, sistemas de comunicaciones, sistemas operativos, sistemas de base de datos.

- *Requisitos del sistema*. La discusión de los requisitos del sistema generalmente se centra en los requisitos funcionales, pero existen otras categorías que merecen atención. Identificar y reunir esos requisitos es el resultado de las actividades realizadas a lo largo del proceso de Ingeniería del Software. Ejemplos de estos requisitos son: accesibilidad, adaptabilidad, disponibilidad, compatibilidad, exactitud, eficiencia, tolerancia a fallos, integridad, interoperabilidad, mantenibilidad, rendimiento, portabilidad, protección, formalidad, reusabilidad, robustez, seguridad, comprobabilidad y usabilidad.

Otra forma de enfocar la enseñanza de la Ingeniería del Software es mediante un enfoque basado en un proyecto para que el alumno se aproxime al trabajo tal cual ocurre (o debiera ocurrir) en la realidad empresarial [Tomayko, 1987; Shaw y Tomayko, 1991].

En Psicología se distinguen dos tipos de conocimientos: *declarativo* y *procedural* [Norman, 1988]. El primero es fácil de transcribir y de enseñar; sin embargo, el segundo es imposible de transcribir y difícil de enseñar, siendo más sencillo de transmitir mediante demostración y de aprender por la práctica. Muchos de los procesos de la Ingeniería del Software dependen del conocimiento procedural. Por este motivo se recomienda una importante parte de experiencia adquirida mediante proyectos [Ardis y Ford, 1989].

En [Shaw y Tomayko, 1991] se presentan diferentes modelos de cursos de Ingeniería del Software que toman el proyecto como eje conductor de los mismos. Los modelos que se enuncian son:

- ***Modelo de Ingeniería del Software como producto***. Es el modelo al que se ajustan los cursos compuestos exclusivamente por clases teóricas. Son cursos que

concentran en, aproximadamente, un cuatrimestre los conceptos de la Ingeniería del Software. Su mayor desventaja es la ausencia de parte práctica y, por tanto, de experiencia. Estos cursos se adecuan a lo que se denomina *énfasis por el ciclo de vida*, que se ajusta a la forma de organizar los libros de texto sobre Ingeniería del Software, especialmente siguiendo el ciclo de vida en cascada, como claramente se aprecia en el libro de Richard Fairley [Fairley, 1985] o en ediciones anteriores de libros tan clásicos como el de Roger S. Pressman [Pressman, 1987] o Ian Sommerville [Sommerville, 1989].

- ***Modelo de la aproximación por seminarios.*** Es similar al anterior en el sentido de que la base del curso son las clases teóricas, pero se distingue en que en este modelo de curso, se reserva un tiempo para que los alumnos presenten trabajos que ellos mismos han realizado sobre algún tema concreto, manejando la bibliografía oportuna.

- ***Modelo de proyecto para grupos pequeños.*** Este modelo de curso incluye la realización de un proyecto de pequeñas dimensiones como parte del curso. Es muy seguido porque divide el curso en trabajo de clase y trabajo de proyecto. Los alumnos se dividen en grupos de entre tres y cinco personas, debiendo abordar un proyecto que les sea familiar y puedan terminar en el tiempo asignado al curso. Este curso permite obtener algunas experiencias derivadas de la aplicación de lo explicado en las clases teóricas, pero es deficiente en el sentido de que no les entrena para el trabajo en proyectos grandes.

- ***Modelo de proyecto para grandes equipos.*** Es la mejor opción para aprender las técnicas que se utilizan en los proyectos reales. La idea es realizar un proyecto con toda la clase. Típicamente se elige un proyecto consistente en el desarrollo de un producto software, idealmente destinado a un cliente real. Los alumnos se organizan en un solo equipo de desarrollo, asumiendo cada uno de ellos el rol que le sea asignado, y que coincidirá con los roles que aparecen en los entornos industriales reales. Cada alumno mantendrá el rol durante todo el curso, y aprenderá las actividades de los otros roles a través de la interacción con los otros miembros del equipo. Es inviable de llevar a cabo cuando el número de alumnos es alto y la asistencia no es obligatoria. Además, la relación con las actividades propias de otros roles no es tan intensa como las que uno lleva a cabo. Más opiniones sobre este tema se pueden encontrar en [Robillard y Robillard, 1998; Stevens et al., 2000].

- ***Modelo de proyecto único.*** El curso entero se dedica a la realización de un proyecto. Suele llevarse a cabo cuando la Ingeniería del Software se divide en dos asignaturas independientes, una dedicada por completo a la teoría y la otra a la práctica.

Con independencia del modelo que se siga existen varios tipos de proyectos. En [Shaw y Tomayko, 1991] se hace un repaso de ellos, encontrando:

- ***El proyecto de "juguete".*** La clase se divide en equipos de tres a cinco personas; cada equipo recibe una labor predefinida por el responsable de la asignatura. Puede existir una variante donde cada equipo crea su propia especificación. Las ventajas son que los alumnos aplican las enseñanzas de la Ingeniería del Software trabajando en equipo, aunque no se satisfagan los requisitos por completo o no se llegue a terminar la implementación, y los proyectos sean fáciles de gestionar. Se puede llegar a situaciones en las que alumnos de segundo ciclo actúen como gestores de los proyectos realizados por alumnos de primer ciclo. La mayor desventaja es que se omiten técnicas propias de los proyectos grandes como puede ser la gestión de la configuración. Es una práctica muy extendida.
- ***Proyecto basado en componentes.*** Los alumnos se organizan en grupos que desarrollan componentes software. Se establecen después una serie de proyectos que se llevarán a cabo con los componentes realizados, para ello debe haber una primera etapa de *adquisición de componentes*, una segunda de integración de los mismos y una tercera de *modificación* o *adaptación* a las necesidades del grupo. Es un tipo de proyecto en el que se manejan técnicas propias de proyectos grandes, y los alumnos se acostumbran a diseñar componentes reutilizables. Su mayor desventaja está en su gestión, muy parecida al caso anterior.
- ***Proyecto para un cliente externo.*** Los estudiantes trabajan en componentes que deberán integrarse para la realización de un producto para un cliente externo, que deberá pasar los criterios de aceptación oportunos acordados con el cliente. Se puede llevar a cabo en pequeños equipos o con un gran equipo. Es el que más se acerca a la realidad, debiendo hacer uso de todas las técnicas propias de un proyecto de grandes dimensiones, donde la relación con el cliente será uno de los puntos más sobresalientes. Un ejemplo de este tipo de experiencias se encuentra documentado en [Abi-Raad, 2000].

- ***Proyectos individuales.*** Cada equipo tiene un proyecto diferente. Los equipos pueden tener clientes externos, que con frecuencia son facilitados por el responsable de la asignatura. Puede convertirse en un caos desde la perspectiva de gestión de la globalidad de los proyectos.

En los últimos años son frecuentes las críticas realizadas a la manera de formar a los ingenieros informáticos, entre otras cosas porque los modelos de enseñanza y las propuestas curriculares de los Centros y Universidades no evolucionan al mismo ritmo que lo hace la disciplina, de forma que los nuevos ingenieros son educados de forma similar como se ha hecho durante años [Shaw, 1992; Shaw, 1998]. Además, esta evolución choca frontalmente con la falta de flexibilidad de los programas universitarios que presenta la Universidad Española, por ejemplo, y con el necesario balance con la estabilidad que necesita el profesorado para impartir una materia con garantías y poder asumir, al tiempo, la parte de investigación.

Por este motivo los estudios de Ingeniería del Software como tales se encuentran inmersos en programas más amplios sobre Ciencia de la Computación, como ocurre en España. En el informe FASE sobre los estudios superiores de Ingeniería del Software [Knoke y Bagert, 1998] identifican programas de Ingeniería del Software en 77 instituciones en todo el mundo, de forma que la mayoría de ellas ofrecen *masters* en algún área relacionada con la Ingeniería del Software, y nueve ofrecían doctorados con parte de Ingeniería del Software. Cabe destacar que programas de doctorado específicos en Ingeniería del Software han comenzado a aparecer en la Carnegie Mellon University (http://www.isri.cs.cmu.edu).

Para D. L. Parnas la educación de un ingeniero del software debe tener unas raíces sólidas en la educación de la ingeniería tradicional [Parnas, 1990; Parnas, 1998]. Mientras que Bertrand Meyer expone que las instituciones que enseñan software son las responsables de producir profesionales del mundo del software que construyan y mantengan los sistemas para satisfacción de sus beneficiarios [Meyer, 2001].

En este sentido Meyer propone que el currículo de un ingeniero informático debe incluir cinco elementos complementarios [Meyer, 2001]:

- **Principios**: Conceptos sobre los que se construye un campo de conocimiento.
    - Abstracción; Distinción entre especificación e implementación; Recursión; Ocultación de la información; Reutilización; Combate a la complejidad; Clasificación; Escalabilidad; Diseño para el cambio; Tipado; Contratos; Manejo de excepciones; Pruebas.

- **Prácticas**: Técnicas para la resolución de problemas que han de aplicarse consciente y regularmente.
    - Gestión de proyectos, Gestión de la configuración, Métricas, Ergonomía e Interfaces de usuario, Documentación, Interacción, Análisis de sistemas de alto nivel, Depuración.
- **Aplicaciones**: Áreas tradicionales y específicas de las técnicas software.
    - Algoritmos, Estructuras de datos, Compiladores, Sistemas Operativos, Bases de datos, Inteligencia artificial, computación numérica.
- **Herramientas**: Estado del arte de los productos que facilitan la aplicación de los principios y prácticas.
- **Matemáticas**: Base formal que hace posible la comprensión del resto de los conceptos. No se trata de que los alumnos tengan unos conocimientos sumamente profundos sobre formalismos, sino que tengan la habilidad de aplicar razonamiento matemático en el desarrollo del software [Meyer, 1990]. Además, en la Ingeniería del Software la abstracción es un concepto imprescindible, y si hay una disciplina que entrena a pensar en abstracto ésta es la Matemática [Devlin, 2001].

Retos	Aspiraciones
Identificar los distintos roles que aparecen en el desarrollo del software y ofrecer la educación apropiada a cada uno de ellos	Discriminar entre los distintos roles propios del desarrollo del software
	Hacer una educación del software para no graduados que sea válida a largo plazo
	Ofrecer especialización mediante práctica y educación superior
Instalar una actitud ingenieril en los programas educativos	Integrar un punto de vista ingenieril en los currículos para no graduados en Ingeniería Informática
Mantener los programas educativos preparados para afrontar el cambio en la disciplina	Hacer los currículos flexibles para responder al cambio
	Explotar la tecnología para soportar la educación
	Ofrecer mecanismos efectivos para que los ingenieros del software se mantengan al día en los conocimientos de la disciplina
Establecer credenciales que representen la habilidad con exactitud	Establecer credenciales distintas y apropiadas para los diferentes roles que aparecen en el desarrollo del software, cuando esto sea posible
	Establecer credenciales que reflejen con exactitud la práctica que pueden desarrollar

Tabla 4.10. Retos y aspiraciones de la educación de la Ingeniería del Software [Shaw, 2000]

Mary Shaw cuestiona la manera en que se está educando a los ingenieros del software porque no se distingue entre los diferentes roles que aparecen en el desarrollo del software, esto es, no se diferencia la formación de un ingeniero del software de la de un programador [Shaw, 2000]. En este mismo artículo Mary Shaw identifica los retos y las aspiraciones a conseguir en la educación de la Ingeniería del Software; éstos se recogen en la Tabla 4.10.

Resumiendo, el objetivo central que se debe perseguir con la educación en la disciplina de Ingeniería del Software es transmitir a los estudiantes todo lo que ellos van a necesitar en su futuro profesional como ingenieros del software, combinando los fundamentos científicos necesarios con una cierta dosis de pragmatismo, y todo aplicado con un sentido de la responsabilidad tanto individual como colectivo [Baber, 1998].

4.3.6.1 La enseñanza de la Ingeniería del Software Orientada a Objetos

La Orientación a Objetos es en sí misma una subdisciplina de la Ingeniería del Software, que requiere que sus conceptos básicos queden asentados de forma sólida para su correcta aplicación en el desarrollo de sistemas software, donde se quiere obtener ventaja de todo su potencial [D'Souza, 1996].

Para la enseñanza de la Ingeniería del Software Orientada a Objetos se aconseja seguir un modelo de aprendizaje constructivista, más que un modelo de aprendizaje objetivista [Hadjerrouit, 1999]. Esto es así porque en la aproximación objetivista se ve el proceso de aprendizaje como una *transmisión pasiva de conocimientos*, donde no se necesita ningún conocimiento previo; el resultado de este enfoque es que el alumno acaba sin una adecuada comprensión de la base conceptual del paradigma objetual, con malos hábitos de programación y con serios malentendidos sobre la tecnología de objetos. Por su parte, la aproximación constructivista presenta el aprendizaje como un *proceso activo de construcción*, donde los alumnos construyen su conocimiento sobre la base de su conocimiento previo, siendo necesario probar lo que se conoce y evaluar si entra en conflicto con los conceptos que están adquiriendo [Tobin y Tippins, 1993; Brandt, 1997].

El enfoque constructivo, se asemeja a la estructura de los ciclos de vida del desarrollo de software en la tecnología de objetos, ya que es iterativo e incremental. Esto es así, porque el conocimiento y el entendimiento se va adquiriendo, ampliando y madurando en una serie de etapas [Meyer, 1996].

Otro aspecto a tener en cuenta es el momento en el que se introduce la orientación a objetos en el currículo del alumno. Hay defensores de introducir la orientación a objetos desde el primer momento en la formación de los titulados en Informática [Tewari y Friedman, 1992; Decker y

Hirshfield, 1994; Adams, 1996; Woodman et al., 1996; Hadjerrouit, 1999]; aunque siendo conscientes de que el aprendizaje de los conceptos propios de la tecnología de objetos es difícil para los nuevos alumnos porque requiere una forma de pensar diferente y más profunda en términos de computación [Hadjerrouit, 1999].

Sobre los conocimientos que se requieren para iniciarse en la orientación a objetos, hay quien opina que aquéllos que tienen experiencia en el desarrollo de sistemas software, típicamente en el paradigma basado en procedimientos, aprenden fácilmente la sintaxis de los lenguajes orientados a objetos, pero esto no significa que hayan captado la esencia del diseño orientado a objetos y sean capaces de plasmar dichos diseños con los lenguajes aprendidos [Rosson y Carroll, 1996]; de hecho hay estudios que indican que el conocimiento de lenguajes de tipo procedimental puede llegar a ser una barrera conceptual para introducirse plenamente en la orientación a objetos [Hadjerrouit, 1999].

La forma más extendida para introducir la tecnología de objetos en los currículos de Informática es mediante los cursos de introducción a la programación, donde los lenguajes procedimentales y los métodos de diseño estructurado se verían desplazados por sus homónimos orientados al objeto.

Por supuesto, hay voces que no comporten la idea de que los conceptos del paradigma de la orientación a objetos se introduzcan por la programación, defendiendo que los principios del modelo de objetos para realizar análisis y diseño deben cubrirse con gran detalle antes de llegar a su implementación real en un lenguaje de programación [Northrop, 1992]; o quién no está de acuerdo con que la orientación a objetos se centre sólo en la programación, defendiendo un enfoque más amplio que cubra todo el ciclo vital, con una perspectiva de Ingeniería del Software [Bézivin et al., 1992].

La orientación a objetos se presta a la utilización de los patrones pedagógicos (*reusable pedagogical design patterns*) [Lilly, 1996; Proto-Patterns, 1999; Bergin, 1998a; Bergin, 1998b]. En concreto el proyecto de patrones pedagógicos [Proto-Patterns, 1999] recoge prácticas efectivas de docentes en tecnología de objetos. Estos patrones deben ser fáciles de repetir y de adaptar. Cada patrón debe ser descrito de manera que sea fácilmente instanciable para diferentes *lecciones* y por diferentes *educadores*. Muchos de estos patrones se refieren de forma explícita a temas de tecnologías de objetos, como por ejemplo [Bellin, 1999; Prieto y Victory, 1999; Vaitkevitchius, 1999], aunque otros muchos se pueden aplicar a la docencia de cualquier asignatura, como por ejemplo [Bergin, 1998c; Jalloul, 1999; Manns, 1999]. Los antipatrones son otra herramienta que se puede aplicar en la pedagogía [Dodani, 1999].

4.4 Revisión de los currículos internacionales

Se puede definir currículo como "un plan para educar estudiantes, ofreciéndoles las características y el conocimiento necesarios para vivir y practicar competentemente una profesión. El currículo debe anticiparse al mundo cambiante en que los alumnos graduados vivirán y trabajarán" [Denning, 1992].

En este apartado se analizan las propuestas más importantes en currículos internacionales en Informática, realizadas por instituciones de alto prestigio dentro del mundo informático. Para ello se han identificado tres categorías de programas que se estudiarán por separado: *Ciencia de la Computación (CS), Sistemas de Información (IS) o Gestión de Sistemas de Información (MIS)* e *Ingeniería del Software (SE)*.

Cada una de estas áreas presenta su propio enfoque, pero claramente tienen un núcleo de conocimiento común [Parrish et al., 1998].

La revisión que aquí se hace se centra en destacar la presencia de la materia objeto de este Proyecto Docente en las diferentes propuestas curriculares.

4.4.1 Currículos centrados en la Ciencia de la Computación

La educación en *Ciencia de la Computación e Ingeniería* ha sido un área activa durante toda la historia de la disciplina. En particular desde el establecimiento de los primeros Departamentos de Ciencia de la Computación a mediados de la década de los sesenta [Hopcroft, 1987], se puso una especial atención al reto de educar a los alumnos en un campo tan sumamente cambiante y que evoluciona con tanta rapidez como es la Informática [Tucker et al., 1996].

Las propuestas curriculares más relevantes que aparecen en el campo de la Ciencia de la Computación tienen como protagonistas a las dos asociaciones más prestigiosas en el mundo de la Informática, ACM y IEEE-CS. Estas organizaciones publican por separado diferentes propuestas curriculares entre los años 1968 y 1983, para terminar aunando esfuerzos para definir una propuesta curricular única en el campo de la Ciencia de la Computación e Ingeniería, la propuesta de ACM/IEEE-CS de 1991 (también conocida como *Computing Curricula 1991*) [Tucker et al., 1991a] convirtiéndose en una referencia internacionalmente aceptada para el establecimiento de los Planes de Estudio de las titulaciones de Informática en la Universidad, incluyendo a la Universidad en España.

La propuesta ACM/IEEE-CS de 1991 se ha visto reciente reemplazada (diciembre de 2001) por el denominado *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

En los siguientes subapartados, y siempre desde el prisma de la Ingeniería del Software, se van a exponer las características de ambas propuestas curriculares, la de 1991 porque ha sido uno de los principales referentes a la hora de establecer los Planes de Estudio en vigor en la Universidad Española en general, y en particular en la Universidad de Salamanca; y la de 2001 porque establece las directrices a seguir de aquí en adelante.

No obstante, previamente al estudio de estos currículos se va a hacer un breve repaso por sus antecedentes históricos, para conocer mejor sus orígenes y las motivaciones que se han perseguido desde un principio.

4.4.1.1 Antecedentes de los Computing Curricula 1991 y 2001

Los esfuerzos por definir un modelo de currículo para los programas de Ciencia de la Computación e Ingeniería de Computadores comenzaron en la década de los sesenta, poco después de que se establecieran los primeros Departamentos en estas áreas. Así, en 1968, siguiendo una serie de estudios preliminares [ACM, 1965; COSINE, 1967; SAC, 1967], ACM publica el primer currículo informático, denominado *Curriculum '68* [ACM, 1968]. Esta propuesta propone los estudios básicos y troncales, agrupados en asignaturas, para una titulación de Ciencia de la Computación, correspondiente a un primer ciclo (*undergraduate* o *bachelor* en los países anglosajones y, en particular, en los EEUU); incluye recomendaciones detalladas para los programas académicos, a través de un conjunto de descripciones de cursos y una bibliografía exhaustiva para cada área descrita. Los propios autores del *Curriculum '68* manifiestan que iba destinado a los que pretendían dedicarse a la investigación dentro de la Informática.

En la década de los setenta, la Informática se desarrolla rápidamente, hasta el punto de que las recomendaciones del *Curriculum '68* quedan pronto obsoletas. En esta década, tanto ACM como IEEE-CS establecen comités para desarrollar un currículo revisado en Ciencia de la Computación. En 1977, el comité de IEEE-CS publica un informe sobre programas en Ciencia de la Computación e Ingeniería [EC, 1977]. Este informe fue significativo por el hecho de que presentaba una vista amplia de la disciplina, incorporando más Ingeniería al currículo y tendiendo un puente para salvar la distancia existente entre los programas orientados al software y los orientados al hardware. Respondiendo a las presiones generadas por el rápido desarrollo del campo, IEEE-CS actualiza su propuesta curricular en 1983 [EAB, 1983].

En 1978, ACM revisa su propuesta curricular, surgiendo así el *Curriculum '78* [Austing et al., 1979]. Esta propuesta fue bastante más compresible que su antecesor y tuvo mucho impacto en la educación de la Informática. Entre sus contribuciones el *Curriculum '78* propone un

programa estándar para un conjunto de cursos que forman el núcleo de conocimiento de la Ciencia de la Computación como disciplina. El *Curriculum '*78 se organiza en un conjunto de asignaturas troncales, complementado por un grupo de asignaturas optativas. Se añaden además un grupo de asignaturas de matemáticas, de las cuales las cinco tienen carácter obligatorio y dos optativo. Estas asignaturas se recogen en la Tabla 4.11.

Código	Asignatura	Carácter
CS1	Programación I	Troncal
CS2	Programación II	Troncal
CS3	Introducción a los computadores	Troncal
CS4	Introducción a la estructura de los computadores	Troncal
CS5	Introducción al procesamiento de ficheros	Troncal
CS6	Sistemas operativos y Arquitecturas de computadores I	Troncal
CS7	Estructuras de datos y análisis de los algoritmos	Troncal
CS8	Organización de los lenguajes de programación	Troncal
MA1	Cálculo elemental	Troncal
MA2	Análisis matemático I	Troncal
MA2A	Probabilidades	Troncal
MA3	Álgebra lineal	Troncal
MA4	Estructuras discretas	Troncal
MA5	Análisis matemático II	Optativa
MA6	Probabilidad y estadística	Optativa
CS9	Computadores y sociedad	Optativa
CS10	Sistemas operativos y arquitecturas de computadores II	Optativa
CS11	Diseño de sistemas de gestión de bases de datos	Optativa
CS12	Inteligencia artificial	Optativa
CS13	Algoritmos	Optativa
CS14	Diseño y desarrollo de software	Optativa
CS15	Teoría de los lenguajes de programación	Optativa
CS16	Autómatas, computabilidad y lenguajes	Optativa
CS17	Matemáticas numéricas: Análisis	Optativa
CS18	Matemáticas numéricas: Álgebra lineal	Optativa

Tabla 4.11. Asignaturas en *Curriculum '78*

La principal crítica al *Curriculum '78* se refiere principalmente al papel secundario atribuido a las matemáticas [Ralston y Shaw, 1980; Ralston, 1984; Berztiss, 1987]. De hecho, ninguna asignatura de matemáticas aparece como prerrequisito de ninguna de informática y el componente matemático del currículo no queda bien definido. Otros autores opinan que esta propuesta es un resumen de otros planes ya existentes, no presentando innovaciones [Aiken,

1980; Shaw, 1985], centrándose más en las aplicaciones que en los fundamentos de la informática [Berztiss, 1987].

El *Curriculum '78* con sólo unas pequeñas modificaciones [Koffman et al., 1984; Koffman et al., 1985], constituye un estándar para la educación en Informática, hasta 1991.

En 1981, y tras dos años de trabajo, el *ACM Committee on Curriculum in Computer Science*, presenta una propuesta curricular orientada a nivel de *master* [Magel et al., 1981]. Plantea como objetivo el desarrollo del pensamiento y de la intuición profesional crítica del estudiante, poniendo énfasis en los conceptos, la teoría y la práctica de la Informática. Las asignaturas que se incluyen en esta propuesta curricular se pueden agrupar en cinco áreas: *Lenguajes de programación*, *Sistemas operativos y arquitectura de computadoras*, *Informática teórica*, *Estructuras de datos y ficheros* y *Otras materias*.

En 1984 se presentan unas guías para la acreditación de los profesionales [Mulder y Dalphin, 1984], y en 1986 se presenta un currículo para las escuelas de artes liberales [Gibbs y Tucker, 1986].

Aunque ACM parte de una tradición más proclive a la Ciencia de la Computación, en la última década del siglo XX se ha acercado a la parte de Ingeniería, convergiendo con IEEE-CS, de orientación tradicionalmente ingenieril. De hecho, ACM se está abriendo incluso al mundo de los Sistemas de Información, cosa que nunca ha hecho IEEE.

Dado que en los trabajos publicados por las dos principales sociedades de Informática tenían bastantes partes en común, deciden aunar esfuerzos y elaborar una única propuesta avalada por ambas sociedades. Así, en 1985 se crea un grupo de trabajo dirigido por Peter Denning y formado por miembros de ACM e IEEE-CS; este grupo publica un informe en diciembre de 1988 [Denning et al., 1988]. Se decide continuar con el trabajo para desarrollar las recomendaciones para un currículo completo, para lo que se crea otro grupo de trabajo conjunto en febrero de 1988 (*The Joint ACM/IEEE-CS Curriculum Task Force*), que da lugar al *Computing Curricula 1991* [Tucker et al., 1991a]. Este currículo es más comprensible que sus predecesores, aunque con un enfoque diferente. Mientras que el *Curriculum '78* y el informe de IEEE-CS de 1983 se centraban en la identificación de programas estándares para cursos individuales, el *Computing Curricula 1991* divide el cuerpo de conocimiento asociado a la Ciencia de la Computación en unidades de conocimiento individuales. Cada una de estas unidades se corresponde con un tema que debe ser cubierto en algún momento en el currículo, aunque las organizaciones tienen la flexibilidad de componer las unidades en cursos como ellas requieran para satisfacer sus necesidades particulares.

A finales de 1998, la *ACM Education Board* y la *Educational Activities Board* de IEEE-CS establecieron un nuevo grupo de trabajo para revisar el *Computing Curricula 1991* [Roberts et al., 1999; Chang et al., 1999], dando lugar de forma definitiva el 15 de diciembre de 2001 al *Computing Curricula 2001* [ACM/IEEE-CS, 2001]. Esta última propuesta intenta reflejar el tremendo avance sufrido por la disciplina en los últimos diez años, llegando a la conclusión que debe haber diferentes propuestas curriculares para la Ciencia de la Computación, la Ingeniería de los Computadores, la Ingeniería del Software y los Sistemas de Información, de forma que el *Computing Curricula 2001* debiera ser la unión de los cuatro. De momento la propuesta recogida en [ACM/IEEE-CS, 2001] se refiere sólo a Ciencia de la Computación.

4.4.1.2 La propuesta conjunta ACM/IEEE-CS (Curricula 91)

La recomendación conjunta (ACM/IEEE-CS 91) [Tucker et al., 1991a; Tucker et al., 1991b; Tucker y Barnes, 1991] incluye un currículo general en Informática, adaptable a las necesidades concretas de cada institución que imparta titulaciones como "*Computer Science*", "*Computer Engineering*", "*Computer Science and Engineering*" y otras similares; pretende ser flexible y adaptarse a los nuevos cambios tecnológicos que se vayan produciendo. El informe constituye un conjunto de guías, más que una prescripción de cursos concretos, en una titulación universitaria de Informática. El contenido de este informe puede resumirse en los siguientes puntos:

- Una identificación de objetivos de un Plan de Estudios universitario en la disciplina de Informática.
- Una definición de qué es la Informática como disciplina vista en forma bidimensional "área/proceso", que comprende la definición de nueve áreas temáticas y tres procesos, uno para cada una de las áreas (Teoría, Abstracción y Diseño).
- Una colección de material curricular avanzado y suplementario que posibilita la profundización en el estudio en algunas de las nueve áreas identificadas. El desarrollo de estos materiales variará en función de los intereses y posibilidades de cada institución que los imparta.
- Un conjunto de consideraciones pedagógicas y curriculares que gobiernen la materialización de los anteriores requisitos comunes y materiales avanzados/suplementarios en una titulación universitaria completa. Se incluyen aspectos como *el papel de los laboratorios, la programación, las matemáticas, conceptos éticos, sociales y profesionales* y hace explícita la noción de los

"**conceptos recurrentes**" comunes a diferentes áreas, independientes de una tecnología en particular y repetidas por toda la disciplina Informática.

A lo largo del documento se insiste en que un currículo es algo más que un conjunto de cursos y por eso tiene que conocerse no solamente la materia básica, sino también tienen que comprenderse los tres procesos o puntos de vista: teoría, abstracción y diseño.

Influencias sobre el *Computing Curricula 91*

Los siguientes puntos resumen los aspectos más significativos de algunos de los trabajos anteriores que han tenido influencia sobre ACM/IEEE-CS91:

- El informe de **ACM de 1968** [ACM, 1968] supone una de las primeras tentativas de definir el ámbito de la Informática como disciplina. Además de incluir un currículo, define los principales campos de la Informática; concretamente identifica tres áreas: *estructuras de información y procesos, sistemas de proceso de información* y ***metodologías***. Propone un currículo central compuesto de cuatro cursos básicos (algoritmos y programación, estructura de computadores y sistemas, estructuras discretas y cálculo numérico) y de cuatro cursos más avanzados (estructuras de datos, lenguajes de programación, organización de computadores y programación de sistemas). Estos cursos enfatizaban el análisis numérico y el hardware, omitiendo la Ingeniería del Software [Tucker y Wegner, 1994].

- El informe de **ACM de 1978** [Austing et al., 1979], proporciona descripciones detalladas de cursos universitarios en Informática, subrayando el término "*Computer Science*" (al igual que el anterior de 1968) y la importancia de la programación. La visión aquí es muy dependiente de la máquina, solamente en uno de los cursos "*opcionales*" avanzados se hace referencia a un curso de "*Diseño y Desarrollo de Software*" que podría incluirse en el currículo; incluye técnicas *top-down* y estructuradas de diseño, formación de equipos de desarrollo y gestión de proyectos. En este momento, se critica el carácter que ha tomado el *Curriculum'78*, por pensarse que se está equiparando Ciencia de la Computación a Programación [Ralston y Shaw, 1980].

- El informe de **la Educational Activities Board de IEEE-CS de 1983**, que describe una serie de áreas para Informática, considerando la orientación *Computer Science and Engineering*. En este informe se hacen recomendaciones sobre material de laboratorio y sobre el propio laboratorio como soporte a las clases teóricas; utiliza una estructura modular para organizar los temas y construir los

cursos, proponiéndose como una base para configurar Planes de Estudios adaptados a distintas situaciones y centros particulares. En este informe se introduce la **Ingeniería del Software** en varios niveles: *como área temática básica*, *como Laboratorio específico de Ingeniería del Software* y *como área temática avanzada*, con un carácter dominado por las técnicas y métodos estructurados, propios de la época, pero subrayando ya la importancia de la fase de requisitos, modelado conceptual, prototipado y validación y verificación.

- El informe de 1989 "*Computing as a discipline*" [Denning et al., 1989], defiende la integración del trabajo de laboratorio con las clases teóricas, subrayando la importancia de introducir aspectos de *diseño* en el currículo y propone que en los cursos introductorios se dé una visión global de toda la carrera, incidiendo en los conceptos fundamentales, mientras que en los cursos superiores se debe ir profundizando en cada uno de esos temas. En este informe se proponen ya las nueve áreas temáticas que, posteriormente, aparecen en el documento ACM/IEEE-CS 91, distinguiendo para cada una de ellas ***Teoría*** (matemáticas subyacentes), ***Abstracción*** (modelado, análisis) y ***Diseño*** (especificar soluciones, estudiar alternativas).

Objetivos del programa de graduación. Perfil de los graduados

El programa debe preparar a los estudiantes para la comprensión de las materias relacionadas con la computación en dos aspectos: *como una disciplina académica* y *como una profesión dentro de su contexto social*. Los estudiantes deben adquirir las habilidades para poder mantenerse actualizados y evaluar las ideas nuevas. Entre estas habilidades se incluyen el leer publicaciones, la asistencia a seminarios y la evaluación de su contenido, así como el trabajo en equipo en proyectos relacionados con problemas de actualidad.

En consecuencia, el primer objetivo del programa de formación ha de ser el proporcionar una cobertura básica, amplia y coherente de la disciplina de la computación. Los estudiantes deben comprender las relaciones existentes entre las diferentes áreas de la computación.

El programa debe preparar a los estudiantes para aplicar sus conocimientos a problemas específicos y con restricciones para elaborar soluciones. Esto incluye la *capacidad de definir un problema claramente, determinar su posibilidad de tratamiento, determinar la oportunidad de consultar con expertos, evaluar y elegir una estrategia de solución adecuada; estudiar, especificar, diseñar, implementar, probar, modificar y documentar esa solución, evaluar alternativas y realizar análisis de riesgos del diseño, integrar tecnologías alternativas en la*

solución y comunicar la solución tanto a los compañeros, a los profesionales de otros campos como al público en general. Esto incluye la capacidad de trabajo en equipo durante todo el proceso de resolución de problemas.

El programa debe proporcionar la suficiente exposición del amplio cuerpo de teoría que subyace en el campo de la computación, de forma que los estudiantes aprecien la profundidad intelectual y los elementos abstractos que continuarán retando a los investigadores en el futuro.

Los graduados tienen que tener presente la alta tasa de cambio tecnológico, la tasa de crecimiento relativo en la teoría de la computación y la interacción delicada que tiene lugar entre las dos.

Principios subyacentes en el diseño curricular

En esta propuesta curricular se identifican nueve áreas de conocimiento y tres procesos que caracterizan las diferentes metodologías operativas utilizadas en la investigación y el desarrollo de la computación.

Cada una de las áreas identificadas tiene una base teórica significativa, abstracciones significativas y realizaciones significativas de diseño e implementación. Las nueve áreas son: *Algoritmos y estructuras de datos; Arquitectura; Inteligencia artificial y robótica; Bases de datos y recuperación de la información; Comunicación hombre-máquina; Computación numérica y simbólica; Sistemas operativos; Lenguajes de Programación* y ***Metodología e Ingeniería del Software***. De todas ellas las relacionadas con este Proyecto Docente son:

- ***Comunicación Hombre-Máquina***. El propósito principal de esta área es la transferencia eficiente de información entre el hombre y la máquina. Se incluyen gráficos, factores humanos que afectan a la interacción eficiente, así como la organización y visualización de la información para una utilización efectiva por parte de las personas.

- ***Metodología e Ingeniería del Software***. El propósito principal de esta área es la especificación, el diseño y la producción de grandes sistemas software. El interés se centra en los principios de programación y del desarrollo del software, la verificación y la validación del software y la especificación y producción de sistemas software que sean seguros y fiables.

La Informática es vista simultáneamente como una disciplina Matemática, Científica y de Ingeniería. Los diferentes profesionales en cada una de las áreas emplean diferentes

metodologías operativas, o procesos, en el curso de sus trabajos de investigación, desarrollo y aplicación.

El primero de estos procesos es el llamado **teoría**, está fundamentado en las Matemáticas, y se utiliza en el desarrollo de teorías matemáticas coherentes. Los principales elementos que componen este proceso son: *definiciones y axiomas*, *teoremas*, *pruebas* e *interpretación de resultados*.

El segundo proceso, denominado **abstracción**, está enraizado en las Ciencias Experimentales, constando de los siguientes elementos: *recolección de datos y formulación de hipótesis*, *modelado y predicción*, *diseño de un experimento* y *análisis de resultados*. Cuando las personas hacen abstracción están modelando algoritmos, estructuras de datos o arquitecturas, por ejemplo; prueban hipótesis acerca de esos modelos, toman decisiones de diseño alternativas... A los estudiantes hay que introducirlos en la abstracción a través de las clases y de los laboratorios.

El tercer proceso, llamado **diseño**, está enraizado en la Ingeniería y se utiliza en el desarrollo de un sistema o dispositivo para resolver un determinado problema. Consta de las siguientes partes: *requisitos*, *especificaciones*, *diseño*, *implementación* y *pruebas*. Cuando los profesionales de la computación diseñan, han de implicarse en la conceptualización y la realización de sistemas en el contexto de las restricciones del mundo real. Los estudiantes aprenden diseño mediante la experiencia directa y mediante el estudio de los diseños de otros. Los proyectos de laboratorio se orientan hacia el diseño, proporcionando a los estudiantes una experiencia de primera mano en el desarrollo de un sistema o un componente de un sistema para resolver un problema particular. Estos proyectos de laboratorio enfatizan en la síntesis de las soluciones prácticas a problemas y, por tanto, obligan a los estudiantes a evaluar las alternativas, costes y rendimientos en el contexto de las restricciones del mundo real. Los estudiantes desarrollan la capacidad de realizar estas evaluaciones viendo y discutiendo ejemplos de diseños, así como recibiendo información sobre sus propios diseños.

Conceptos recurrentes

Existen ciertos conceptos fundamentales que aparecen de forma recurrente en el diseño de los currículos de computación, los cuales representan un papel importante en el diseño de los cursos individuales. Estos conceptos son *ideas*, *materias*, *principios* y *procesos* que ayudan a unificar una disciplina académica en su substrato. En la propuesta curricular ACM/IEEE-CS91 se han identificado doce conceptos, a saber:

- **Ligadura**. El proceso de concretar abstracciones mediante la asociación de propiedades adicionales a dicha abstracción. Por ejemplo, la asociación de procesos a procesadores o la creación de instancias concretas a partir de descripciones abstractas.
- **Complejidad de los grandes problemas**. Los efectos del crecimiento no lineal de la complejidad cuando crece el tamaño del problema.
- **Modelos conceptuales y formales**. Diferentes modos de formalizar, caracterizar, visualizar y concebir ideas o problemas. Los ejemplos incluyen modelos conceptuales del tipo de los tipos abstractos de datos y los modelos semánticos, y lenguajes visuales utilizados en la especificación y diseño de sistemas, tales como flujos de datos y diagramas entidad-relación.
- **Consistencia y compleción**. Incluye la consistencia de un conjunto de axiomas que sirven como especificación formal, la consistencia de la teoría con los hechos observados y la consistencia interna de un lenguaje. La compleción incluye la suficiencia de un conjunto de axiomas dados para capturar todos los comportamientos que se desea, la suficiencia funcional de los sistemas software y hardware, así como la capacidad de un sistema para comportarse adecuadamente en condiciones de error o situaciones no previstas.
- **Eficiencia**. La medida de los costes relativos de los recursos tales como espacio, tiempo, dinero o personal.
- **Evolución**. El hecho del cambio y sus implicaciones.
- **Niveles de abstracción**. La naturaleza y uso de la abstracción en computación; la utilización de abstracciones para manejar la complejidad, estructurar sistemas, ocultar detalles y capturar patrones recurrentes. La capacidad de representar una entidad o sistema por abstracción tiene diferentes niveles de detalle y especificidad.
- **Ordenación en el espacio**. De los conceptos de localización y proximidad en la disciplina de la Informática. Además de la localización física, como en las redes o en la memoria de un computador, incluye el emplazamiento de la organización (por ejemplo, de procesadores y procesos; definiciones de tipo y las operaciones asociadas) y la localización conceptual (por ejemplo, el ámbito del software, la cohesión y el acoplamiento).
- **Ordenación en tiempo**. El concepto de tiempo en la ordenación de los eventos. Esto incluye el tiempo *como un parámetro en los modelos formales* (como por

ejemplo en la lógica temporal), *como sinónimo de sincronización de procesos* o *como una parte esencial en la ejecución de los algoritmos*.

- **Reutilización**. La capacidad de una técnica, concepto o componente de sistema para ser reutilizado en un nuevo contexto o situación.
- **Seguridad**. La capacidad de los sistemas software y hardware para responder y defenderse de las peticiones no autorizadas, inapropiadas o imprevistas.
- **Decisiones y consecuencias**. El fenómeno de las decisiones en Informática y de las consecuencias que acarrean; los efectos técnicos, económicos, culturales que se derivan de la selección de una determinada alternativa de diseño.

El papel de los laboratorios

Un currículo de formación en Informática se compone, idealmente, de un programa integrado de clases teóricas y experiencias de laboratorio.

El papel de los laboratorios es muy importante en el diseño de un currículo en formación Informática, presentando las siguientes características y ventajas:

- Los laboratorios permiten demostrar la aplicación de los principios en el diseño, implantación y prueba de los sistemas software.
- Los laboratorios enfatizan las técnicas que utilizan las herramientas actuales y conducen hacia métodos experimentales adecuados incluyendo la presentación oral y escrita de los hallazgos.
- Los ejercicios de laboratorio han de estar diseñados para mejorar la experiencia del estudiante en las metodologías del software mediante el desarrollo de diseños e implantaciones.
- Las experiencias del laboratorio incrementan la capacidad para resolver problemas, las habilidades analíticas y la capacitación profesional.

Se diferencian dos tipos de laboratorios, *laboratorios abiertos* y *laboratorios cerrados*. Los primeros consisten en una asignación de trabajo que se llevará a cabo sin supervisión. Un laboratorio cerrado conlleva la asignación de trabajo de forma planificada, estructurada y supervisada. La finalización de un trabajo de laboratorio debe de estar acompañada de un informe oral y/o escrito por parte del estudiante.

La unidad de conocimiento de Metodología e Ingeniería del Software

El currículo en Informática se organiza sobre la base de unidades del conocimiento. Se entiende por unidad de conocimiento *la designación de una colección coherente de materias dentro de una de las nueve áreas principales de la disciplina de la computación* [Tucker et al., 1991a].

Para cada unidad de conocimiento se especifican los contenidos bajo el epígrafe de materias, los laboratorios propuestos, la posible relación con otras unidades de conocimiento, las unidades de conocimiento que son prerrequisito y si la unidad del conocimiento es requisito para otras unidades de conocimiento.

Las unidades de conocimiento que se definen en la propuesta curricular ACM/IEEE-CS91, relacionadas con el presente Proyecto Docente caen dentro del área de Metodología e Ingeniería del Software; cuyas unidades de conocimiento a su vez se recogen en la Tabla 4.12.

SE: Metodología e Ingeniería del Software (se recomiendan unas 44 horas teóricas)
SE1: Conceptos fundamentales para la resolución de problemas
SE2: El proceso de desarrollo del software
SE3: Requisitos y especificaciones del software
SE4: Diseño e implementación del software
SE5: Verificación y validación

Tabla 4.12. Unidades de conocimiento para el área de Metodología e Ingeniería del Software

De las unidades de conocimiento de esta área se describen aquéllas que se estima que deben estar contempladas en las asignaturas que se ajustan al perfil y al entorno en el que se va a desarrollar la docencia objeto de la plaza a concurso.

- **SE2.- El proceso de desarrollo del software.**

  Introducción a los modelos y elementos relacionados con el desarrollo de software de calidad. Utilización de entornos y herramientas que faciliten el diseño y la implantación de grandes sistemas software. El papel y la utilización de los estándares.

  1) Materias.

  1. Modelos del ciclo de vida del desarrollo del software.
  2. Objetivos del diseño del software.
  3. Documentación.
  4. Gestión y control de configuraciones.
  5. Elementos de la fiabilidad del software.
  6. Mantenimiento.

7. Herramientas de especificación y diseño, herramientas de implementación.

2) Laboratorios (abiertos).

a. Implementar un prototipo para una especificación determinada.

b. Dado un diseño de software y una implementación intermedia de un desarrollo iterativo, completar la implementación correspondiente a la siguiente iteración.

c. Crítica de un conjunto de documentación determinado.

d. Dada una implementación, unas especificaciones y un conjunto de nuevas especificaciones, modificar el código de acuerdo a las nuevas especificaciones.

3) Relacionado con: Especificaciones y requisitos del software.

4) Prerrequisitos: Tipos abstractos de datos.

5) Requisito para: Diseño e implementación del software.

- **SE3.- Especificaciones y requisitos del software.**

  Introducción al desarrollo de especificaciones formales e informales para la definición de los requisitos de un sistema software.

  1) Materias.

  a. Especificaciones informales.

  b. Especificaciones formales, especificaciones algebraicas de los TAD, precondiciones y poscondiciones.

  2) Laboratorios.

  a. Producir un documento de análisis de requisitos.

  b. Producir un documento con las especificaciones formales correspondientes a un conjunto de especificaciones informales.

  3) Relacionado con: El proceso de desarrollo del software, control de tipos, semántica de los lenguajes.

  4) Prerrequisitos: Tipos abstractos de datos.

  5) Requisito para: Diseño e implementación del software, verificación y validación del software.

- **SE4.- Diseño e implementación del software.**

  Introducción a los paradigmas principales que rigen el diseño y la implementación de grandes sistemas software.

  1) Materias.

  a. Diseño dirigido por funciones/procesos.

b. Diseño ascendente, soporte para reutilización.
c. Estrategias de implementación (por ejemplo, ascendente, descendente, equipos)
d. Elementos de la implementación, mejora de rendimientos, depuración.

2) Laboratorios (abiertos).

a. Realizar un diseño objeto para un conjunto de especificaciones.
b. Implementar el diseño anterior.
c. Dado una especificación de problema y un conjunto de módulos ejecutables con sus especificaciones, realizar un diseño ascendente, reutilizando lo más posible.
d. Realizar una implementación descendente para un diseño software dado.

3) Relacionado con: Bases de datos, paradigmas de programación.

4) Prerrequisitos: IS2, IS3.

Otras consideraciones

La propuesta curricular ACM/IEEE-CS91 propone un curso optativo de nivel avanzado en Ingeniería del Software, centrado en los métodos y herramientas necesarios para incrementar la calidad, además de reducir el coste y la complejidad de mantenimiento de los sistemas software. Este curso tiene como prerrequisitos, además de diversas unidades de las áreas de algoritmos y estructuras de datos, cálculo numérico, lenguajes de programación y asuntos tanto éticos como sociales de la profesión, así como todo el área de metodología e Ingeniería del Software.

Se incluye una propuesta detallada de un programa universitario en Informática con énfasis en la Ingeniería del Software (*Implementation G: A Program in Computer Science – Software Engineering Emphasis*) [Tucker et al., 1991a].

En cuanto a la tecnología de objetos, cabe decir que tiene muy poca presencia en esta propuesta curricular (unas diez horas teóricas divididas en cuatro áreas [Osborne, 1992]), quizás porque ésta se centra en describir de una forma general el esqueleto básico de la Informática como disciplina, más que en establecer su cuerpo de conocimiento.

Aunque no es perfecta, esta propuesta curricular ha sido la que mayor influencia ha tenido en un plano internacional, incluyendo a España. Durante todos los años que ha estado en vigor el *Computing Curricula 91* se han propuesto diferentes modificaciones para incorporar o enfatizar diferentes aspectos, como la tecnología de objetos, la Ingeniería del Software o los Sistemas de Información; algunas de estas propuestas son: [Scragg et al., 1994; Knight et al., 1994; Shackelford y LeBlanc, 1994; Hirmanpour et al., 1995; Reynolds y Fox, 1996; Pham, 1997; Jackson et al., 1997].

4.4.1.3 La propuesta conjunta ACM/IEEE-CS (Curricula 2001)

Reconocidas las limitaciones del *Computing Curricula 91* y el tremendo avance de la Informática en estos años, de nuevo ACM e IEEE-CS se unen para crear un nuevo grupo de trabajo que defina una nueva propuesta curricular en el campo de la Ciencia de la Computación [ACM/IEEE-CS, 2001].

Principios

Sobre la base del análisis de pasadas propuestas curriculares y de los cambios técnicos y culturales sufridos por la Informática, el equipo de trabajo del *Computing Curricula 2001* ha formulado los siguientes principios que guían su trabajo:

1. La Informática es un campo extenso que se extiende más allá de los límites de la Ciencia de la Computación.
2. La Ciencia de la Computación basa sus fundamentos en una amplia variedad de disciplinas.
3. La rápida evolución de la Ciencia de la Computación requiere una revisión continua del currículo.
4. El desarrollo de un currículo propio de la Ciencia de la Computación debe ser sensitivo a los cambios en la tecnología, a los nuevos desarrollos en Pedagogía y a la importancia del aprendizaje continuo durante toda la vida.
5. El *Computing Curricula 2001* debe ir más lejos de unas unidades de conocimiento para ofrecer una guía significativa en términos de diseño de cursos individuales.
6. El *Computing Curricula 2001* debe identificar las propiedades fundamentales y el conocimiento que todos los estudiantes deben poseer.
7. El cuerpo de conocimiento requerido debe hacerse lo más pequeño posible.
8. El *Computing Curricula 2001* debe tener un ámbito internacional.
9. El desarrollo del *Computing Curricula 2001* debe ser lo más abierto posible.
10. El *Computing Curricula 2001* debe incluir la práctica profesional como un componente integrado en el currículo de los no graduados.
11. El *Computing Curricula 2001* debe incluir discusiones de estrategias y tácticas para la implementación de recomendaciones de alto nivel.

El cuerpo de conocimiento de la Ciencia de la Computación

En el *Computing Curricula 2001* el número de áreas pasa de las nueve del *Computing Curricula 91* a catorce (Tabla 4.13), habiendo sido asignada cada una de ellas a un grupo de trabajo distinto para que definiera el cuerpo de conocimiento asociado a cada área.

El cuerpo de conocimiento se organiza jerárquicamente en tres niveles. El nivel más alto es el **área**, que representa un campo disciplinar particular. Casa área se identifica por dos letras. Las áreas se dividen en divisiones más pequeñas llamadas **unidades**. Cada unidad representa un módulo temática individual dentro de un área. Cada unidad está dividida en **temas** o **tópicos** que son el nivel más bajo de la jerarquía.

Como uno de los objetivos era que el cuerpo de conocimiento fuera tan pequeño como fuera posible, se ha definido un núcleo mínimo consistente en aquellas unidades en las que había un amplio consenso en que el material que representan es esencial para cualquiera que quiera obtener el grado en el campo. Las unidades que forman parte del programa pero que quedan fuera del núcleo son opcionales. Esta división en unidades troncales y opcionales lleva a las siguientes conclusiones:

1. El núcleo se refiere a aquellas unidades que se requieren para que los estudiantes adquieran un grado en Informática.
2. El núcleo no es un currículo completo.
3. El núcleo debe ser completado con material adicional.
4. Las unidades troncales no son necesariamente aquéllas que aparecen en un conjunto de cursos introductorios al comienzo de los estudios.

Para tener una métrica de la cobertura de las unidades se ha elegido la **hora**, correspondiéndose con el tiempo de clase requerido para presentar el material en un formato de clase tradicional. Las unidades troncales tienen asignadas un mínimo de horas lectivas. Esta métrica se ha elegido por motivos de consistencia con anteriores informes. Esta asignación temporal no incluye el tiempo de dedicación fuera de clase, que se considera que es el triple del asignado a cada unidad. Ahora bien, no existe ninguna relación entre las horas de las unidades troncales y la distribución que se puede hacer de éstas para conformar una determinada asignatura o diseñar una propuesta curricular concreta.

El informe considera que una asignatura cuatrimestral puede estar formada por unas 45 sesiones, cada una con una duración entre 50 minutos y una hora. Restando el tiempo para exámenes, las asignaturas durarían unas 40 horas.

En la Figura 4.8 se recoge el cuerpo de conocimiento del *Computing Curricula 2001* expresado en áreas y unidades troncales (subrayadas) y optativas.

Área	Código
Estructuras Discretas	DS
Fundamentos de Programación	PF
Algoritmos y Complejidad	AL
Arquitectura y Organización	AR
Sistemas Operativos	OS
Redes	NC
Lenguajes de Programación	LP
Interacción Persona-Ordenador	HC
Gráficos y Computación Visual	GV
Sistemas Inteligentes	IS
Gestión de la Información	IM
Aspectos Sociales y Profesionales	SP
Ingeniería del Software	**SE**
Ciencia de la Computacional y Métodos Numéricos	CN

Tabla 4.13. Áreas de interés en el *Computing Curricula 2001* [ACM/IEEE-CS, 2001]

Finalmente las unidades se empaquetan en cursos. Los cursos se dividen en el *Computing Curricula 2001* en tres categorías de acuerdo al nivel. Los cursos introductorios son típicamente cursos de nivel básico ofertados el primero y el segundo año de carrera. Los cursos intermedios construyen los fundamentos para un mayor y posterior estudio en el campo, suceden típicamente el segundo y tercer año. Los cursos avanzados se dan en los últimos años y se centran en aquellos temas que requieren una preparación significativa en términos de cursos preliminares.

Las principales diferencias en lo tocante a las áreas entre el *Computing Curricula 1991 y el Computing Curricula 2001*, a parte del número, se pueden resumir en los siguientes puntos:

- *Se han añadido las Estructuras Discretas (DS) como un área separada.* En el *Computing Curricula 1991*, la Matemática Discreta aparece sólo como un prerrequisito para los tópicos que requieren una madurez matemática. El *Computing Curricula 2001* ha decidido enfatizar la dependencia de la Informática de la Matemática Discreta, incluyéndola como una nueva área.

- *La necesidad de incluir la instrucción en el uso de un lenguaje de programación se ha explicitado añadiendo un área específica sobre Fundamentos de Programación. (PF).* El *Computing Curricula 1991* incluye un área denominada “Introducción a un Lenguaje de Programación”, pero se identificaba como un componente opcional en el currículo.

Se esperaba que los alumnos adquiriesen los conocimientos básicos sobre programación de ordenadores en el Instituto, pero esto no ha llegado a ocurrir, lo que ha provocado la creación de esta área.

- *El área sobre los Aspectos Sociales, Éticos y Profesionales (SP) se ha integrado en el currículo, dándole la misma importancia que las demás áreas.* Esto se ha producido debido a que, desde la publicación del *Computing Curricula 1991*, se ha producido un consenso creciente en el hecho de que todos los estudiantes deben ser conscientes del impacto social de su trabajo y de las responsabilidades éticas de ser un profesional de la Informática.

- *Gráficos y Computación Visual (GV) y Redes (NC) se han añadido como áreas separadas.* Muchas de las áreas se han expandido en gran medida desde 1991. Como resultado, algunas de ellas que antes simplemente eran temas de un área más general, han crecido tanto que ya no cuadran adecuadamente en la estructura anterior.

- *Cuatro áreas, la de Algoritmos y estructuras de Datos (AL), Inteligencia Artificial y Robótica (AI), Bases de Datos y Recuperación de Información (DB) y Computación Numérica y Simbólica (NU), cambian significativamente de nombre, pasando a denominarse Algoritmos y Complejidad (AL), Sistemas Inteligentes (IS), Gestión de la Información (IM) y Ciencia Computacional (CN).* La justificación de este cambio de nombres es la de identificar mejor los nuevos contenidos que se han añadido al área a lo largo de los diez años transcurridos.

DS. Discrete Structures (43 core hours)
- DS1. Functions, relations, and sets (6)
- DS2. Basic logic (10)
- DS3. Proof techniques (12)
- DS4. Basics of counting (5)
- DS5. Graphs and trees (4)
- DS6. Discrete probability (6)

PF. Programming Fundamentals (38 core hours)
- PF1. Fundamental programming constructs (9)
- PF2. Algorithms and problem-solving (6)
- PF3. Fundamental data structures (14)
- PF4. Recursion (5)
- PF5. Event-driven programming (4)

AL. Algorithms and Complexity (31 core hours)
- AL1. Basic algorithmic analysis (4)
- AL2. Algorithmic strategies (6)
- AL3. Fundamental computing algorithms (12)
- AL4. Distributed algorithms (3)
- AL5. Basic computability (6)
- AL6. The complexity classes P and NP
- AL7. Automata theory
- AL8. Advanced algorithmic analysis
- AL9. Cryptographic algorithms
- AL10. Geometric algorithms
- AL11. Parallel algorithms

AR. Architecture and Organization (36 core hours)
- AR1. Digital logic and digital systems (6)
- AR2. Machine level representation of data (3)
- AR3. Assembly level machine organization (9)
- AR4. Memory system organization and architecture (5)
- AR5. Interfacing and communication (3)
- AR6. Functional organization (7)
- AR7. Multiprocessing and alternative architectures (3)
- AR8. Performance enhancements
- AR9. Architecture for networks and distributed systems

OS. Operating Systems (18 core hours)
- OS1. Overview of operating systems (2)
- OS2. Operating system principles (2)
- OS3. Concurrency (6)
- OS4. Scheduling and dispatch (3)
- OS5. Memory management (5)
- OS6. Device management
- OS7. Security and protection
- OS8. File systems
- OS9. Real-time and embedded systems
- OS10. Fault tolerance
- OS11. System performance evaluation
- OS12. Scripting

NC. Net-Centric Computing (15 core hours)
- NC1. Introduction to net-centric computing (2)
- NC2. Communication and networking (7)
- NC3. Network security (3)
- NC4. The web as an example of client-server computing (3)
- NC5. Building web applications
- NC6. Network management
- NC7. Compression and decompression
- NC8. Multimedia data technologies
- NC9. Wireless and mobile computing

PL. Programming Languages (21 core hours)
- PL1. Overview of programming languages (2)
- PL2. Virtual machines (1)
- PL3. Introduction to language translation (2)
- PL4. Declarations and types (3)
- PL5. Abstraction mechanisms (3)
- PL6. Object-oriented programming (10)
- PL7. Functional programming
- PL8. Language translation systems
- PL9. Type systems
- PL10. Programming language semantics
- PL11. Programming language design

Note: The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.

HC. Human-Computer Interaction (8 core hours)
- HC1. Foundations of human-computer interaction (6)
- HC2. Building a simple graphical user interface (2)
- HC3. Human-centered software evaluation
- HC4. Human-centered software development
- HC5. Graphical user-interface design
- HC6. Graphical user-interface programming
- HC7. HCI aspects of multimedia systems
- HC8. HCI aspects of collaboration and communication

GV. Graphics and Visual Computing (3 core hours)
- GV1. Fundamental techniques in graphics (2)
- GV2. Graphic systems (1)
- GV3. Graphic communication
- GV4. Geometric modeling
- GV5. Basic rendering
- GV6. Advanced rendering
- GV7. Advanced techniques
- GV8. Computer animation
- GV9. Visualization
- GV10. Virtual reality
- GV11. Computer vision

IS. Intelligent Systems (10 core hours)
- IS1. Fundamental issues in intelligent systems (1)
- IS2. Search and constraint satisfaction (5)
- IS3. Knowledge representation and reasoning (4)
- IS4. Advanced search
- IS5. Advanced knowledge representation and reasoning
- IS6. Agents
- IS7. Natural language processing
- IS8. Machine learning and neural networks
- IS9. AI planning systems
- IS10. Robotics

IM. Information Management (10 core hours)
- IM1. Information models and systems (3)
- IM2. Database systems (3)
- IM3. Data modeling (4)
- IM4. Relational databases
- IM5. Database query languages
- IM6. Relational database design
- IM7. Transaction processing
- IM8. Distributed databases
- IM9. Physical database design
- IM10. Data mining
- IM11. Information storage and retrieval
- IM12. Hypertext and hypermedia
- IM13. Multimedia information and systems
- IM14. Digital libraries

SP. Social and Professional Issues (16 core hours)
- SP1. History of computing (1)
- SP2. Social context of computing (3)
- SP3. Methods and tools of analysis (2)
- SP4. Professional and ethical responsibilities (3)
- SP5. Risks and liabilities of computer-based systems (2)
- SP6. Intellectual property (3)
- SP7. Privacy and civil liberties (2)
- SP8. Computer crime
- SP9. Economic issues in computing
- SP10. Philosophical frameworks

SE. Software Engineering (31 core hours)
- SE1. Software design (8)
- SE2. Using APIs (5)
- SE3. Software tools and environments (3)
- SE4. Software processes (2)
- SE5. Software requirements and specifications (4)
- SE6. Software validation (3)
- SE7. Software evolution (3)
- SE8. Software project management (3)
- SE9. Component-based computing
- SE10. Formal methods
- SE11. Software reliability
- SE12. Specialized systems development

CN. Computational Science (no core hours)
- CN1. Numerical analysis
- CN2. Operations research
- CN3. Modeling and simulation
- CN4. High-performance computing

Figura 4.8. Cuerpo de conocimiento del *Computing Curricula 2001* [ACM/IEEE-CS, 2001]

La unidad de conocimiento de Ingeniería del Software

En la Tabla 4.14 se recogen las unidades para el área de conocimiento de Ingeniería del Software en el *Computing Curricula 2001*. Se sigue el convenio de subrayar aquellas unidades que se consideran troncales.

SE: Ingeniería del Software (31 horas troncales)
SE1: Diseño del software
SE2: Uso de APIs
SE3: Herramientas y entornos software
SE4: Proceso software
SE5: Requisitos software y especificaciones
SE6: Validación del software
SE7: Evolución del software
SE8: Gestión de proyectos software
SE9: Desarrollo basado en componentes
SE10: Métodos formales
SE11: Confiabilidad del software
SE12: Desarrollo de sistemas especializados

Tabla 4.14. Unidades de conocimiento para el área de Ingeniería del Software

En el *Computing Curricula 2001* la Ingeniería del Software se entiende como la disciplina relacionada con la aplicación de la teoría, conocimiento y práctica a la construcción efectiva y eficiente de sistemas software que satisfagan los requisitos de los usuarios y clientes. La Ingeniería del Software es aplicable a cualquier sistema, independientemente de su tamaño. La Ingeniería del Software comprende todas las fases del ciclo de vida de un sistema software. La Ingeniería del Software emplea método de ingeniería, procesos, técnicas y métricas. Se beneficia del uso de herramientas para la gestión del desarrollo del software.

Las unidades del área de conocimiento se reparten por los diferentes cursos introductorios e intermedios propuestos en el *Computing Curricula 2001*, y además se proponen (pero no se detallan) una serie de cursos avanzados sobre Ingeniería del Software, los cuales se recogen en la Tabla 4.15.

A continuación se van a exponer los tópicos recomendados en cada una de las unidades del área de Ingeniería del Software.

- **SE1.- Diseño del software.**

  - Mínimo de tiempo: 8 horas.

  - Tópicos:

1. Conceptos y principios de diseño.
2. Patrones de diseño.
3. Arquitectura software.
4. Diseño estructurado.
5. Análisis y diseño orientado a objetos.
6. Diseño de componentes.
7. Diseño para reutilización.

- **SE2.- Uso de APIs.**

  - Mínimo de tiempo: 5 horas.

  - Tópicos:

    1. Programación con APIs.
    2. Inspectores (navegadores) de clases y herramientas relacionadas.
    3. Programación por ejemplo.
    4. Depuración en el entorno de API.
    5. Introducción a la programación basada en componentes.

- **SE3.- Herramientas y entornos software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Entornos de desarrollo.
    2. Herramientas para el análisis de requisitos y el modelado en diseño.
    3. Herramientas de prueba.
    4. Herramientas para la gestión de la configuración.
    5. Mecanismos para la integración de herramientas.

- **SE4.- Proceso software.**

  - Mínimo de tiempo: 2 horas.

  - Tópicos:

    1. Ciclo de vida del software y modelos de procesos.
    2. Modelos de seguimiento de procesos.
    3. Métricas para proceso software

- **SE5.- Requisitos software y especificaciones.**

  - Mínimo de tiempo: 4 horas.

  - Tópicos.

1. Obtención (elicitación) de requisitos.
2. Técnicas de modelado en el análisis de requisitos.
3. Requisitos funcionales y no funcionales.
4. Prototipado.
5. Conceptos básicos de las técnicas de especificación formal.

- **SE6.- Validación del software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Planificación de la validación.
    2. Fundamentos de la prueba del software.
    3. Técnicas de caja negra y de caja blanca.
    4. Pruebas de unidad, de integración, de validación y de sistema.
    5. Pruebas orientadas a objetos.
    6. Inspecciones.

- **SE7.- Evolución del software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Mantenimiento del software.
    2. Características del software mantenible.
    3. Reingeniería.
    4. Sistemas legados.
    5. Reutilización del software.

- **SE8.- Gestión de proyectos software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Gestión de equipos.
    2. Planificación.
    3. Técnicas de estimación y medida del software.
    4. Análisis de riesgos.
    5. Aseguramiento de la calidad del software.
    6. Gestión de la configuración del software
    7. Herramientas de gestión de proyectos.

- **SE9.- Desarrollo basado en componentes [opcional].**

  - Tópicos:

    1. Fundamentos.
    2. Técnicas básicas.
    3. Aplicaciones.
    4. Arquitectura de los sistemas basados en componentes.
    5. Diseño orientado a componentes.
    6. Manejo de eventos.
    7. *Middleware*.

- **SE10.- Métodos formales [opcional].**

  - Tópicos:

    1. Conceptos sobre métodos formales.
    2. Lenguajes de especificación formal.
    3. Especificaciones ejecutables y no ejecutables.
    4. Pre y post aserciones.
    5. Verificación formal.

- **SE11.- Confiabilidad del software [opcional].**

  - Tópicos:

    1. Modelos de confiabilidad del software.
    2. Redundancia y tolerancia a fallos.
    3. Clasificación de defectos.
    4. Métodos probabilísticas de análisis.

- **SE12.- Desarrollo de sistemas especializados [opcional].**

  - Tópicos:

    1. Sistemas en tiempo real.
    2. Sistemas cliente-servidor.
    3. Sistemas distribuidos.
    4. Sistemas paralelos.
    5. Sistemas basados en la web.
    6. Sistemas altamente integrados.

Cursos avanzados en el área de Ingeniería del Software
CS390 Desarrollo de software avanzado
CS301 Ingeniería del Software
CS392 Diseño de software
CS393 Ingeniería del Software y especificación formal
CS394 Ingeniería del Software empírica
CS395 Mejora del proceso software
CS396 Desarrollo basado en componentes
CS397 Entornos de programación
CS398 Sistemas de seguridad crítica

Tabla 4.15. Propuesta de cursos avanzados en Ingeniería del Software

El área de conocimiento de Interacción Persona-Ordenador (HC) y la unidad de Modelado de datos (IM3) perteneciente al área de conocimiento de Gestión de la Información (IM) guardan una estrecha relación con el área de conocimiento de Ingeniería del Software (SE).

Concretamente el campo de la Interacción Persona-Ordenador ha sufrido un avance considerable desde que en 1995 Sutcliffe en el prefacio de la segunda edición de su libro dedicado a la Interacción Persona-Ordenador [Sutcliffe, 1995] argumentaba que: "*en 1988 había muy pocos libros dedicados a la Interacción Persona-Ordenador... y el tema es tenido en cuenta en muy pocos cursos de Ciencia de la Computación*". En 1995, cuando aparece su libro existe una oferta más considerable de libros en esta parcela, lo que denota su reconocimiento, y hoy en día aparece como un área de conocimiento independiente en el *Computer Curricula 2001*. Pero además, el campo de la Interacción Persona-Ordenador cuenta con varias propuestas curriculares [Hefley et al., 1992; Kirby et al., 1994; Kirby et al., 1995; BCS, 1995] y con voces que piden una mayor integración de ésta con la Ingeniería del Software, presentando la figura del *ingeniero en usabilidad* [Dowell y Long, 1989; Faulkner y Culwin, 2000].

Plan de Estudios y la Ingeniería del Software

Para organizar las unidades y materias en una propuesta curricular concreta, el *Computing Curricula 2001* establece tres niveles de asignaturas para una titulación: asignaturas de iniciación, correspondientes al primero o segundo cuatrimestre de la titulación; asignaturas intermedias, correspondientes al segundo o tercer cuatrimestre y que establecen los fundamentos para estudios más avanzados de especialización; y las asignaturas avanzadas, que se imparten en cuatrimestres posteriores y se centran en las materias que requieren un estudio más exhaustivo en relación con el trabajo realizado en cursos anteriores.

La organización de los contenidos en estos tres tipos de asignaturas se puede hacer de diversas maneras dependiendo de los objetivos de la titulación, la Universidad, el ámbito

social... El Comité que ha elaborado el documento asume seis estrategias, como las más aceptadas, para la definición de las asignaturas de iniciación.

Según estas recomendaciones, la Ingeniería del Software es un área para la que sólo se debería cubrir, en las asignaturas de iniciación, un subconjunto de sus materias troncales:

SE1. Diseño del Software. Materias para las asignaturas de iniciación:

- Principios y conceptos fundamentales del diseño.
- Análisis y diseño orientado a objetos.
- Diseña para reutilización.

SE2. Utilización de APIs. Materias para las asignaturas de iniciación:

- Programación con APIs.
- Inspectores y visualizadores de clases y herramientas relacionadas.
- Programación a través de ejemplo.
- Depuración en el entorno de API.

SE3. Herramientas y entornos de software. Materias para las asignaturas de iniciación:

- Entornos de programación.
- Herramientas para pruebas.

SE5. Requisitos y especificaciones del software. Materias para las asignaturas de iniciación:

- Importancia de la especificación en el proceso software.

SE6. Validación del software. Materias para las asignaturas de iniciación:

- Fundamentos de pruebas, incluyendo la creación de un plan de pruebas y la generación de casos de prueba.

El *Computing Curricula 2001* recomienda un esquema de tres cuatrimestres para cubrir las asignaturas de iniciación. Hay que destacar que, cuando se revisa la manera en que se construyen las asignaturas, se observa una discrepancia profunda con el sistema educativo español. Para cualquiera de las seis estrategias de construcción para las asignaturas de iniciación, las recomendaciones realizan cada asignatura con materias de diferentes áreas. Así, un esquema de *anchura primero* – que pretende dar una visión global de la Informática en las asignaturas de iniciación – estaría constituido por una secuencia de tres asignaturas para los tres primeros cuatrimestres de la titulación ($CS101_B$, $CS102_B$ y $CS103_B$), más otra asignatura de

introducción $CS100_B$. La troncalidad del área de Ingeniería del Software se cubriría en un 30% en los cuatrimestres segundo y tercero (asignaturas $CS102_B$ y $CS103_B$).

Además, e independientemente de la estrategia elegida, el informe propone cinco asignaturas de iniciación adicionales (CS105, CS106, CS115, CS120 y CS130) en las que se cubre un 6% adicional de la troncalidad de Ingeniería del Software. Estas nueve asignaturas, repartidas en los tres primeros cuatrimestres de la titulación dan como resultado tres asignaturas a la semana en cada cuatrimestre, en las que se cubre entre un 36% y un 64% de la troncalidad del área de Ingeniería del Software.

Para las asignaturas intermedias, también se proponen varias estrategias que, en este caso, son temáticas: por materias (8 asignaturas), intensivo (5 asignaturas), sistemas (9 asignaturas) y orientado a web (8 asignaturas). De las 19 asignaturas propuestas, la mayoría comunes para varias de las estrategias temáticas, sólo cuatro contienen materias troncales de Ingeniería del Software: $CS255_{\{S,W\}}$, $CS290_{\{T\}}$, $CS291_{\{S\}}$ y $CS292_{\{C,W\}}$. Sin embargo, la tasa de cobertura del área de Ingeniería del Software en estas asignaturas es muy alto. Por ejemplo, la asignatura $CS292_{\{C,W\}}$ contiene un 50% de la troncalidad de Ingeniería del Software.

En cuanto a las asignaturas avanzadas, el *Computing Curricula 2001* propone para el área de Ingeniería del Software las asignaturas recogidas en la Tabla 4.15.

Por último, el *Computing Curricula 2001* hace un especial hincapié en la necesidad de que todos los estudiantes realicen un proyecto, con trabajo en equipo, como parte indispensable de su Plan de Estudios.

4.4.1.4 Modelo de currículo para las artes liberales

Este modelo de currículo [Gibbs y Tucker, 1986] presenta una aproximación alternativa a la disciplina de la Informática, pone un gran énfasis en que la Ciencia de la Computación y tiene un cuerpo coherente de principios científicos. Define la Informática como un estudio sistemático de propiedades formales, implementación y aplicación de algoritmos y estructuras de datos.

La Ingeniería en esta propuesta curricular está ausente; en una posterior revisión [Walker y Schneider, 1996] se le asignan trece horas a la Ingeniería del Software (frente a las cuarenta y cuatro que recomendaba el *Computing Curricula 91*). Se presenta también un curso optativo que lleva por nombre *Ingeniería del Software*.

Existe alguna propuesta para la creación de un programa de estudios centrado en la Ingeniería del Software para su desarrollo en las Escuelas de Artes Liberales, como por ejemplo [Tymann et al., 1994].

4.4.2 Currículos centrados en los Sistemas de Información

Los Sistemas de Información son parte esencial de las organizaciones. Son sistemas complejos que requieren tanto experiencia técnica como de organización para el diseño, el desarrollo y la gestión.

Hay una estrecha relación entre los Sistemas de Información y la Ciencia de la Computación y la Ingeniería del Software. Sin embargo, hay grandes diferencias ya que los Sistemas de Información se concentran en la parte organizativa y en la aplicación de las tecnologías de la información para sus objetivos.

Ninguna de las propuestas curriculares conjuntas de ACM/IEEE-CS entra en los Sistemas de Información, ya que son ajenos a la aplicación de las tecnologías de la información a los Sistemas de Información en las empresas u organizaciones; es decir, estarían cercanos a lo que en España se denomina *Ingeniería Técnica en Informática de Gestión*.

Dentro de un Plan de Estudios centrado en los Sistemas de Información, la Ingeniería del Software es una disciplina instrumental y su importancia relativa dependerá del grado de orientación tecnológica que se le quiera dar a la titulación, así hay titulaciones basadas en Sistemas de Información más orientadas al mundo empresarial, mientras otras están más orientadas al desarrollo de aplicaciones; estas últimas son las que utilizan conocimientos de Ingeniería del Software.

El desarrollo de currículos para Sistemas de Información comienza a principios de la década de los setenta, con el Plan de Estudios definido por la ACM [Ashenhurst, 1972]. Las razones de esta incursión en terrenos tradicionalmente ajenos fueron [Camps, 1999]:

1. El sector de los Sistemas de Información era y es el sector de aplicación de la Informática que más técnicos solicita y emplea.
2. En EEUU algunos departamentos de Ciencia de la Computación impartían enseñanzas propias de los Sistemas de Información.

4.4.2.1 Antecedentes de los modelos de currículos para los Sistemas de Información

La cronología de los diferentes currículos definidos en el campo de los Sistemas de Información comienza a principios de la década de los setenta con una propuesta curricular liderada por ACM. Los hechos más relevantes se presentan a continuación:

- Mayo de 1972: *ACM Graduate Professional Programs in Information Systems* [Ashenhurst, 1972].
- Diciembre de 1973: *ACM Undergraduate Programs in Information Systems* [Couger, 1973].
- Marzo de 1981: *ACM Educational Programs and Information Systems* [ACM, 1981; Nunamaker, 1981].
- 1981 *DPMA Curriculum for Undergraduate Information Systems Education* [DPMA, 1981].
- En el año 1982, **ACM** elabora un nuevo informe para la enseñanza de los Sistemas de Información, el currículo **ACM-IS-82** [Nunamaker et al., 1982]. En él se hacen las siguientes distinciones entre el currículo en Ciencia de la Computación y el currículo para Sistemas de Información:
    - El currículo de Sistemas de Información enseña conceptos y procesos de Sistemas de Información, en dos contextos; conocimientos de organización y gestión, y conocimientos técnicos sobre Sistemas de Información. Por el contrario, la Ciencia de la Computación tiende a ser enseñada en un entorno de matemáticas, algoritmos y tecnología.
    - En cuanto a conocimientos técnicos, el currículo de Sistemas de Información pone substancial énfasis en la capacidad para desarrollar la estructura de un Sistema de Información para una organización (institución/empresa) y para diseñar e implementar aplicaciones. Al titulado en Ciencia de la Computación se le suele hablar menos de análisis de requisitos de información y de consideraciones organizativas, pero adquiere mayores conocimientos en desarrollo de algoritmos, programación, software de sistemas y hardware.
- Durante 1990 se desarrolla el **IS'90** propuesto por **DPMA** (*Data Processing Management Association*) [Longenecker y Feinstein, 1991].
- En 1994 aparece el **IS'94** de **DPMA** [Longenecker et al., 1994].
- Quince años después del currículo **ACM-IS-82**, en 1997 ACM vuelve a tratar los Planes de Estudio para los Sistemas de Información, cuando conjuntamente con la **AITP** (*Association of Information Technology Professionals*), antes **DPMA**, y la **AIS**

(*Association for Information Systems*) propusieron un nuevo currículo para los Sistemas de Información, el **IS'97** [Davis et al., 1997], de amplia repercusión.

- Actualmente se está trabajando una nueva actualización denominada **IS'2000**.
- Tanto el **IS'97** como el **IS'2000** son Planes de Estudios para estudiantes no graduados. Existe una propuesta de *master* especializado en Sistemas de Información, denominado **MSIS2000** [Gorgone et al., 1999].

4.4.2.2 Ingeniería del Software en los currículos para los Sistemas de Información

La presencia de la Ingeniería del Software en estos modelos de currículo empieza a ser significativa a partir del modelo IS'90 definido por la **DPMA** [Longenecker y Feinstein, 1991].

	ÁMBITO	TEMAS
SI'97.7-Análisis y diseño lógico	Ofrece una comprensión del proceso de desarrollo y modificación de los sistemas software. Permite a los alumnos evaluar y escoger una metodología. Enfatiza la comunicación entre las partes interesadas. ADOO. Modelado de datos. Ciclo de vida estándar.	Fases del ciclo de vida; técnicas de entrevista; JAD; DOO; prototipado; diseño de bases de datos; análisis de riesgos; gestión de proyectos; métricas de calidad del software; evaluación y adquisición de paquetes software; código de ética profesional.
SI'97.8-Diseño físico e implementación con SGBD	Diseño e implementación de Sistemas de Información con un SGBD.	Modelado de datos: herramientas y técnicas; paradigma estructurado y OO; modelos de bases de datos; CASE; repositorios; *datawarehouse*; IGU; cliente-servidor; conversiones; mantenimiento; formación de usuarios.
SI'97.9-Diseño físico e implementación con entornos de programación	Diseño físico, programación, prueba e implantación de un sistema. Implementación OO y cliente-servidor utilizando entornos de desarrollo.	Selección del entorno de programación cliente-servidor; construcción de software: estructurado, orientado a eventos y OO; pruebas; calidad del software; formación de usuarios; gestión de la configuración; mantenimiento; ingeniería inversa y reingeniería.
SI'97.10-Gestión de proyectos y práctica	Cubre los factores necesarios de la gestión del desarrollo de sistemas. Cubre tanto los aspectos técnicos como los de comportamiento. Se centra en la gestión del desarrollo de sistemas de nivel empresarial.	Gestión del ciclo de vida; integración de sistemas y bases de datos; gestión de redes y cliente-servidor; métricas para la gestión de proyectos y para la evaluación del rendimiento de sistemas; gestión de recursos humanos; análisis de coste-beneficio; técnicas de presentación; gestión de cambios.

Tabla 4.16. Descripción de los cursos más relacionados con la Ingeniería del Software del IS'97

En el último modelo de currículo **IS'97** definido por la **ACM**, la **AIS** y la **AITP** (formalmente **DPMA**) [Davis et al., 1997] presenta veinte subáreas significativas, de las que las áreas de **Análisis, diseño e implementación de Sistemas de Información** y de **Gestión de Proyectos** serían las más directamente relacionadas con la Ingeniería del Software, aunque muchas otras tendrían una relación más colateral. Por su parte, el curso que más directamente se relaciona con la Ingeniería del Software es el que lleva el epígrafe **SI'97.7-Análisis y diseño lógico**, estando también relacionados los cursos con epígrafes **SI'97.8, SI'97.9** y **SI'97.10**; la descripción de estos cursos se encuentra en la Tabla 4.16**.**

En el modelo curricular para graduados **MSIS2000**, definido por la ACM y la AIS [Gorgone et al., 1999] presenta una línea profesional directamente relacionada con la Ingeniería del Software, **Analista y diseñador de sistemas**, que trataría temas de: *metodologías de diseño avanzadas (ADOO, RAD, Prototipado), Gestión de proyectos avanzada, Integración de sistemas* y *Consultoría de Sistemas de Información.*

4.4.3 Currículos centrados en la Ingeniería del Software

En su mayor parte, los currículos centrados en la Ciencia de la Computación están más orientados a la formación de científicos en computación que ingenieros, mientras que la realidad empresarial e industrial demanda profesionales que sean capaces de afrontar con éxito los proyectos que paulatinamente quedan inconclusos o con carencias significativas, suponiéndoles graves pérdidas económicas.

Para poder formar profesionales que afronten con garantías los problemas reales de las empresas se necesitan programas que hagan hincapié en la Ingeniería del Software, dado que los programas existentes centrados en la Ciencia de la Computación prestan poca atención a la Ingeniería del Software [Jalics y Golden, 1995; Vaughn, 2000].

Existen diferentes propuestas curriculares que se centran en la Ingeniería del Software, ya sea en el contexto de los estudios de graduación o postgrado, cuya influencia está derivando en la definición de un cuerpo de conocimiento propio de la Ingeniería del Software [Abran et al., 2001c] que sienta la base para el establecimiento de una propuesta curricular de ámbito internacional que tenga a la Ingeniería del Software como el objetivo central de la misma.

Se presentan a continuación las propuestas curriculares elaboradas por el **Instituto de Ingeniería del Software** (SEI – *Software Engineering Institute*) en la **Universidad Carnegie-Mellon** (CMU) en USA y por el **WGSEET** (*Working Group on Software Engineering Education and Training*).

4.4.3.1 Las propuestas del SEI-CMU

En 1985 la Universidad Carnegie Mellon presenta un currículo general en Informática [Shaw, 1985] donde la *Ingeniería del Software* aparece como una asignatura en el nivel intermedio, denominada "*Organización de programas*" ("*programming in the small*", ampliación de TAD, POO, reusabilidad, especificaciones formales e informales...) y en los cursos avanzados en forma de dos asignaturas: "*Ingeniería del Software*" ("*programming in the large*", diseño avanzado y especificación, descomposición en módulos, CASE, prototipado, modelado...) y "*Laboratorio de Ingeniería del Software*" (desarrollo de proyectos en grupo, colaboración con la Industria). En este currículo aparece el concepto de "*nociones recurrentes*" que después aparecerían como novedad en el currículo ACM/IEEE-CS91.

Con posterioridad el Instituto de Ingeniería del Software, SEI a partir de ahora, también en la Carnegie Mellon, realiza diferentes currículos específicos en Ingeniería del Software para estudios de graduación y de postgrado.

Programas de postgrado

El SEI establece primeramente estudios de postgrado, *masters*, centrados en Ingeniería del Software, que se detallan en diferentes informes y artículos entre 1989 y 1991 [Gibbs, 1989; Ardis y Ford, 1989; Ford, 1991a], siendo este último el de mayor difusión.

En el informe de 1991 se realiza un desarrollo muy completo y detallado de las áreas temáticas de Ingeniería del Software. El citado informe incluye un modelo de currículo, numerosa bibliografía sobre Ingeniería del Software y descripciones de revistas de investigación de la disciplina. Incluye también descripciones detalladas de los contenidos de la serie de audiovisuales (cintas de vídeo) que constituyen a su vez un ejemplo de una implementación del modelo de currículo. Se describen, además, los programas para postgraduados en Ingeniería del Software de 23 universidades de todo el mundo.

El material se organiza en cursos universitarios, e incorpora el concepto de "*unidad de conocimiento*", al igual que el ACM/IEEE-CS91. También como esta última propuesta, el currículo se organiza en una serie de materias fundamentales optativas avanzadas o complementarias (20-30% de un currículo), y proyectos prácticos de desarrollo (al menos un 30% del trabajo total del estudiante).

La secuencia que sugiere (aunque en esto no es restrictivo) es la de una aproximación docente, comenzando con una visión de proceso para colocar cada actividad individual en su contexto, para seguir con aspectos de gestión del software y actividades de control, finalizando con las actividades concretas de desarrollo y la visión producto.

En el currículo se incluyen materias tanto de Ingeniería del Software, como, de una forma más amplia, de Ingeniería de Sistemas. Las materias fundamentales se distribuyen en 21 unidades, sin recomendación temporal, describiendo para cada unidad sus contenidos, aspectos de la actividad que son más importantes y objetivos educativos de la unidad. A continuación se ofrece una relación de dichas unidades.

- El proceso de Ingeniería del Software.
- Evolución del software.
- Generación de software.
- Mantenimiento de software.
- Comunicación técnica.
- Gestión de configuraciones software.
- Conceptos de calidad del software.
- Aseguramiento de la calidad del software.
- Organización y gestión de proyectos de software.
- Economía de proyectos software.
- Conceptos de operación del software.
- Análisis de requisitos.
- Especificación.
- Diseño de sistemas.
- Diseño de software.
- Implementación de software.
- Pruebas del software.
- Integración de sistemas.
- Sistemas de tiempo real “empotrados”.
- Interfaces hombre-máquina.
- Asuntos de la profesión.

El *master* en Ingeniería del Software que se imparte en la Universidad Carnegie Mellon consta de tres elementos básicos [Garlan et al., 1997]:

1. **Un núcleo curricular:** Formado por los cursos sobre los que recaen los fundamentos de Ingeniería del Software, haciendo un énfasis especial en el análisis, diseño y gestión de grandes sistemas software. Los cursos que conforman este currículo son:

a. Modelos de sistemas software.

b. Métodos de desarrollo de software.

c. Gestión del desarrollo del software.

d. Análisis de elementos software.

e. Arquitecturas de sistemas software.

2. **Desarrollo de un proyecto:** Durante la duración del *master*, los alumnos planifican e implementan un proyecto software de tamaño significativo para un cliente externo. El trabajo se hace en equipo supervisado por profesores.

3. **Cursos de especialización:** De carácter optativo, que permiten que los estudiantes desarrollen una experiencia más profunda en una de las siguientes especialidades, entre las que se incluyen *sistemas de tiempo real, interfaces hombre-máquina* y *mejora del proceso software*.

Programas de graduación

En el SEI se llega a la conclusión de que la incesante demanda de ingenieros del software no se puede cubrir sólo con los estudiantes de postgrado, lo que hace necesaria la creación de programas de graduación en Ingeniería del Software. Estos programas quedan documentados en [Ford, 1990; Ford, 1991b; Ford, 1994].

El esquema de este currículo es:

1. **Matemáticas y Ciencia.** Con los objetivos de preparar a los alumnos para participar en una sociedad cada día más tecnológica, así como de ofrecerles los fundamentos necesarios para afrontar el resto de las asignaturas de la titulación. Se recomiendan dos asignaturas de matemáticas discretas, una de estadística y probabilidad, dos de cálculo, una de métodos numéricos, una de física, una de química y una de biología.

2. **Ciencia de Ingeniería y Diseño de Ingeniería.** Con los siguientes cursos:

   a. Análisis. Ofrece al alumno el conocimiento para realizar modelos y razonar sobre el proceso software. Algunos de los temas a tratar son: *desarrollo formal de algoritmos y programas (incluyendo la verificación formal de los mismos); técnicas de abstracción y modelado; sistemas formales y su aplicación a la Ingeniería del Software; métricas, análisis de algoritmos; análisis de rendimiento...*

b. **Arquitecturas software.** Abordan la solución de problemas recurrentes a un alto nivel. Algunos temas pueden ser: *representación de datos, información y conocimiento; gestión de recursos; sistemas expertos; sistemas de tiempo real embebidos; sistemas concurrentes, paralelos o distribuidos...*

c. **Hardware.** En la Ingeniería del Software no todo es software, debiendo un ingeniero del software comprender el hardware que está presente en los sistemas informáticos. Se tratan temas relacionados con *lenguajes ensambladores; arquitectura de computadores; sistemas digitales; redes...*

d. **Proceso software.** Se encarga de transmitir el conjunto de herramientas, métodos y prácticas que se utilizan en la creación de software. Temas de este curso son: *análisis de requisitos; especificación y métodos formales; diseño; técnicas de implementación y lenguajes; verificación y validación; evolución del software; evaluación de productos y procesos software; gestión y organización de equipos de trabajo; temas éticos y profesionales.*

3. **Humanidades, Ciencias Sociales y Optativas.** Para completar la formación del alumno.

4.4.3.2 La propuesta del WGSEET

Cuando se presentaron los cuerpos de conocimiento sobre Ingeniería del Software, ya se comentó la iniciativa del WGSEET para la definición de un modelo de currículo para esta disciplina, que se recoge en [Bagert et al., 1999].

Esta propuesta curricular se centra en la generación de nuevos graduados que tengan en la Ingeniería del Software la base de conocimientos para afrontar una vida profesional condicionada por las necesidades de su entorno industrial y empresarial.

Arquitectura del currículo

En el diseño de este currículo intervienen una serie de elementos básicos, como se puede apreciar en la Figura 4.9.

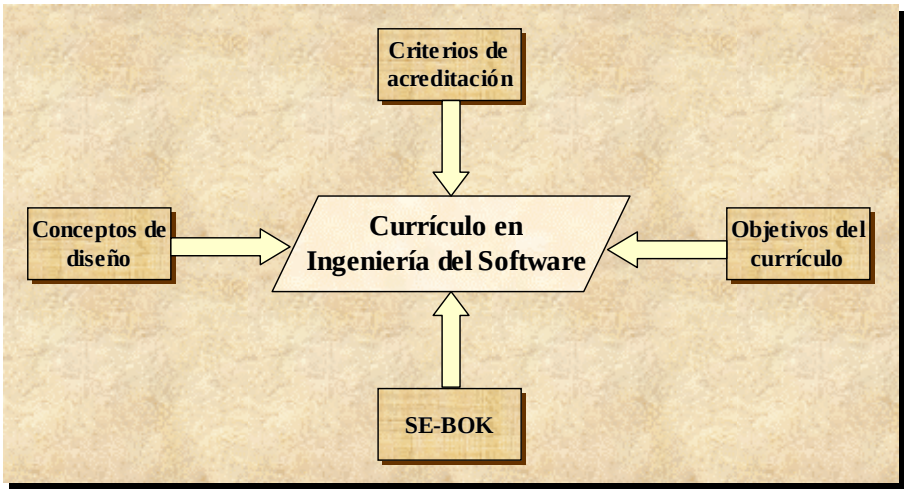


Figura 4.9. Arquitectura del currículo en Ingeniería del Software del WGSEET [Bagert et al., 1999]

La determinación de los objetivos del currículo se convierte en una actividad prioritaria en el diseño del mismo. Conlleva consideraciones sobre la misión de la institución que lo va a implantar y sobre la estrategia que se quiere seguir. En un nivel más fundamental, todos los afectados por la definición del currículo (profesores, alumnos, empresarios...) debieran estar representados en el momento de establecer estos objetivos.

El cuerpo de conocimientos ofrece la base de fundamentos para fijar el contenido del currículo, mientras que los criterios de acreditación introducen las bases para asegurar la calidad y la evaluación del mismo.

Por último, los conceptos de diseño construyen un marco filosófico para el desarrollo de currículos efectivos en Ingeniería del Software, así como para la definición del modelo de currículo del WGSEET.

Conceptos de diseño

La lista de conceptos que aparecen en esta sección constituye los principios básicos para el modelo de currículo en Ingeniería del Software propuesto por el WGSEET.

1. Soporta el desarrollo de varios programas de estudios, con la característica común de centrarse en la Ingeniería del Software (Ingeniería del Software, Ciencia de la Computación, Sistemas de Información...).
2. Introduce a los alumnos, desde el comienzo de sus estudios, en la naturaleza, ámbito e importancia de la Ingeniería del Software.
3. El modelo incorpora dos niveles de educación en Ingeniería del Software:

   a. La denominada Ingeniería del Software a pequeña escala (*Software Engineering in the Small*)
      i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados de forma individual o en pequeños grupos.
      ii. El desarrollo del software con estas características se encuentra concentrado en los primeros cursos.
   b. La denominada Ingeniería del Software a gran escala (*Software Engineering in the Large*)
      i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados en equipo.
      ii. El desarrollo del software con estas características se encuentra localizado en cursos avanzados o en pequeños proyectos realizados para empresas.

4. El modelo establece un balance entre *producto* y *proceso*.
   a. Las actividades relacionadas con el producto incluyen métodos, técnicas y propiedades usados en la construcción de los elementos software asociados en un producto software (planes de desarrollo, planes de aseguramientos de la calidad, especificación de requisitos, especificación de diseño, código, documentación...).
   b. Las actividades relacionadas con el proceso incluyen los estándares, procedimientos, guías, métricas y técnicas de análisis y soporte que ofrecen el marco adecuado para llevar a cabo las actividades de construcción del producto de forma efectiva y eficiente.
5. El modelo ofrece una guía para el desarrollo de programas de graduación que pueden ser acreditados por organizaciones externas.
6. Se incluyen asuntos éticos, sociales y profesionales que estén directamente relacionados con la práctica profesional de la Ingeniería del Software.
7. El modelo enfatiza el concepto de trabajo en grupo.
8. Ofrece especialización en dominios de aplicación concretos (sistemas empotrados, bases de datos y Sistemas de Información, sistemas inteligentes, redes...).

9. Asegura la práctica de la Ingeniería del Software:
    a. Prácticas en laboratorios planificadas.
    b. Proyectos de Ingeniería del Software a realizar en laboratorio.
    c. Educación cooperativa y prácticas en empresas.

Contenidos del currículo

Aunque la organización y puesta en marcha de varios currículos centrados en la Ingeniería del Software puede diferir de unos a otros, todos ellos deben ofrecer asignaturas que incluyan conocimientos sobre Matemáticas, Fundamentos de Ciencia de la Computación, Ingeniería del Software, Educación General, Ciencias Naturales y Conocimientos de Dominios Específicos.

En la siguiente lista se describe lo que, como mínimo, el WGSEET considera que se debe incluir en cada área:

1. **Fundamentos de Ciencia de la Computación**
    a. Programación, estructuras de datos y algoritmos.
    b. Conceptos de lenguajes de programación.
    c. Organización de computadores.
    d. Sistemas software (gestión de procesos y recursos, computación concurrente y distribuida, redes).
2. **Matemáticas**
    a. Matemática discreta.
    b. Estadística y probabilidad.
3. **Ciencias naturales**
4. **Educación general**
    a. Comunicación (oral y escrita).
    b. Ciencias sociales y humanidades.
5. **Ingeniería del Software**
    a. Ingeniería de requisitos.
    b. Diseño del software.
    c. Calidad del software.
    d. Arquitecturas software.
    e. Construcción del software.

f. Evolución del software.

g. Métodos formales.

h. Interfaces hombre-máquina.

i. Organización, planificación de proyectos.

j. Proceso software.

k. Ética y profesionalidad en la Ingeniería del Software.

Módulos propios de la Ingeniería del Software

Los módulos que se presentan en la Tabla 4.17, pueden hacerse corresponder cada uno de ellos con una sola asignatura, o combinarse varios de ellos en una asignatura.

Módulo	Título	Descripción
IS1	*Introducción a la Ciencia de la Informática para ingenieros del software 1*	Introducción a la programación, normalmente en el paradigma procedimental. Las características básicas de la Ingeniería del Software se integran en este curso.
IS2	*Introducción a la Ciencia de la Informática para ingenieros del software 2*	Introducción a las estructuras de datos y al paradigma objetual. Se continúa con la introducción de conceptos de Ingeniería del Software.
IS3	*Introducción a la Ingeniería del Software*	Descripción general de la Ingeniería del Software como disciplina; introduce los principios fundamentales y las metodologías.
IS4	*Ética y profesionalismo*	Cubre material sobre aspectos históricos, sociales y económicos de la Ingeniería del Software. Incluye el estudio de las responsabilidades y riesgos profesionales, así como sobre la propiedad intelectual.
IS5	*Requisitos del software*	Introduce los conceptos básicos y principios de la ingeniería de requisitos: sus herramientas, técnicas y métodos para el modelado del software.
IS6	*Diseño del software*	Métodos y técnicas utilizados en la fase de diseño. Énfasis en el diseño orientado a objetos.
IS7	*Calidad del software*	Aseguramiento de la calidad y gestión de la configuración.
IS8	*Construcción y evolución del software*	Examina problemas, métodos y técnicas asociadas con la construcción del software, dado un diseño de alto nivel y teniendo en cuenta el mantenimiento del mismo.
IS9	*Proyecto*	Permite a los estudiantes poner en práctica los conocimientos adquiridos en el resto de módulos, al entrar a formar parte de un equipo de desarrollo que se encarga de la realización de un proyecto.

Tabla 4.17. Módulos relacionados con la Ingeniería del Software en el currículo WGSEET

Para una descripción más detallada de cada uno de estos módulos, incluyendo un índice de primer nivel de los contenidos de cada uno de ellos, se recomienda la consulta de [Bagert et al., 1999].

Influencias de otras propuestas curriculares

La propuesta curricular del WGSEET recibe influencias de otras propuestas anteriores, que también coinciden en el objetivo de establecer un currículo independiente para la disciplina de Ingeniería del Software. De todas ellas se van presentar las dos más relevantes.

Thomas B. Hilburn propone un modelo conceptual de currículo en Ingeniería del Software [Hilburn, 1997] que se basa en tres pilares fundamentales:

- ***Las personas:*** Un currículo debe soportar las actividades que formen al alumno para trabajar y comunicarse con sus semejantes de una forma efectiva. Las capacidades a considerar son: *educación general, capacidad de comunicación, trabajo en grupo* así como *ética y profesionalismo.*
- ***Proceso:*** Los estudiantes deben darse cuenta de la necesidad de utilizar un proceso definido para desarrollar productos software, distinguiendo el proceso para la creación de software de carácter individual, el proceso para trabajar en equipo y el proceso que involucra a los estándares seguidos por una organización.
- ***Tecnología:*** Los alumnos deben adquirir los conocimientos tecnológicos y científicos necesarios para el desarrollo de software de calidad, incluyendo conocimientos de *Matemáticas y Ciencia*, *Ciencia de la Computación* y *Desarrollo de Software*.

A. J. Cowling propone un modelo de currículo en Ingeniería del Software multi-dimensional [Cowling, 1998], que establece las siguientes dimensiones:

- **Los diferentes niveles de abstracciones que definen los componentes hardware y software**. Establece que el ámbito de la Ingeniería del Software está en la construcción de sistemas en los que su parte principal está constituida por componentes software. Permite distinguir a los ingenieros del software de aquéllos que trabajan en la parte de arquitecturas de computadores.
- **El balance de los contenidos en Informática con respecto a otras ramas de la Ciencia y la Ingeniería**. Permite distinguir entre la Ingeniería del Software y otras disciplinas.

- **El balance entre la teoría, el modelado y su aplicación práctica**. La base de esta tercera dimensión es la observación de que la Ingeniería del Software no es la mera construcción de sistemas software, sino que la construcción de éstos se hace acorde a un método de Ingeniería.
- **El balance entre la parte técnica y la parte no técnica**. Las tres primeras dimensiones se centran en la parte técnica, pero una característica importante de la Ingeniería es que consiste en algo más que un conjunto de aspectos técnicos. Se necesitan contemplar aspectos económicos, sociales y sicológicos.

4.5 La Ingeniería del Software en otros Planes de Estudios españoles

En [Molina, 2000] se realiza un pequeño estudio sobre los currículos de Ingeniería del Software en 11 universidades españolas. Se advierte que, aunque los contenidos globales de las titulaciones aplican las propuestas de los currículos internacionales (y por supuesto las directrices de los Planes de Estudio oficiales) la organización en asignaturas concretas es muy variada, probablemente debido a razones históricas y a una introducción progresiva de la disciplina en las titulaciones. Los temas más frecuentemente abordados son los referidos a aspectos generales como *Análisis y diseño del Software*, *Ciclo de vida del Software*, *Planificación*, *Pruebas y Validación*... Por otro lado, los temas menos corrientes son los relacionados con *Aspectos legales* (sólo 10%), *Especificación formal*, *Reingeniería* o *Componentes* (todos ellos con un 20%).

Siguiendo la misma estela, en este apartado se presenta un análisis actualizado y no exhaustivo sobre la inclusión de materias relacionadas con Ingeniería del Software en Facultades y Escuelas de Informática de varias Universidades españolas. El estudio se ha dividido en dos partes, correspondientes a las asignaturas de las titulaciones técnicas y superior. Aunque todavía existen Universidades que mantienen diplomaturas o licenciaturas con Planes a extinguir, sólo se han considerado titulaciones vigentes y temarios reales de las asignaturas.

En el caso de las Ingenierías Técnicas se han incluido en el estudio las asignaturas directamente relacionadas con la Ingeniería del Software. En la especialidad de Gestión (I.T.I.G.), esta materia es troncal, al contrario de la especialidad de Sistemas (I.T.I.S.). En este segundo caso se han recogido las asignaturas obligatorias u optativas correspondientes cuando ha sido posible.

Respecto a la titulación en Ingeniería en Informática (I.I.), además de las Universidades con mayor tradición en Informática, se han analizado específicamente situaciones similares a la que se da en la Universidad de Salamanca (una titulación superior sólo de segundo ciclo). En cualquier caso, además de las asignaturas troncales de la materia se han incluido aquellas optativas con relación directa con la Ingeniería del Software.

Las universidades estudiadas han sido las siguientes: Burgos, Carlos III de Madrid, Complutense de Madrid, Málaga, Oviedo, Politécnica de Cataluña, Politécnica de Madrid, Politécnica de Valencia, Sevilla y Valladolid. Se han tenido en cuenta las distintas situaciones en especial en la titulación superior (estudios de primer y segundo ciclos frente a estudios de sólo segundo ciclo).

Por último, hay que tener en cuenta que en la mayoría de las Universidades españolas, los Planes de Estudios y el contenido concreto de cada materia de las titulaciones de Informática están en constante actualización, por lo que es posible que se hayan producido modificaciones desde el momento en que se recogió esta información (enero y febrero de 2002, vía web, en general). La información recogida se resume en los siguientes apartados.

4.5.1 Titulaciones de primer ciclo

Universidad de Burgos

http://www.ubu.es/.

Asignatura: Análisis e Ingeniería del Software.

Titulación: ITIG.

Créditos: 12.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software. Ciclo de vida del proyecto. Paradigma estructurado: análisis y diseño. Paradigma objetual: análisis y diseño, introducción a los patrones de diseño. Gestión de proyectos.

Universidad Carlos III de Madrid

http://www.uc3m.es/.

Asignatura: Ingeniería del Software 1.

Titulación: ITIG.

Créditos: 7.

Carácter: Troncal.

Contenidos: Introducción la ingeniería del Software. Técnicas de análisis de aplicaciones de gestión. Técnicas de especificación del funcionamiento de la aplicación. Técnicas de diseño de aplicaciones de gestión.

Asignatura: Ingeniería del Software 2.

Titulación: ITIG.

Créditos: 7.

Carácter: Troncal.

Contenidos: Introducción. Procesos de gestión de proyectos software. Gestión de la configuración. Estimación de proyectos software. Organización de proyectos software. Planificación de proyectos software. El Plan de desarrollo software. Seguimiento y control de proyectos software. Gestión y Control de calidad.

Universidad Complutense de Madrid

http://www.ucm.es/.

Asignatura: Ingeniería del Software de Gestión 1. Ingeniería del Software 1.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal en ITIG, optativa en ITIS.

Contenidos: Conceptos básicos. Gestión, estimación y planificación de proyectos. Gestión de configuraciones. Ingeniería del Software orientada a objetos.

Asignatura: Ingeniería del Software de Gestión 2. Ingeniería del Software 2.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal en ITIG, optativa en ITIS.

Contenidos: El modelo de objetos. Clases y objetos. El Lenguaje Unificado de Modelado. Notación. Modelo de Proceso. El proceso de desarrollo de software unificado. Patrones de diseño orientados a objetos. Pruebas y métricas orientadas a objetos.

Universidad de Málaga

http://www.uma.es/.

Asignatura: Ingeniería del Software de Gestión.

Titulación: ITIG.

Créditos: 12.

Carácter: Troncal.

Contenidos: Introducción y estrategias para el desarrollo de sistemas de información. Metodologías de desarrollo. Gestión de proyectos. Métrica. Planificación de sistemas. Análisis, diseño e implementación de sistemas de información.

Asignatura: Ingeniería del Software.

Titulación: ITIS.

Créditos: 6.

Carácter: Optativa.

Contenidos: Introducción al Proceso Unificado. Desarrollo del Software. Captura de Requisitos. Análisis, diseño e implementación. Prueba. Métrica 3.

Universidad Politécnica de Cataluña

http://www.upc.es/.

Asignatura: Ingeniería del Software I. Especificación.

Créditos: 6.

Carácter: Obligatoria en ITIG y Optativa en ITIS.

Contenidos: Paradigmas en Ingeniería del Software. Requisitos y especificaciones. Especificación orientada a objetos: la notación UML. Especificación de sistemas de tiempo real en UML. Pruebas. Otros métodos de especificación.

Asignatura: Ingeniería del Software. Diseño I.

Créditos: 6.

Carácter: Obligatoria en ITIG y Optativa en ITIS.

Contenidos: Introducción al diseño. Diseño orientado a objetos. Pruebas del software.

Universidad Politécnica de Madrid

http://www.upm.es/.

Asignatura: Ingeniería del Software.

Titulación: ITIG/ITIS.

Créditos: 9.

Carácter: Troncal en ITIG y Obligatoria en ITIS.

Contenidos: Proceso y gestión del software. Análisis de requisitos del sistema y del software. Diseño e implantación del software. Prueba, mantenimiento y documentación del software. Automatización del software.

Asignatura: Metodologías de Desarrollo.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción. Análisis y determinación de requisitos. Análisis estructurado. Diseño de sistemas. Pruebas de software. Implantación del sistema. Interfaces de métrica v.3.

Universidad Politécnica de Valencia

http://www.upv.es/.

Asignatura: Ingeniería del Software.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal ITIG, Optativa ITIS.

Contenidos: Introducción. Paradigmas de ciclo de vida. Planificación y gestión de proyectos. Desarrollo de software orientado a objetos. Técnicas de prueba de programas.

Asignatura: Laboratorio de Ingeniería del Software.

Titulación: ITIG.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Análisis de aplicaciones de gestión. Construcción de proyectos software. Uso de herramientas CASE.

Universidad de Sevilla

http://www.us.es/.

Asignatura: Ingeniería del Software de Gestión I.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería de Software. Análisis de requisitos. Metodología. Especificación estructurada de sistemas software. Especificación orientada a objetos de sistemas software.

Asignatura: Ingeniería del Software de Gestión II.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción. Diseño. Diseño estructurado. Diseño OO. Diseño de planes de pruebas. La visión del diseño en Métrica.

Asignatura: Ingeniería del Software de Gestión III.

Titulación: ITIG.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Fundamentos de programación, diseño de interfaz de usuario. Arquitecturas de sistemas. Acceso a datos. Programación OO. Mapeo de modelo de objetos a modelo de datos relacional. Bases de datos en Internet. Sistemas de soporte a la toma de decisiones.

Universidad de Valladolid

http://www.infor.uva.es/.

Asignatura: Ingeniería del Software I.

Titulación: ITIG.

Créditos: 7,5.

Carácter: Troncal.

Contenidos: Conceptos básicos. Desarrollo del software. Análisis de necesidades y estudio de viabilidad. Diseño. Diseño de la interfaz. Diseño del tratamiento. Las etapas del método.

Asignatura: Ingeniería del Software II.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Métodos y técnicas de análisis y diseño orientado a objetos.

Asignatura: Ingeniería del Software I.

Titulación: ITIS.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Conceptos básicos. Principios de la Ingeniería del Software. Modelado de análisis. Modelado de la implementación.

Asignatura: Ingeniería del Software II.

Titulación: ITIS.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Conceptos básicos de la orientación a objetos. Especificaciones orientadas a objetos. Diseño de sistemas interactivos.

4.5.2 Titulaciones de Segundo ciclo

Universidad de Burgos

http://www.ubu.es/.

Asignatura: Planificación y Gestión de Proyectos.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Análisis y definición de requisitos. Gestión de configuraciones. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones.

Asignatura: Diseño y Mantenimiento del Software I.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Diseño, propiedades y mantenimiento de software. Análisis de aplicaciones.

Asignatura: Diseño y Mantenimiento del Software II.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Diseño, propiedades y mantenimiento de software. Análisis de aplicaciones.

Universidad Carlos III de Madrid

http://www.uc3m.es/.

Asignatura: Procesos Software I.

Titulación: II (segundo ciclo).

Créditos: 7.

Carácter: Troncal.

Contenidos: El proceso de desarrollo de software. La orientación a objetos. UML: el lenguaje unificado de modelado. Modelado estructural. Modelado dinámico. Modelado arquitectónico.

Asignatura: Procesos Software II.

Titulación: II (segundo ciclo).

Créditos: 5.

Carácter: Troncal.

Contenidos: La Ingeniería del negocio: Perspectivas. Patrones de Diseño. Metodologías Específicas para la Reutilización. Arquitecturas basadas en Componentes. Diseño de componentes. Modelos de Madurez en Reutilización.

Universidad Complutense de Madrid

http://www.ucm.es/.

Asignatura: Ingeniería del Software.

Titulación: II.

Créditos: 18.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería de Software. El proceso de desarrollo de software. Planificación y gestión de Proyectos. Análisis y Especificación de Requisitos. Diseño de Software. Ingeniería del software en sistemas distribuidos. Análisis de programas. La medida del Software. Mantenimiento de aplicaciones. Reingeniería de Software.

Universidad de Málaga

http://www.uma.es/.

Asignatura: Ingeniería del Software-Análisis.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Sistemas software complejos. Tecnologías de Objetos. El Lenguaje de Modelado Unificado. Proceso Unificado. Especificación de Requisitos. Análisis.

Asignatura: Ingeniería del Software-Diseño.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: El modelo de objetos. Patrones de diseño. Diseño orientado a objetos con UML. Lenguajes y aplicaciones.

Asignatura: Ingeniería del Software-Proyectos.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Gestión de Proyectos. Herramientas. Proyecto.

Universidad de Oviedo

http://www.uniovi.es/.

Asignatura: Ingeniería del Software I.

Titulación: II (segundo ciclo).

Créditos: 18.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software. Metodología estándar en la Administración: Métrica-2. Análisis de Sistemas según Métrica-2. Métodos de diseño. Diseño estructurado. Técnicas y estrategias de prueba del software. Métodos orientados a objetos. Introducción a la Planificación y la estimación. Documentación, Estándares y Herramientas CASE.

Asignatura: Ingeniería del Software II.

Titulación: II (segundo ciclo).

Créditos: 18.

Carácter: Optativa.

Contenidos: Métodos Estructurados para desarrollo de sistemas en Tiempo Real. Aseguramiento de la Calidad del Software. Funciones SQA. El Plan General de Calidad. Auditoría informática. Planificación, estimación y métricas. La Ingeniería del Software desde el punto de vista de la empresa.

Universidad Politécnica de Cataluña

http://www.upc.es/.

En el Plan de II, también aparecen las asignaturas reflejadas en el primer ciclo. Además aparece una segunda asignatura dedicada al diseño y, como optativa, una asignatura enfocada específicamente a los métodos formales.

Asignatura: Ingeniería del Software. Diseño II.

Titulación: II.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Arquitectura del software. Plataformas de sistemas distribuidos. CORBA. Diseño orientado a objetos de sistemas de información.

Asignatura: Métodos Formales en Ingeniería del Software.

Créditos: 4,5.

Carácter: Optativa.

Contenidos: Métodos formales orientados a modelos (Z, VDM). Métodos formales axiomáticos (Larch, OBJ), demostradores automáticos: el demostrador de Larch.

Universidad Politécnica de Madrid

http://www.upm.es/.

Asignatura: Ingeniería del Software I.

Créditos: 9.

Carácter: Troncal.

Contenidos: Introducción al proceso de IS. Factores Humanos. Gestión de Configuración. Aseguramiento de la calidad. Gestión de proyectos. Evaluación de Procesos. Adquisición del Software.

Asignatura: Ingeniería del Software II.

Créditos: 12.

Carácter: Troncal.

Contenidos: Ingeniería de Requisitos. Aproximación de Desarrollo Estructurado. Aproximación al Diseño Orientado a Objetos.

Asignatura: Profundización en Ingeniería del Software.

Créditos: 6.

Carácter: Optativa.

Contenidos: Seminarios sobre diversos temas como evaluación del proceso, métricas, ingeniería de usabilidad.

Asignatura: Aspectos profesionales de la Ingeniería del Software.

Créditos: 4,5.

Carácter: Libre elección.

Universidad Politécnica de Valencia

http://www.upv.es/.

A pesar de que el nombre que recibe, las asignaturas de Ingeniería de la Programación e Ingeniería de Requerimientos encajan con el perfil de Ingeniería del Software. También se ofrece una asignatura optativa sobre métodos formales.

Asignatura: Ingeniería de la Programación.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Análisis y diseño orientados a objetos. Garantía de calidad del software (se trata de mostrar una metodología de ciclo de vida completo, como por ejemplo OMT; el tema de calidad se orientará hacia métricas).

Asignatura: Laboratorio de Ingeniería de la Programación.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Nuevas tecnologías para el desarrollo de software. Programación por componentes. Programación cliente-servidor, etc. Prácticas en Borland Delphi (o similar).

Asignatura: Ingeniería de Requerimientos.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Estabilización de Requisitos. Validación de Requisitos. Modelos de tiempo real. Dinámica. Modelos del tiempo. Lógicas modales dinámicas. Herramientas y metodologías asociadas.

Asignatura: Métodos Formales de la Ingeniería del Software.

Titulación: II.

Créditos: 6.

Carácter: Optativa.

Contenidos: Necesidad del formalismo en Ingeniería del Software. Prototipado automático. Especificación lógica del Software. Especificación algebraica del software. Especificación orientada a objetos. Técnicas mixtas. Demostración formal de propiedades.

Universidad de Sevilla

http://www.us.es/.

Asignatura: Ingeniería del Software I.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software y a la orientación a objetos. Elicitación de requisitos de sistemas software. Modelos orientados a objetos de un sistema software. Especificación orientada a objetos de sistemas software. Metodología.

Asignatura: Ingeniería del Software II.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Conceptos básicos y fundamentales. Separación de conceptos, diseño modular, abstracción y diseño genérico. Diseño Arquitectónico basado en Patrones. Tecnologías de Diseño.

Asignatura: Ingeniería del Software III.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la gestión y dirección de Proyectos. Métricas del software. Métodos tradicionales de planificación y estimación. Gestión de la calidad del software. Gestión y control del riesgo. Herramientas de ayuda a la gestión.

Asignatura: Métodos Formales en la Ingeniería del Software.

Titulación: II.

Créditos: 6.

Carácter: Optativa.

Contenidos: Lenguajes de especificación. Especificación basada en UML y OCL. Una metodología de especificación basada en métodos formales. Especificación de tipos abstractos de datos, componentes, sistemas. Técnicas de diseño basadas en métodos formales, patrones de diseño. Diseño de los aspectos de concurrencia. Diseño de los aspectos de persistencia. Diseño de los aspectos específicos de las aplicaciones distribuidas.

Universidad de Valladolid

http://www.infor.uva.es/.

Asignatura: Ingeniería del Software I.

Titulación: II (segundo ciclo).

Créditos: 9.

Carácter: Troncal.

Contenidos: Modelos de ciclo de vida. Ingeniería de requisitos. UML. Métodos formales.

Asignatura: Ingeniería del Software II.

Titulación: II (segundo ciclo).

Créditos: 9.

Carácter: Troncal.

Contenidos: Diseño de software. Arquitectura de software. Métricas. Gestión de proyectos.

4.6 Resultados del análisis. Conclusiones

Desde el punto de vista de las recomendaciones internacionales, realizadas en países con diferentes estructuras universitarias y profesionales, la aplicación directa de las mismas no es realizable sin tener en cuenta la realidad universitaria y el entorno social y profesional de nuestro país. No obstante, el reconocimiento internacional de las mismas así como la autoridad científica y profesional de sus autores las convierten en valiosas guías a considerar a la hora de elaborar una propuesta docente específica.

Todas ellas coinciden en la importancia de perseguir tanto la obtención de conocimientos como de capacidades y habilidades por parte del alumno.

El *Computing Curricula 2001* [ACM/IEEE-CS, 2001], aunque no se especializa en Ingeniería del Software, aumenta considerablemente los contenidos sobre la materia respecto a la propuesta de 1991. Sin entrar en demasiadas profundidades, proporciona una guía sobre los contenidos básicos de la disciplina en el contexto de las titulaciones generalistas de Informática.

Por otro lado, las propuestas del SEI (e incluso los currículos propuestos para titulaciones de Sistemas de Información, en el caso de ITIG) aportan matices que pueden ayudar a definir una visión global de la Ingeniería del Software. De estas propuestas, el informe del SEI [Ford, 1991a], aunque un poco antiguo y dirigido básicamente a una titulación de postgrado, es bastante completo y detallado, incluyendo numerosas referencias bibliográficas e incluso material audiovisual de soporte.

Por último, la publicación del SWEBOK [Abran et al., 2001c] representa poner a nuestra disposición un cuerpo organizado de conocimientos sobre la Ingeniería del Software que puede jugar un importante papel a la hora de localizar referencias sobre temas concretos.

Analizando los contenidos de las distintas propuestas, se pueden agrupar, a grandes rasgos, en dos grandes bloques que se corresponden con la doble visión producto/proceso de la Ingeniería del Software. El primer bloque se puede organizar siguiendo las fases del ciclo de

vida del software. Siguiendo el esquema del SWEBOK y del *Computing Curricula 2001* (entre paréntesis, la clave correspondiente a las unidades del *Computing Curricula*):

- Requisitos del software (SE5).
- Diseño del software (SE1).
- Construcción del software (SE2, además de PF y PL).
- Prueba del software (SE6).
- Mantenimiento y evolución del software (parcialmente SE7).

El segundo bloque está constituido por las materias relacionadas con el Proceso Software y su gestión, en general:

- Gestión de la configuración del software (SE8).
- Gestión de la Ingeniería del software (SE8).
- Proceso de Ingeniería del software (SE4).
- Calidad del software (SE8).

Se puede dejar fuera, puesto que es transversal, la unidad de conocimiento Herramientas y métodos de la Ingeniería del Software (equivalente aproximadamente a la unidad SE3 del *Computing Curricula 2001*).

En cuanto a la importancia relativa de cada bloque, en la asignación de horas del *Computing Curricula 2001* al primer bloque se le dedican 23 de las 31 horas propuestas como fundamentales (*core*), o lo que es lo mismo, el 74%, en tanto que para el Proceso y las herramientas se deja el 26% restante. Evidentemente se trata de un enfoque dirigido a alumnos de cursos de graduación y la mayoría del tiempo se dedica al estudio de conceptos y técnicas fundamentales. En los programas de postgrado, del tipo del SEI, los porcentajes varían considerablemente. En otro orden de cosas, en el *Computing Curricula* la presencia de métodos formales se limita a aspectos básicos en la unidad SE5 (Requisitos software y especificaciones), pero la unidad opcional SE10 (Métodos formales) está dedicada expresamente a ellos.

Por último, no hay que olvidar un componente importantísimo de los sistemas modernos, la interfaz persona-ordenador. Tanto el SWEBOK como el *Computing Curricula* la consideran un área independiente de la Ingeniería del Software. Sin embargo, al no estar incluida en las directrices de los Planes de Estudio españoles, sus contenidos básicos se deben incluir en las asignaturas de Ingeniería del Software, con independencia de la necesidad de asignaturas optativas sobre el tema.

Sobre la presencia de la Ingeniería del Software en las Universidades españolas, se observa que en muchas de ellas se aumenta la troncalidad recomendada y se añaden asignaturas obligatorias u optativas sobre la disciplina, apareciendo también en la Ingeniería Técnica de Informática de Sistemas, donde no es troncal. En relación con los contenidos propuestos se observa cómo se intenta conjugar la adaptación de los currículos académicos a la realidad social, con el interés en mantener el rigor y el carácter de "persistencia" de los conceptos relacionados con esta disciplina. Éste es un objetivo compatible con la adquisición de conocimientos y destrezas concretas mediante la realización de prácticas y desarrollo de proyectos, contribuyendo así a alcanzar una formación integral del estudiante.

En el primer ciclo, se aprecia una fuerte presencia de lo que se puede llamar Ingeniería del Software convencional, básicamente, métodos estructurados y orientados a objetos, con una explosión reciente de temas dedicados al Lenguaje Unificado de Modelado (UML). La gestión de los proyectos software ocupa el segundo lugar entre los temas abordados con más frecuencia. A la hora de repartir la disciplina en dos o tres asignaturas cuatrimestrales, las diferencias entre los temarios estudiados se limitan, en lo fundamental, a pocas variantes: división basada en el ciclo de vida, división entre métodos estructurados y orientados a objeto, división entre temas de gestión y metodológicos.

En el segundo ciclo la situación es mucho más compleja. El punto de partida común es la presencia de al menos 18 créditos troncales, que en algunos casos se aumenta y en otros se complementa con asignaturas optativas. Se pueden detectar dos situaciones diferentes, según el tipo de organización de los estudios. La mayoría de las titulaciones superiores son "completas", de modo que el alumno llega al segundo ciclo sin haber estudiado previamente la disciplina. El segundo caso corresponde a algunas titulaciones de sólo segundo ciclo, enfocadas a alumnos con titulación previa en alguna de las Ingenierías Técnicas y por tanto con conocimientos previos sobre Ingeniería del Software. Sin embargo, a pesar de la diferencia, en el segundo ciclo predomina la formación desde cero en Ingeniería del Software, aunque con mayor profundidad. Sin embargo, y como se expone en [Canos et al., 1998], en el diseño de los currículos sería conveniente evitar repeticiones en el contenido en los dos ciclos así como distinguir entre la orientación del futuro ingeniero en Informática (más cercano a tareas de Planificación, Gestión de proyectos, Análisis y Especificación de Requisitos) y los ingenieros técnicos (más orientados al trabajo en las fases clásicas del desarrollo propiamente dicho).

En general se detecta mayor presencia de temas avanzados: Patrones de diseño, Desarrollo basado en componentes, Reutilización, Reingeniería de Software, Sistemas distribuidos, Aseguramiento de la calidad... Otros temas, como Prototipado, Especificaciones formales,

Demostración de propiedades, aparecen en la mayoría de los casos en asignaturas optativas o no aparecen en absoluto.

Llama la atención la coincidencia de casi todas las propuestas internacionales en cuanto a la consideración de los aspectos éticos, sociales y profesionales, y la poca importancia que a estos temas se le da en las Universidades españolas; esta situación está cambiando tímidamente, mediante la introducción de asignaturas optativas sobre dichos aspectos.

A modo de resumen, se puede concluir que un programa completo de formación en Ingeniería del Software ha de tener en cuenta los siguientes puntos:

- La formación específica en Ingeniería del Software ha de estar relacionada con la formación en algoritmos, programación, arquitectura de ordenadores, bases de datos, sistemas operativos o redes de ordenadores dentro de un marco general de formación en Informática.
- El programa ha de contemplar todos los aspectos del desarrollo, cubriendo todos los aspectos relacionados con el ciclo de vida del software, incluyendo el funcionamiento en un entorno real. Por lo dicho en el punto anterior, algunas fases de desarrollo (construcción y prueba del software o diseño de bases de datos) se tratan en otras disciplinas y hay que tener ese hecho en cuenta a la hora de diseñar y coordinar los temarios.
- En el programa de Ingeniería del Software han de incluirse además las áreas de gestión de procesos software, selección y utilización de componentes y herramientas software, garantía de calidad, interacción persona-ordenador, documentación y estándares. Sería deseable incluir aspectos complementarios como la formación en los aspectos sociales, éticos, legales, económicos, trabajo en grupo y técnicas de comunicación oral y escrita.
- El desarrollo de este programa necesita la existencia de recursos de laboratorio y herramientas que permitan la realización de prácticas y proyectos. Es esencial el acceso por parte de los alumnos y los profesores a herramientas que soporten el ciclo de vida del software.
- Es necesaria la existencia de material bibliográfico sobre todo en lo que se refiere a fuentes primarias (revistas) dada la continua y rápida evolución de los principios, métodos y técnicas de la Ingeniería del Software, con el fin de mantener una actualización permanente.

- Por último, hay que tener en cuenta que la formación en Ingeniería del Software ha de integrar requisitos técnicos con requisitos generales de educación para preparar a los estudiantes para una carrera profesional en este campo. Esta formación ha de permitirles adaptarse a los desarrollos tecnológicos de forma efectiva, rápida y con el menor coste posible a través de seminarios de adaptación impartidos desde el entorno académico (cursos de postgraduado) o desde el entorno industrial (seminarios de formación continuada).

El próximo capítulo se dedica a presentar la propuesta concreta que, siguiendo estos objetivos, el candidato se propone llevar adelante en el marco de las titulaciones en Informática de la Universidad de Salamanca.

4.7 Perfil profesional del ingeniero en informática

El contexto profesional en el que se van a mover los futuros titulados en Informática es un aspecto a tener en cuenta cuando se elabora un Proyecto Docente, ya que el objetivo de la educación universitaria es formar profesionales capaces de hacer frente a las demandas de la sociedad. Es esencial, por lo tanto, orientar los contenidos de las asignaturas para dar respuesta a las exigencias que la sociedad tiene en cada momento.

Peter J. Denning y R. Dunham [Denning y Dunham, 2001] defienden que los profesionales de las tecnologías de la información son los que ocupan en primer lugar en la tercera oleada profesional. Alvin Toffler [Toffler y Toffler, 1990] divide la historia de la civilización en tres oleadas: la era agrícola, la era industrial y la era de la información. En esta era de la información van a tomar una importancia capital habilidades como coordinación, relación con los clientes, gestión de compromisos, trabajo en equipo, aprendizaje continuo o capacidad de empresarial [Denning y Dunham, 2001].

Actualmente, es difícil precisar qué se entiende por un ingeniero en Informática, incluso muchos profesionales de la informática todavía no tienen muy claro en que consiste su profesión [Earles, 2000; Denning, 2001]. Esto puede ser debido a la falta de madurez de la misma unido al hecho de que esta disciplina ha ido acumulando a lo largo de los años tanto conocimiento científico que la obliga a cambiar constantemente. Existen períodos de tiempo en los que el cambio social y tecnológico es particularmente rápido, por lo que es más necesario entonces analizar y discutir la naturaleza y la evolución de la profesión. El momento actual puede ser uno de esos períodos.

Algunos pioneros en la Informática y organizaciones profesionales han intentado definir esta disciplina [Abrahams, 1987; Hartmanis, 1994; Denning, 1998; Bagert, 1999; Shaw, 2000]. La mayoría de esos intentos concluyen que la Ciencia de la Computación no es ni una Ciencia ni una Ingeniería en el sentido tradicional. Esta Ciencia define una relación nueva entre la teoría y la práctica. En Informática, a diferencia de otras disciplinas, no se puede separar la teoría y la Ingeniería porque sería desastroso [Denning, 1998]. Los recientes avances en algoritmos genéticos y redes neuronales son sólo unos pocos ejemplos de lo que podría producirse cuando se acopla la teoría con la práctica. Esto pone de manifiesto que los aspectos de Ciencia y de Ingeniería, en computación, están mucho más cercanos que en otras disciplinas.

Actualmente, la confusión sobre esta Ciencia, deriva en gran medida del nombre de la disciplina. La palabra Ciencia se refiere a conjunto de conocimientos. Puede usarse colectivamente para hacer referencia al campo de la Ingeniería como Ciencias de Ingeniería, pero el título de ingeniero en Informática no se debe asociar automáticamente con la construcción de software. Los problemas en computación demandan la invención de nuevas metodologías para la exploración de procesos intelectuales y de información. Algunas de esas metodologías se usan para desarrollar hardware y software. A los profesionales de la Informática les concierne el análisis y diseño de hardware y software para realizar nuevas funciones, o para realizar funciones antiguas de una nueva forma [Simons et al., 1991].

Hay una fuerte demanda en la sociedad de profesionales de la Informática que puedan desarrollar sistemas basados en metodologías con una base teórica sólida y probadas científicamente [West, 1997]. Esta necesidad es más evidente en el campo de la Ingeniería del Software debido a que está menos madura que la Ingeniería del Hardware, es más compleja, y produce productos estratégicos.

En la producción de sistemas software son importantes los aspectos técnicos, pero también lo son la disciplina, la capacidad de trabajo en grupo y el conocimiento del dominio de aplicación en el que se va a trabajar. Sin embargo, aún contando con una formación correcta y continuada, no es fácil hacer las cosas correctas porque suele faltar un soporte de la parte de gestión. A diferencia de otros campos profesionales, en Informática no se puede esperar que los gestores y los clientes tengan conocimientos de los métodos de desarrollo de software. Se necesitan profesionales que conozcan sus capacidades y puedan demandar los recursos y el soporte que necesitan para llevar a cabo un trabajo de calidad [Humphrey, 2000].

Los programas en Informática se pueden ver desde una nueva perspectiva, a través de la que se reconoce la importancia del enfoque de sistemas como sistemas de hardware y software. Se puede, entonces, apreciar tres procesos: teoría, abstracción y diseño [Tucker et al., 1991a].

En los últimos años, los mejores centros educativos ofrecen programas en Informática fuertemente acoplados con la Ingeniería. El problema es que esos programas no tienen el mismo nombre. En EEUU se conocen como "*Computer Science*", "*Computer Engineering*" y "*Computer Science and Engineering*", mientras que en Europa se denominan "*Informatics*". La Ingeniería del Software ha evolucionado pasando de ser una actividad de la "Ingeniería de Computadores" a ser una disciplina propia.

Actualmente existen dos tendencias relativas a la profesión informática, una de ellas defiende una única profesión en la que coexistan varias especialidades, mientras que otra tendencia propone la separación en dos profesiones Ingeniería de Computadores e Ingeniería del Software.

Los defensores de la primera corriente sugieren que el establecimiento de la profesión requiere la coordinación de esfuerzos para abordar los siguientes aspectos (en el marco educativo y laboral de EEUU) [El-Kadi, 1999]:

- *Redefinición*. La Informática es una nueva disciplina de Ingeniería. No se debe perder más tiempo intentando conocer su naturaleza, sino saber qué es y qué papel juega en la sociedad.
- *Educación*. Desde esta nueva perspectiva, IEEE-CS y ACM deben diseñar un currículo para preparar a los ingenieros en informática. El *Computing Curricula 91* [Tucker et al., 1991a] es un excelente punto de partida. Puede actualizarse con algunas reformas para centrar los objetivos educativos en una disciplina unificada con algunas subespecialidades. En el diseño de los nuevos currículos se debe prestar especial atención en introducción de nuevas áreas para incluir nuevos campos [Lewis, 1994; Simons et al., 1991]: diseño VSLI, procesamiento paralelo, diseño de redes de ordenadores, sistemas distribuidos, diseño de algoritmos, tecnología de orientación a objetos, inteligencia artificial, sistemas expertos y robótica. La cobertura con detalle de todos esos campos permitirá realizar especializaciones. (Este punto se ha hecho realidad con el *Computing Curricula 2001* [ACM/IEEE-CS, 2001]).
- *Titulación*. Es necesaria una cooperación entre IEEE-CS, ACM, CSAB y ABET en el diseño de exámenes para la obtención de la titulación [ACM/IEEE-CS, 1998; Mead y Turner, 1998; CSAB, 2000; ABET, 2000].

Las actividades que puede desarrollar un Ingeniero en Informática se pueden encuadrar en alguna de las áreas siguientes, que no son mutuamente excluyentes [Ng, 1999]:

- Desarrollo de software.
- Desarrollo de hardware.
- Integración de sistemas.
- Infraestructura de redes.
- Sistemas de control.
- Data warehousing.
- ERP.
- Negocio electrónico y computación Internet/Intranet.

Los defensores de la profesión única argumentan que la Ingeniería del Software no ha madurado lo suficiente para establecerse como una profesión separada, sin embargo sociedades tan prestigiosas como IEEE-CS y ACM promueven la profesión de Ingeniería del Software.

El SEI (*Software Engineering Institute*) ha desarrollado un modelo para caracterizar la madurez de una profesión en función de ocho componentes [Ford y Gibbs, 1996]:

- Educación profesional inicial.
- Acreditación.
- Desarrollo de habilidades.
- Certificación.
- Titulación.
- Desarrollo profesional.
- Código de ética.
- Sociedad profesional.

G. Ford y N. Gibbs estudiaron varias profesiones que pueden considerarse maduras como Medicina o Derecho y comprobaron que el camino de desarrollo profesional en esos campos es muy similar. La Figura 4.10 muestra esas características.

Los aspirantes a profesionales primeramente reciben una **educación**, generalmente universitaria, que precede al trabajo profesional. La calidad de un programa de una titulación universitaria se asegura por la acreditación. Esta acreditación permite a los profesionales graduados en esos programas iniciar su vida profesional con el conocimiento que necesitan para desarrollarla de una manera efectiva.

Para llegar a ser un profesional se deben adquirir habilidades en la aplicación de los conocimientos adquiridos. La **certificación y/o titulación** asegura la capacidad individual para entrar en la práctica profesional. La certificación es un proceso voluntario que ayuda a determinar quien está cualificado para participar en una profesión. La titulación es similar salvo en que es obligatoria y la administra una autoridad gubernamental.

El **desarrollo profesional** se refiere al mantenimiento y mejora de los conocimientos y habilidades adquiridos durante el período educativo después de iniciada la práctica profesional. Estos requisitos de desarrollo son especialmente fuertes en profesiones cuyo cuerpo de conocimiento técnico cambia rápidamente. Un aspecto del desarrollo profesional es el aprendizaje de estándares de práctica apropiados.

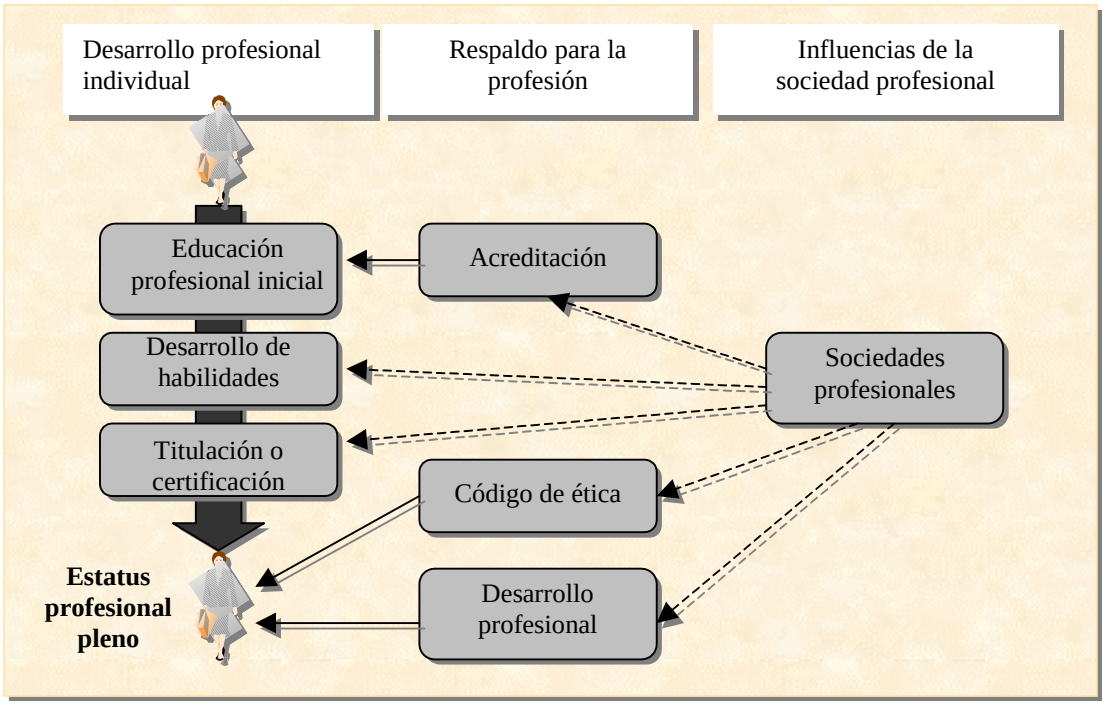


Figura 4.10. Etapas del desarrollo profesional en profesiones bien establecidas

En su comienzo las **sociedades profesionales** tenían como único objetivo promover el intercambio de conocimiento. Las funciones actuales incluyen desarrollo de estándares, definición de criterios de certificación, definición de códigos de ética y acciones disciplinarias para la violación de dichos códigos. Las sociedades IEEE-CS y ACM han trabajado activamente en la definición de la profesión de Ingeniería del Software. Dicho trabajo ha sido instrumentado, bajo su amparo, por el SWECC (*Software Engineering Coordinating Committe*).

La definición de un **código ético** se realiza para dirigir el comportamiento responsable de los profesionales. Una sociedad profesional ayuda a asegurar que el resto de los componentes interactúan de forma apropiada.

Componente	*Ad hoc*	Específico	Maduro
Educación profesional inicial	Preparación común de los titulados para entrar en la profesión.	Existe una formación reconocida de la educación inicial en Ingeniería del Software pero no existe un currículo estándar.	Existe un modelo de currículo nacionalmente aceptado; el modelo se examina y se revisa regularmente.
Acreditación de la Educación	Acreditación basada en criterios de ingeniería o de ciencia de la computación.	Acreditación basada en criterios de Ingeniería del Software. Fusión de ABET y CSAB.	Las directrices se examinan y se revisan regularmente.
Desarrollo de habilidades	Trabajo en proyectos en los centros educativos, programas de cooperación, programas de entrenamiento en compañías.	Surgen las directrices sobre la práctica necesaria para la profesión de ingeniero de software.	Los mecanismos de adquisición de habilidades están situados y su uso está generalizado.
Certificación	Certificación ICCP, ASQC; certificación comercial relacionada con paquetes de software y tecnologías.	Certificación como ingeniero de software. Estándares reconocidos nacionalmente.	Certificación en áreas especializadas de Ingeniería del Software. Estándares de certificación especializada reconocidos nacionalmente.
Titulación	Titulación oficial como ingeniero profesional.	Exámenes dirigidos específicamente a habilidades en Ingeniería del Software.	Titulación basada en exámenes apropiados; colaboración NSPE y NCEE.
Desarrollo profesional	Desarrollo seguido de forma individual según sus necesidades.	Directrices para el desarrollo profesional.	Educación reconocida nacionalmente y directrices de entrenamiento y currículo.
Código ético	Códigos de ACM, IEEE, ASQC, ICCP.	Código ético específico par Ingenieros de software.	Código ampliamente respetado y adoptado; mecanismos disciplinarios de incumplimiento.
Sociedad profesional	ACM, IEEE-CS, otras.	Sociedad que representa explícitamente la Ingeniería del Software.	Sociedad con un nivel apropiado de productos y servicios para los ingenieros de Software.

Tabla 4.18. Evolución de los componentes de la profesión de Ingeniería del Software

La caracterización de la madurez en el ámbito de componentes de una profesión se realiza en función de cuatro etapas evolutivas:

- *No existencia*. El componente no existe de ninguna forma.

- *Ad Hoc*. Existe algo relacionado con el componente, pero no se identifica con la profesión.
- *Específico*. El componente existe y se identifica claramente con la profesión.
- *Maduro*. El componente ha existido durante muchos años, se ha llevado a cabo de manera apropiada dentro de la profesión y está sujeto a mejoras continuas.

Se considera que una profesión está madura cuando sus componentes han alcanzado el nivel de madurez.

Los autores del documento que recoge este modelo de madurez, reconocen que la Ingeniería del Software es una profesión relativamente inmadura, sin embargo sugieren que el modelo propuesto puede ayudar a la maduración de la profesión. La Tabla 4.18 refleja la visión de dichos autores sobre la evolución de la profesión de Ingeniería del Software.

En el *Computing Curricula 2001* [ACM/IEEE-CS, 2001] se indican las características, capacidades y habilidades que se les suponen a los titulados en Informática. Brevemente, se enumeran a continuación:

- **Características generales:**
  - Perspectiva en el ámbito del sistema.
  - Apreciación de la interrelación entre teoría y práctica.
  - Familiaridad con los temas comunes (tales como abstracción, complejidad y evolución).
  - Experiencia significativa en proyectos.
  - Adaptabilidad.
- **Capacidades y habilidades:**
  - *Cognitivas:*
    - Conocimiento y comprensión.
    - Modelado.
    - Obtención y especificación de requisitos.
    - Métodos y herramientas.
    - Responsabilidad profesional.
  - *Prácticas:*
    - Diseño e implementación.
    - Evaluación.

- Gestión de la información.
- Interacción Persona-Ordenador.
- Manejo de los riesgos.
- Herramientas.
- Operación con hardware y software.

o *Adicionales:*

- Comunicación.
- Trabajo en equipo.
- Capacidad de comprender y explicar cuantitativamente la dimensión de un problema.
- Autogestión.
- Desarrollo profesional.

Como se ha comentado anteriormente, sociedades como IEEE-CS y ACM han desarrollado actividades encaminadas a hacer evolucionar la profesión de Ingeniería del Software, tales como el establecimiento de requisitos de certificación para los desarrolladores de software y acreditación para programas universitarios [ACM/IEEE-CS, 1998]. La acreditación es un tema espinoso, con opiniones encontradas. Así, como argumenta Parnas, las certificaciones dan confianza, pero no garantizan resultados seguros [Parnas, 2001].

Para completar este trabajo *IEEE-CS/ACM Join Task Force* han establecido recientemente otra clave de éxito para esta profesión: el código de ética de la Ingeniería del Software (*Software Engineering Code of Ethics and Professional Practice*) [ACM/IEEE-CS, 1999c]. La versión 5.2 ha sido adoptada por ambas sociedades después de un intenso proceso de revisión [Gotterbarn et al., 1999a]. En el Cuadro 4.1 se ofrece un breve resumen del mismo.

En España existe una Proposición de Ley para la creación de los Colegios Oficiales de Doctores e Ingenieros en Informática. Tales Colegios Profesionales deberán contemplar códigos de conducta. Gran parte de las tareas de los ingenieros en informática están relacionadas con el software, por lo que el código de la ACM/IEEE-CS que puede ser de gran utilidad para orientar la profesión en nuestro país [Dolado, 1999].

Como se ha comentado en apartados anteriores de este capítulo, las sociedades ACM e IEEE-CS están trabajando también de manera conjunta en un proyecto para desarrollar la guía SWEBOK (*Software Engineering Body of Knowledge*) [Abran et al., 2001c]. La articulación de

dicho cuerpo de conocimiento es un paso esencial par el desarrollo de una profesión debido a que representa un amplio consenso en los contenidos de la disciplina.

4.8 Referencias

[ABET, 2000] Accreditation Board for Engineering and Technology. "*Accreditation Policy and Procedure Manual*". Baltimore, MD: ABET, Inc., http://www.abet.org/images/policies.pdf. November 2000.

[Abi-Raad, 2000] Abi-Raad, M. "*Systems analysis with attitude!*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITICSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 57-60. ACM Press, 2000.

[Abrahams, 1987] Abrahams, P. "*President's letter*". Communications of the ACM, 30(6):472-47, June 1987.

[Abran et al., 1999] Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.5. ACM/IEEE-CS, October 1999.

[Abran et al., 2000] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.7. ACM/IEEE-CS, April 2000. Available on line at http://www.swebok.org [Última vez visitado 31-12-2001].

[Abran et al., 2001a] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair, Professional Practics Committee). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.9. ACM/IEEE-CS, February 2001. Available on line at http://www.swebok.org [Última vez visitado 31-12-2001].

[Abran et al., 2001b] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair, Professional Practics Committee). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Trial Version 0.95. IEEE-CS, May 2001. Available on line at http://www.swebok.org [Última vez visitado 28-12-2001].

[Abran et al., 2001c] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair). "*Guide to the Software Engineering Body of Knowledge SWEBOK*". IEEE-CS Press, 2001.

[ACM, 1965] ACM Curriculum Committee on Computer Science. "*An Undergraduate Program in Computer Science – Preliminary Recommendations*". Communications of the ACM, 8(9):543-552. September 1965.

[ACM, 1968] ACM Curriculum Committee on Computer Science. "*Curriculum '68: Recommendations for Academic Programs in Computer Science*". Communications of the ACM, 11(3):151-197. March 1968.

[ACM, 1981] ACM Committee on Computer Curricula of ACM Education Board. "*ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities*". ACM, New York, 1981.

[ACM/IEEE-CS, 1998] ACM/IEEE-CS. "*Accreditation Criteria for Software Engineering*". http://www.acm.org/serving/se/Accred.htm. [Última vez visitado 31/12/2001]. September 1998.

[ACM/IEEE-CS, 1999a] ACM/IEEE-CS. "*Software Engineering Education Project*". http://www.acm.org/serving/se/SWEE.htm. [Última vez visitado 31/12/2001]. 1999.

[ACM/IEEE-CS, 1999b] ACM/IEEE-CS. "*1999 Plan for the Software Engineering Education Project (SWEEP)*". Draft 0.5. http://www.acm.org/serving/se/sweep.htm. [Última vez visitado 31/12/2001]. April 1999.

[ACM/IEEE-CS, 1999c] ACM/IEEE-CS. "*Software Engineering Code of Ethics and Professional Practice*". Version 5.2. http://www.acm.org/serving/se/code.htm. [Última vez visitado 31/12/2001]. 1999.

[ACM/IEEE-CS, 2001] The Joint Task Force on Computing Curricula: IEEE Computer Society and Association for Computing Machinery. "*Computing Curricula 2001 – Computer Science*". Final Report, http://www.computer.org/education/cc2001/final/cc2001.pdf. [Última vez visitado, 27-12-2001]. December 15, 2001.

[Ada95-Web] "*Ada 95 Reference Manual: Language and Standard Library*". http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-toc.html [Última vez visitado 8/1/2000].

[Adams, 1996] Adams, J. C. "*Object-Centered Design. A Five-Phase Introduction to Object-Oriented Programming in CS1-2*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 78-82. ACM. 1996.

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[Aiken, 1980] Aiken, R. M. "*Computer Science Education in the 1980's*". IEEE Computer, 13(1):41-46. January 1980.

[Alhir, 1998] Alhir, S. S. "*The Object-Oriented Paradigm*". http://home.earthlink.net/~salhir/theobjectorientedparadigm.html. [Última vez visitado 23/12/1999]. October 1998.

[Ardis y Ford, 1989] Ardis, M., Ford, G. "*1989 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-89-TR-21 (ESD-TR-89-29), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). June 1989.

[Ashenhurst, 1972] Ashenhurst, R. L. (Editor). "*Curriculum Recommendations for Graduate Professional Programs in Information Systems*". ACM, 1972.

[Ashworth y Goodland, 1990] Ashworth, C., Goodland, M. "*SSADM: A Practical Approach*". McGraw-Hill, 1990.

[Atkinson et al., 1989] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S. "*The Object-Oriented Database System Manifesto*". In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto (Japan). 1989. Also in Deductive and Object-Oriented Databases, Elsevere Science Publishers, Amsterdam, Netherlands, 1990.

[Atkinson y Buneman, 1987] Atkinson, M., Buneman, P. "*Types and Persistence in Database Programming Languages*". ACM Computing Surveys, 19(2). 1987.

[Austing et al., 1979] Austing, R., Barnes, B., Bonnette, D., Engel, G., Stokes, G. "*Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*". Communications of the ACM, 22(3):147-166. March 1979.

[Baber, 1998] Baber, R. L. "*Software Engineering Education: Issues and Alternatives*". Annals of Software Engineering, 6:39-59. 1998.

[Bagert, 1999] Bagert, D. J. "*Taking the Lead in Licensing Software Engineers*". Communications of the ACM, 42(4):27-29, April 1999.

[Bagert et al., 1999] Bagert, D. J., Hilburn, T. B., Hislop, G., Lutz, M., McCracken, M., Mengel, S. "*Guidelines for Software Engineering Education. Version 1.0*". Working Group on Software Engineering Education and Training (WGSEET). August 1999.

[Bailin, 1989] Bailin, S. C. "*An Object-Oriented Requirements Specification Method*". Communications of the ACM, 32(5):608-623. May 1989.

[Barr y Feigenbaum, 1981] Barr, A., Feigenbaum, E. "*The Handbook of Artificial Intelligence*". Vol. 1. William Kaufmann, 1981.

[Basili, 1991] Basili, V. R. "*The Future Engineering of Software: A Management Perspective*". IEEE Computer, 24(9):90-96. September 1991.

[Bauer, 1972] Bauer, F. L. "*Software Engineering*". Information Processing 71. Amsterdam: North Holland, 1972.

[BCS, 1989] The British Computer Society and The Institution of Electrical Engineering. "*A Report on Undergraduate Curricula for Software Engineering Curricula*". June 1989.

[BCS, 1995] The British HCI Group. "*Whither HCI education in the UK?*". Interfaces, Nº28, Spring, 1995.

[Beizer, 2000] Beizer, B. "*Software Is Different*". Annals of Software Engineering, 10:293-310. 2000.

[Bellin, 1999] Bellin, D. "*Pedagogical Pattern #4. Brainstorming Pattern*". Version 2.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp4.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Bergin, 1998a] Bergin, J. "*Pedagogical Patterns*". http://csis.pace.edu/~bergin/PedPat1.2.html. [Última vez visitado, 3/8/1999]. October 1998.

[Bergin, 1998b] Bergin, J. "*Six Pedagogical Patterns*". http://csis.pace.edu/~bergin/fivepedpat.html. [Última vez visitado, 3/8/1999]. October 1998.

[Bergin, 1998c] Bergin, J. "*Pedagogical Pattern #32. Spiral Pattern*". Versión 1.2. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp32.htm. [Última vez visitado, 20/8/1999]. October 1998.

[Bertino y Martino, 1993] Bertino, E., Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

[Berztiss, 1987] Berztiss, A. "*A Mathematically Focused Curriculum for Computer Science*". Communications of the ACM, 5(30):356-365. May 1987.

[Bézivin et al., 1992] Bézivin, J., Roux, O., Royer, J.-C. "*Teaching Object-Oriented Programming or Using the Object Model to Teach Software Engineering*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 269-275. ACM. 1992.

[Birtwistle et al., 1973] Birtwistle, G. M., Dahl, O.-J., Myhrhaug, B., Nygaard, K. "*Simula Begin*". Studentliteratur, 1973.

[Blaha y Premerlani, 1998] Blaha, M., Premerlani, W. "*Object-Oriented Modeling and Design for Database Applications*". Prentice Hall, 1998.

[Blair et al., 1991] Blair, G., Gallagher, J., Hutchinson, D., Shepard, D. "*Object-Oriented Languages, Systems and Applications*". Halsted Press, 1991.

[Blum, 1992] Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, 1992.

[BOE, 1997] *Boletín Oficial del Estado de 4 de Noviembre de 1997*; Resolución de 15 de Octubre, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero Técnico en Informática de Sistemas.

[BOE, 1999] *Boletín Oficial del Estado de 1 de Julio de 1999*; Resolución de 10 de junio de 1999, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero en Informática (2º ciclo).

[Boehm, 1976] Boehm, B. W. "*Software Engineering*". IEEE Transactions on Computers. C-25(12):1226-1241. December 1976.

[Boehm, 1981] Boehm, B. W. "*Software Engineering Economics*". Prentice Hall, 1981.

[Booch, 1991] Booch, G. "*Object Oriented Design with Applications*". The Benjamin/Cummings Publishing Company, 1991.

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Booch et al., 1996a] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

[Booch et al., 1996b] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.91 Addendum UML update. Rational Software Corporation, September 1996.

[Booch et al., 1997a] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997.

[Booch et al., 1997b] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0.1. Rational Software Corporation, 19 March 1997.

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

[Booch y Rumbaugh, 1995] Booch, G., Rumbaugh, J. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

[Bourque et al., 1998] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L., Shyne, K., Pflug, B., Maya, M., Tremblay, G. "*Guide to the Software Engineering Body of Knowledge. A Straw Man Version*". ACM/IEEE-CS, September 1998.

[Bourque et al., 1999a] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L. "*The Guide to the Software Engineering Body of Knowledge*". IEEE Software, 16(6):35-44. November-December 1999.

[Bourque et al., 1999b] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L., Frailey, D. "*Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge*". Technical Report. January 1999.

[Bowyer, 1996] Bowyer, K. W. "*Ethics and Computing: Living Responsibly in a Computerized World*". IEEE Computer Society Press, 1996.

[Brandt, 1997] Brandt, D. S. "*Constructivism: Teaching for Understanding of the Internet*". Communications of the ACM, 40(10):112-117. October 1997.

[Bryant, 2000] Bryant, A. "*Metaphor, Myth and Mimicry: The Bases of Software Engineering*". Annals of Software Engineering, 10:273-292. 2000.

[Budd, 1991] Budd, T. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Buxton y Randell, 1970] Buxton, J. N., Randell, B. (Editors). "*Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 October, 1969*". Brussels: Scientific Affairs Division, NATO. April 1970.

[Buxton et al., 1976] Buxton, J. M., Naur, P., Randell, B. (Editors) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[Cameron, 1989] Cameron, J. "*JSP & JSD: The Jackson Approach to Software Development*". 2nd edition. IEEE Computer Society Press, 1989.

[Camps, 1999] Camps Paré, R. "*¿Qué Informática Se Enseña en la Universidad? (Primera Parte)*". Novática. Nº 141: 48-51. Septiembre/Octubre, 1999.

[Canos et al., 1998] Canós, J. H., Penadés, M. C., Pelechano, V., Sánchez, J., Ramos, I., González, A. "*La Ingeniería del Software en los Planes de Estudio: ¿Dos Perspectivas de una Disciplina o Más de lo Mismo?*". IV Jornades sobre L'Ensenyament Universitari de la Informática (JENUI'98). Páginas 105-112, San Julià de Loira, Andorra, 1998.

[Castellanos et al., 1991] Castellanos, M. G., Saltor, F., García-Solaco, M. "*The Development of Semantic Concepts in BLOOM Model Using an Object Metamodel*". Technical Report LSI-91-22. Dept. de Llenguatges i Sistemes Informátics. Universidad Politécnica de Cataluña, 1991.

[Coleman et al., 1994] Coleman, D., Arnold, P., Bodoff, S., Dolin, C., Hayes, F., Jeremaes, P. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

[Cook y Daniels, 1994] Cook, S., Daniels, J. "*Designing Object Systems: Object Oriented Modelling with Syntropy*". Prentice-Hall, 1994.

[Cook et al., 1997] Cook, S., Selic, B., Gangopadhyay, D., Gheorge, S., Gullekson, G., Hogg, J., McGee, J., Meier, M., Mitra, S., Saaltink, M., Warmer, J., Wills, A. "*OMG OA&D RFP OMG OA&D RFP Response*". Document Version 1.0. IBM Corporation and ObjecTime Limited. 10 January 1997.

[COSINE, 1967] COSINE Committee. "*Computer Science in Electrical Engineering*". Washington, DC: Commission on Engineering Education. September 1967.

[Couger, 1973] Couger, J. (Editor). "*Curriculum Recommendations for Undergraduate Programs in Information Systems*". Communications of the ACM, 16(12): 727-749. December 1973.

[Cowling, 1998] Cowling, A. J. "*A Multi-Dimensional Model of the Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education and Training – CSEE&T'98. (February 22-25, 1998, Atlanta, GA, USA). Pages 44-55. IEEE Computer Society. 1998.

[CSAB, 2000] Computing Sciences Accreditation Board. "*Criteria for Accrediting Programs in Computer Science in the United States*". Version 1,0, http://www.csab.org/criteria2k_v10.html. January 2000.

[Champeaux et al., 1993] Champeaux, D., Lea, D., Faure, P. "*Object-Oriented System Development*". Addison Wesley, 1993.

[Chang et al., 1999] Chang, C. K., Engel, G., King, W., Roberts, E., Shackelford, R., Sloan, R. H., Srimani, P. K. "*Curricula 2001: Bringing the Future to the Classroom*". Computer, 32(9):85-88. September 1999.

[Chen, 1976] Chen, P. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March 1976.

[Dahl et al., 1970] Dahl, O.-J., Myrhaug, B., Nygaard, K. "*(Simula 67) Common Base Language*". Norsk Regnesentral (Norwegian Computing Center), Publication N. S-22, Oslo. October 1970.

[Dahl y Hoare, 1972] Dahl, O.-J., Hoare, C. A. R. "*Hierarchical Program Structures*". In Dahl, Dijkstra, Hoare, *Structured Programming*, Academic Press, pages 175-220. 1972.

[Davis, 1982] Davis, A. M. "*Rapid Prototyping Using Executable Requirements Specifications*". ACM Software Engineering Notes, 7(5):39-44, 1982.

[Davis, 1992] Davis, A. M. "*Operational Prototyping: A new Development Approach*". IEEE Software, 9(5):70-78, 1992.

[Davis, 1993] Davis, A. M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

[Davis et al., 1997] Davis, G. B., Gorgone, J. T., Couger, J. D., Feinstein, D. L., Longenecker, Jr. H. E. (Editors). "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS y AITP, 1997.

[Decker y Hirshfield, 1994] Decker, R., Hirshfield, S. "*The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS1*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education (SIGCSE'94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 51-55. ACM. 1994.

[DeMarco, 1979] DeMarco, T. "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

[Denning, 1992] Denning, P. J. "*Educating a New Engineer*". Communications of the ACM, 35(12):82-97. December 1992.

[Denning, 1998] Denning, P. J. "*Computer Science and Software Engineering: Filing for Divorce?*". Communications of the ACM, 40(8):128. August 1998.

[Denning, 2001] Denning, P. J. "*Who Are We*". Communications of the ACM, 44(2):15-19. February 2001.

[Denning et al., 1988] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J., Young, P. R. "*Report of the ACM Task Force on the Core of Computer Science*". ACM Press, 1988.

[Denning et al., 1989] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J., Young, P. R. "*Computing as a Discipline*". Communications of the ACM, 32(1):9-23. January 1989.

[Denning y Dunham, 2001] Denning, P. J., Dunham, R. "*The Core of the Third-Wave Professional*". Communications of the ACM, 44(11):21-25. November 2001.

[Devlin, 2001] Devlin, K. "*The Real Reason Why Software Engineers Need Math*". Communications of the ACM, 44(10):21-22, October 2001.

[Dijkstra, 1968a] Dijkstra, E. "*Go to Statement Considered Harmful*". Communications of the ACM, 11(3):147-148. March 1968.

[Dijkstra, 1968b] Dijkstra, E. "*The Structure of 'THE' Multiprogramming System*". Communications of the ACM, 11(5):341-346. May 1968.

[Diller, 1990] Diller, A. "*Z: An Introduction to Formal Methods*". John Wiley & Sons, 1990.

[Dodani, 1999] Dodani, M. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January 1999.

[Dolado, 1999] Dolado, J. "*El Código de Ética y Práctica Profesional de la Ingeniería del Software de la ACM/IEEE Computer Society*". Novática. Nº 140. Julio-Agosto, 1999.

[Dorling, 1993] Dorling, A. "*Software Process Improvement and Capability Determination*". Software Quality Journal, 12(4): 209-224. December 1993.

[Dowell y Long, 1989] Dowell, J., Long, J. "*Towards a Conception of an Engineering Discipline of HCI*". Ergonomics, 32:1513-1535. 1989.

[DPMA, 1981] Data Processing Management Association. "*DPMA Model Curriculum, 1981*". Published by DPMA, Chicago, 1981.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Duncan, 1996] Duncan, W. R. "*A Guide to the Project Management Body of Knowledge*". PMI Standards Committee. Project Management Institute, Four Campus Boulevart, Newtown Square, PA 19073-3299, USA. 1996.

[Dupuis et al., 1999] Dupuis, R., Bourque, P., Abran, A., Moore, J. W., Tripp, L. L. "*Guide to the Software Engineering Body of Knowledge*". In 1999 Frontiers in Education Conference – FIE'99. (San Juan, Porto Rico, November 10-13, 1999). http://www.lrgl.uqam.ca/publications/pdf/467.pdf. [Última vez visitado, 31-12-2001]. 1999.

[Dupuis et al., 2000] Dupuis, R., Bourque, P., Tait, J., McGarry, J., DeWeese, P., Moore, J. W. "*Guide to the Software Engineering Body of Knowledge. Overview and Applications*". In STC 2000, The Twelfth Annual Software Technology Conference, Salt Lake City, Utah 30 April - 4 May 2000. http://www.lrgl.uqam.ca/publications/pdf/514.pdf. [Última vez visitado, 31-12-2001]. 2000.

[D'Souza, 1996] D'Souza, D. F. "*Objects: Education vs. Training*". ICON Computing Inc. http://www.iconcomp.com/papers/education-vs-training/EducationvsTraining.frm.html. [Última vez visitado, 24/5/1999]. 1996.

[D'Souza y Wills, 1999] D'Souza, D. F., Wills, A. C. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

[EAB, 1983] Education Activities Board. "*The 1983 Model Program in Computer Science and Engineering*". Technical Report 932. IEEE Computer Society. December 1983.

[Earles, 2000] Earles, J. "*Software Engineering – Myth or Reality*". http://www.cbd-hq.com. 2000.

[EC, 1977] Education Committee of the IEEE Computer Society. "*A Curriculum in Computer Science and Engineering*". Publication EHO119-8, Computer Society of the IEEE. January 1977.

[Ehrig y Mahr, 1985] Ehrig, H., Mahr, B. "*Fundamentals of Algebraic Specification 1*". Springer-Verlag, EATCS Nº6, 1985.

[El-Kadi, 1999] El-Kadi, A. "*Stop that Divorce*". Communications of the ACM, 42(12):27-28. December 1999.

[Fairley, 1985] Fairley, R. "*Software Engineering Concepts*". McGraw-Hill, 1985.

[Faulkner y Culwin, 2000] Faulkner, X., Culwin, F. "*Enter the Usability Engineer: Integrating HCI and Software Engineering*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITICSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 61-64. ACM Press, 2000.

[Firesmith et al., 1998] Firesmith, D., Henderson-Sellers, B., Graham, I. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

[Ford, 1990] Ford, G. "*1990 SEI Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-90-TR-3 (ESD-TR-90-204), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March 1990.

[Ford, 1991a] Ford, G. "*1991 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-91-TR-2 (ESD-TR-91-2), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April 1991.

[Ford, 1991b] Ford, G. "*The SEI Undergraduate Curriculum in Software Engineering*". In Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education – SIGCSE'91. (March 7-8, 1991, San Antonio, Texas, USA). Pages 375-385. ACM. 1991.

[Ford, 1994] Ford, G. "*A Progress Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-94-TR-11 (ESC-TR-94-011), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). May 1994.

[Ford y Gibbs, 1989] Ford, G., Gibbs, N. E. "*A Master of Software Engineering Curriculum*". IEEE Computer, 22(9):59-71, September 1989.

[Ford y Gibbs, 1996] Ford, G., Gibbs, N., E. "*Mature Profession of Software Engineering*". Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January 1996.

[Frakes et al., 1991] Frakes, W. B., Fox, C., Nejmeh, B. A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

[GAFC-USAL, 1997] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1997-1998*". Ediciones Universidad de Salamanca, 1997.

[GAFC-USAL, 1998] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1998-1999*". Ediciones Universidad de Salamanca, 1998.

[GAFC-USAL, 1999] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1999-2000*". Ediciones Universidad de Salamanca, 1999.

[GAFC-USAL, 2001] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 2001/2002*". Ediciones Universidad de Salamanca, 2001.

[Gane y Sarson, 1977] Gane, C., Sarson, T. "*Structured Systems Analysis and Design*". Improved Systems Technologies, Inc., 1977.

[Gane y Sarson, 1979] Gane, C., Sarson, T. "*Structured Systems Analysis: Tools and Techniques*". Prentice-Hall, 1979.

[García, 2000] García, F. J. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[García et al., 2000] García Peñalvo, F. J., Moreno García, Mª N., García-Bermejo Giner, J. R. y Luis Reboredo, A. de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García y Pardo, 1998] García Peñalvo, F. J., Pardo Aguilar, C. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Garlan et al., 1997] Garlan, D., Gluch, D. P., Tomayko, J. E. "*Agents of Change: Educating Software Engineering Leaders*". IEEE Computer, 30(11):59-65 November 1997.

[Gibbs, 1989] Gibbs, N. E. "*The SEI Education Program: The Challenge of Teaching Future Software Engineers*". Communications of the ACM, 32(5):594-605. May 1989.

[Gibbs y Tucker, 1986] Gibbs, N. E., Tucker, A. B. "*Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 29(3):202-210. March 1986.

[Ginige y Murugesan, 2001] Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. January-March 2001.

[Goguen et al., 1992] Goguen, J. A., Winkler, T., Meseguer, J., Futatsugui, K., Jouannaud, J. P. "*Introducing OBJ*". SRI-CSL Report. Draft of January 1992.

[Goguen y Meseguer, 1988] Goguen, J. A., Meseguer, J. "*Order-Sorted Álgebra I*". Technical Report, SRI International, Stanford University, 1988.

[Goldberg, 1985] Goldberg, A. "*Smalltalk-80: The Interactive Programming Environment*". Addison-Wesley, 1985.

[Goldberg, 1986] Goldberg, R. "*Software Engineering: An Emerging Discipline*". IBM Systems Journal. 25(3/4), 1986.

[Goldberg y Robson, 1983] Goldberg, A., Robson, D. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Gomaa, 1983] Gomaa, H. "*The Impact of Rapid Prototyping on Specifying User Requirements*". ACM Software Engineering Notes, 8(2):17-28, 1983.

[Gómez et al., 1998] Gómez, A., Juristo, N., Montes, C. y Pazos, J. "*Ingeniería del Conocimiento*". Madrid. Ceura, 1998.

[Gorgone et al., 1999] Gorgone, J. T., Gray, P., Feinstein, D. L., Kasper, G. M., Luftman, J. N., Stohr, E. A., Valacich, J., Wigand, R. T. "*MSIS 2000 Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems*". ACM and AIS. November 1999.

[Gotterbarn, 1999] Gotterbarn, D. "*How the New Software Engineering Code of Ethics Affects You*". IEEE Software, 16(6):58-64. November/December 1999.

[Gotterbarn et al., 1997a] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics*". Communications of the ACM, 40(11):110-118. November 1997.

[Gotterbarn et al., 1997b] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics, Version 3.0*". IEEE Computer, 30(11):88-92. November 1997.

[Gotterbarn et al., 1999a] Gotterbarn, D., Miller, K., Rogerson, S. "*Computer Society and ACM Approve Software Engineering Code of Ethics*". IEEE Computer, 32(10):84-88. October 1999.

[Gotterbarn et al., 1999b] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics Is Approved*". Communications of the ACM, 42(10):102-107. October 1999.

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd edition. Addison-Wesley, 1994.

[Graham et al., 1997] Graham, I., Henderson-Sellers, B., Younessi, H. "*The Open Process Specification*". Addison Wesley (Open Series), 1997.

[Guttag, 1980] Guttag, J. "*Abstract Data Types and the Development of Data Structures*". In *Programming Language Design*. Computer Society Press, 1980.

[Hadjerrouit, 1999] Hadjerrouit, S. "*A Constructivist Approach to Object-Oriented Design and Programming*". In Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education (ITiCSE'99). (June 27-July 1, 1999, Cracow, Poland). Pages 171-174. ACM. 1999.

[Hartmanis, 1994] Hartmanis, J. "*On Computational Complexity and the Nature of Computer Science*". Communications of the ACM, 37(10):198-202. October 1994.

[Hatley y Pirbhai, 1987] Hatley D. J., Pirbhai, A. "*Strategies for Real-Time System Specification*". Dorset House, 1987.

[Hefley et al., 1992] Hefley, B. (editor), Hewett, T. T. (chair), Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W. "*Curricula for Human-Computer Interaction*". ACM SIGCHI. http://sigchi.org/cdg/. [Última vez visitado, 4/1/2002]. ACM Press, 1992.

[Henderson-Sellers y Edwards, 1994a] Henderson-Sellers, B., Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

[Henderson-Sellers y Edwards, 1994b] Henderson-Sellers, B., Edwards, J. M. "*MOSES: A Second Generation Object-Oriented Methodology*". Object Magazine, pp. 68-73. June 1994.

[Henderson-Sellers et al., 1998] Henderson-Sellers, B., Simons, A., Younessi, H. "*The Open Toolbox of Techniques*". Open Series. Addison Wesley, 1998.

[Hilburn, 1997] Hilburn, T. B. "*Software Engineering Education: A Modest Proposal*". IEEE Software, 14(6):44-48. November/December 1997.

[Hilburn et al., 1998] Hilburn, T. B., Bagert, D. J., Mengel, S., Oexmann, D. "*Software Engineering Across Computing Curricula*". In Proceedings of the 6th Annual Conference on the Teaching of Computing/3rd Annual Conference on Integrating Technology into Computer Science Education on Changing the Delivery of Computer Science Education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 117-121. ACM. 1998.

[Hilburn et al., 1999] Hilburn, T. B., Hirmanpour, I., Khajenoori, S., Turner, R., Qasem, A. "*A Software Engineering Body of Knowledge Version 1.0*". Technical Report CMU/SEI-99-TR-004 (ESC-TR-99-004), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April 1999.

[Hirmanpour et al., 1995] Hirmanpour, I. Hilburn, T. B., Kornecki, A. "*A Domain Centered Curriculum. An Alternative Approach to Computing Education*". In Proceedings of the 26th SISCSE Technical Symposium on Computer Science Education, SIGCSE'95. (March 2-4, 1995, Nashville, TN, USA). ACM. 1995.

[Hoare, 1975] Hoare, C. A. R. "*Software Engineering*". Computer Bulletin. Pages 6-7. December 1975.

[Hoare, 1985] Hoare, C. A. R. "*Communicating Sequential Processes*". Prentice-Hall, 1985.

[Holland et al., 1997] Holland, S., Griffiths, R., Woodman, M. "*Avoiding Object Misconceptions*". In Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 131-134. 1997.

[Hopcroft, 1987] Hopcroft, J. E. "*Computer Science: The Emergence of a Discipline*". Communications of the ACM, 30(3):198-202. March 1987.

[Horan, 1995] Horan, P. "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[Humphrey, 1989] Humphrey, W. S. "*Managing the Software Process*". Addison-Wesley, 1989.

[Humphrey, 1993] Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science*, Van Nostrand Reinhold, p. 1218,1993.

[Humphrey, 2000] Humphrey, W. S. "*Software – A Performing Science?*". Annals of Software Engineering, 10:261-271. 2000.

[IEEE, 1983] IEEE. "*Standard Glossary of Software Engineering Terminology*". ANSI/IEEE Std. 729-1983. IEEE, 1983.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[IEEE-CS, 2001a] IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part I*". Vol.8, N1, January-March 2001.

[IEEE-CS, 2001b] IEEE-CS. "IEEE Multimedia. Special Issue on Web Engineering Part II". Vol.8, N2, April-June 2001.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

[Jackson, 1983] Jackson, M. A. "*System Development*". Prentice-Hall, 1983.

[Jackson et al., 1997] Jackson, U., Manaris, B., McCauley, R. "*Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum.*". In Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education, SIGCSE'97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 360-364. ACM. 1997.

[Jalics y Golden, 1995] Jalics, P., Golden, D. "*A Profile of Undergraduate Computer Science Curricula*". Computer Science Education, 6(2):179-192. November 1995.

[Jalloul, 1999] Jalloul, G. "*Pedagogical Pattern #48. Academic to Industrial Project Link (LINK) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp48.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Jones, 1987] Jones, A. K. "*The Object Model: A Conceptual Tool for Structuring Software*". In Gerald E. Peterson, editor, *TUTORIAL: Object-Oriented Computing*, Volume 2: Implementations. IEEE Computer Society Press, 1987.

[Jones, 1991] Jones, C. B. "*Systematic Software Construction Using VDM*". Prentice-Hall, 1991.

[Kirby et al., 1994] Kirby, M., Life, A., Istance, H., Hole L., Crombie, A. "*HCI Curricula in the UK: What Is Being Taught and What Should Be Taught*". BCS HCI Curriculum WG Interim Report. London: BCS. 1994.

[Kirby et al., 1995] Kirby, M., Life, A., Istance, H., Hole L., Crombie, A. "*HCI Curricula: What Is Being Taught on Computing Courses in the UK*". In Proceedings of Interact '95, Lillehammer, Norway. 1995.

[Knight et al., 1994] Knight, J. C., Prey, J. C., Wulf, W. A. "*Undergraduate Computer Science Education: A New Curriculum Philosophy & Overview*". In Proceedings of the Twenty-fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 155-159. ACM. 1994.

[Knoke y Bagert, 1998] Knoke, P., Bagert, D. "*Graduate Software Engineering Program Survey Results & Evaluation*". Forum for Advancing Software Engineering Education (FASE), 8(9), September 1998. (http://www.cs.ttu.edu/fase/v8n09.txt).

[Kobryn, 1999] Kobryn, C. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October 1999.

[Kobryn, 2002] Kobryn, C. "*Will UML 2.0 Be Agile or Awkward?*". Communications of the ACM, 45(1):107-110. January 2002.

[Koffman et al., 1984] Koffman, E., Miller, P., Wardle, C. "*Recommended Curriculum for CS1: 1984*". Communications of the ACM, 27(10):998-1001. October 1984.

[Koffman et al., 1985] Koffman, E., Miller, P., Wardle, C. "*Recommended Curriculum for CS2: 1984*". Communications of the ACM, 28(8):815-818. August 1985.

[Konrad et al., 1995] Konrad, M. D., Paulk, Mark C., Graydon, A. W. "*An Overview of SPICE's Model for Process Management*". In Proceedings of the Fifth International Conference on Software Quality, Austin, TX, 23-26 October 1995.

[Lalonde y Pugh, 1990] Lalonde, W. R., Pugh, J. R. "*Inside Smalltalk*". Vol. 1. Prentice-Hall, 1990.

[Lalonde y Pugh, 1991] Lalonde, W. R., Pugh, J. R. "*Inside Smalltalk*". Vol. 2. Prentice-Hall, 1991.

[Layman, 1994] Layman, B. "*ISO-9000 Standards and Existing Quality Models: How They Relate*". American Programmer Review, pp. 9-15. February 1994.

[Le Moigne, 1973] Le Moigne, J. L. "*Les Systemes D'Information dan les Organizations*". Presses Universitaires de France, 1973.

[Lebsanft y Synspace, 1994] Lebsanft, E., Synspace, A. G. "*BOOTSTRAP: Experiences with Europe's Software Process Assesment & Improvement Method*". In Proceedings of the 1st World Congress for Software Quality, 1994.

[Letelier et al., 1998] Letelier, P., Sánchez, P., Pastor, O., Ramos, I. "*OASIS Versión 3: Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-98.4011, 1998.

[Levine et al., 1991] Levine, L., Pesante, L. H., Dunkle, S. B. "*Technical Writing for Software Engineers*". Curriculum Module, SEI-CM-23, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). November 1991.

[Levy, 1984] Levy, H. "*Capability-Based Computer Systems*". Digital Press, 1984.

[Lewerentz y Rust, 2000] Lewerentz, C., Rust, H. "*Are Software Engineers True Engineers?*". Annals of Software Engineering, 10:311-328. 2000.

[Lewis, 1994] Lewis, T.G. "*Where Is Computing Headed?*". IEEE Computer, 27(8):59-63. August 1994.

[Lilly, 1996] Lilly, S. "*Patterns for Pedagogy*". Object Magazine, 5(8):93-96. January 1996.

[Liskov, 1993] Liskov, B. "*A History of CLU*". History of Programming Languages - The Second ACM SIGPLAN Conference on History of Programming Languages. (April 20 - 23, 1993, Cambridge United States). Pages 133-147. ACM Press, 1993.

[Liskov y Zilles, 1977] Liskov, B., Zilles, S. "*An Introduction to Formal Specifications of Data Abstractions*". Current Trends in Programming Methodology: Software Specification and Design. Vol. 1. Prentice-Hall, 1977.

[Longenecker y Feinstein, 1991] Longenecker, Jr. H. E., Feinstein, D L. (Editors). "*IS'90 The DPMA Model Curriculum for a Four Year Undergraduate Degree for the 1990s*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1991.

[Longenecker et al., 1994] Longenecker, Jr. H. E., Fournier, R., Reaugh, W. R., Feinstein D. L. (Editors). "*IS'94 The DPMA Two Year Model Curriculum for IS Professionals*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1994.

[Magel et al., 1981] Magel, K. I., Austing, R. H., Berztiss, A., Engel, G. L., Hamblen, J. W., Hoffmann, A. A. J., Mathis, R. "*Recommendations for Master's Level Programs in Computer Science*". Communications of the ACM, 24(3):115-123. March 1981.

[Manns, 1999] Manns, M. L. "*Pedagogical Pattern #8. Lab-Discussion-Lecture-Lab (LDLL) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp8.htm. [Última vez visitado, 20/8/1999]. July 1999.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[MAP, 2001a] Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

[MAP, 2001b] Ministerio de Administraciones Públicas. "*MÉTRICA. Versión 3. Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información*". http://www.map.es/csi/metrica3/. [Última vez visitado, 20-10-2001]. 2001.

[Marqués, 1995] Marqués Corral, J. M. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

[McDermid, 1991] McDermid, J. (Editor). "*The Software Engineer's Reference Book*". Butterworth Heinemann, 1991.

[Mead y Turner, 1998] Mead, N. R., Turner, A. J. "*Current Accreditation, Certification, and Licensure Activities Related to Software Engineering*". Annals of Software Engineering, 6:167-180. 1998.

[Meyer, 1990] Meyer, B. "*Introduction to the Theory of Programming Languages*". Prentice Hall, 1990.

[Meyer, 1996] Meyer, B. "*Teaching Object Technology*". IEEE Computer, 29(12):117. December 1996.

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Meyer, 2001] Meyer, B. "*Software Engineering in the Academy*". IEEE Computer, 34(5):28-35, May 2001.

[Michael, 2000] Michael, M. "*Fostering and Assessing Communication Skills in the Computer Science Context*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 119-123. ACM Press. 2000.

[Molina, 2000] Molina Marco, A. "*Proyecto Docente. Metodología y Tecnología de la Programación*". Área de Conocimiento de Lenguajes y Sistemas Informáticos. Universidad Politécnica de Valencia. Marzo, 2000.

[Mulder y Dalphin, 1984] Mulder, M. C., Dalphin, J. "*Computer Science Program Requirements and Accreditation – An Interim Report of the ACM/IEEE Computer Society Joint Task Force*". Communications of the ACM, 27(4):330-335, April 1984.

[Mynatt, 1990] Mynatt, B. "*Software Engineering with Students Project Guidance*". Prentice Hall International, 1990.

[Naur y Randell, 1969] Naur, P., Randell, B. (Editors). "*Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968*". Brussels: Scientific Affairs Division, NATO. January 1969.

[Ng, 1999] Ng, C. "*Experience Qualification Guidelines for Registration through the Computer and Software Engineering Disciplines*". Draft. Technical Report. Association of Professional Engineers y Geoscientists, British Columbia, March 1999.

[Norman, 1988] Norman, D. A. "*The Psychology of Everyday Things*". New York: Basic Books, Inc. 1988.

[Northrop, 1992] Northrop, L. M. "*Finding an Educational Perspective for Object-Oriented Development*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 245-249. ACM. 1992.

[Nunamaker, 1981] Nunamaker, J. F. (Editor). "*Educational Programs in Information Systems. A Report of the ACM Curriculum Committee on Information Systems*". Communications of the ACM, 24(3):124-133. March 1981.

[Nunamaker et al., 1982] Nunamaker, J. F., Couger, J. D., Davis, G. B. "*Information System Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs*". Communications of the ACM 25(11):781-805. November 1982.

[OMG, 1998] OMG. "*OMG Unified Modeling Language Specification. Version 1.2*". Object Management Group Inc. ftp://ftp.omg.org/pub/docs/ad/98-12-02-pdf. July 1998.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. [Última vez visitado, 31/12/2001]. June 1999.

[OMG, 2000a] OMG. "*OMG UML 2.0 Infraestructure RFP*". Document ad/00-09-01. September 2000.

[OMG, 2000b] OMG. "*OMG UML 2.0 Superstructure RFP*". Document ad/00-09-02. September 2000.

[OMG, 2000c] OMG. "*OMG UML 2.0 OCL RFP*". Document ad/00-09-03. September 2000.

[OMG, 2001a] OMG. "*OMG UML 2.0 Diagram Interchange RFP*". Document ad/01-02-09. February 2001.

[OMG, 2001b] OMG. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. [Última vez visitado, 01/09/2001]. September 2001.

[Orr, 1977] Orr, K. T. "*Structured System Development*". Yourdon Press, 1977.

[Orr, 1981] Orr, K. T. "*Structured Requirements Definition*". Ken Orr & Associates, 1981.

[Osborne, 1992] Osborne, M. "*The Role of Object-Oriented Technology in the Undergraduate Computer Science Curriculum*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 303-308. ACM. 1992.

[Pancake, 1995] Pancake, C. M. "*The Promise and the Cost of Object Technology: A Five Years Forecast*". Communications of the ACM 38(10):32-49. October 1995.

[Parnas, 1972] Parnas, D. L. "*On the Criteria To Be Used in Decomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972.

[Parnas, 1990] Parnas, D. L. "*Education for Computing Professionals*". IEEE Computer, 23(1):17-22, January 1990.

[Parnas, 1998] Parnas, D. L. "*Software Engineering Programmes Are Not Computer Science Programmes*". Annals of Software Engineering, 6(1):19-37, January 1998.

[Parnas, 2001] Parnas, D. L. "*Why Software Developers Should Be Licensed*". Engineering Dimensions, 36-39. May-June 2001.

[Parnas y Weiss, 1987] Parnas, D. L., Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4): 259-265, December 1987.

[Parrish et al., 1998] Parrish, A., Borie, R., Cordes, D., Dixon, B., Hale, D., Hale, J., Jackson, J., Sharpe, S. "*Computer Engineering, Computer Science and Management Information Systems: Partners in a Unified Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education & Training – CSEET'98. (February 22-25, 1998. Atlanta, GA – USA). Pages 67-75. IEEE Computer Society. 1998.

[Pastor et al., 1997] Pastor, O., Insfrán, E., Pelechano, V., Romero, J., Merseguer, J. "*OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods*". In Advanced Information Systems Engineering (9th International Conference, CAiSE97 – Barcelona, Catalonia, Spain, June 1997). A. Olivé, J. A. Pastor (editors). Pages 145-158. Lecture Notes in Computer Science – LNCS 1250. Springer-Verlag. 1997.

[Pastor y Ramos, 1995] Pastor, O., Ramos. I. "Oasis 2.1.1: *A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain) SPUPV-95.788, 1995.

[Paulk et al., 1993a] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

[Paulk et al., 1993b] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

[Pham, 1997] Pham, B. "*The Changing Curriculum of Computing and Information Technology in Australia*". In Proceedings of the Second Australasian Conference on Computer Science Education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. 1997.

[Piattini, 1994] Piattini Velthuis, M. G. "*Definición de una Metodología para el Desarrollo de Bases de Datos Orientadas al Objeto Fundamentadas en Extensiones del Modelo Relacional*". Tesis Doctoral. Facultad de Informática, Universidad Politécnica de Madrid. 1994.

[Piattini et al., 1996] Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pollock, 2001] Pollock, L. "*Integrating an Intensive Experience with Communications Skills Development into a Computer Science Course*". In Proceedings of the Thirty Second SIGCSE Technical Symposium on Computer Science Education, SIGCSE'2001. (Charlotte, NC – USA, 2001). Pages 287-291. ACM, 2001.

[Pressman, 1987] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 2nd edition. McGraw Hill, 1987.

[Pressman, 1992] Pressman, R. S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Pressman, 2000] Pressman, R. S. "*Software Engineering: A Practitioner's Approach – European Adaptation*". 5th Edition. McGraw-Hill, 2000.

[Prieto y Victory, 1999] Prieto, M., Victory, P. "*Pedagogical Pattern #20. Identity Pattern*". Version 1.0. http://www-lifia.info.unlp.edu.ar/ppp/pp20.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Proto-Patterns, 1999] "*The Pedagogical Patterns Project. Successes in Teaching Object-Technology (PROTO-PATTERNS)*". http://www-lifia.info.unlp.edu.ar/ppp/index.html. [Última vez visitado, 20/8/1999]. July 1999.

[Ralston, 1984] Ralston, A. "*The First Course in Computer Science Needs a Mathematics Corequisite*". Communications of the ACM, 10(27):1002-1005. October 1984.

[Ralston y Shaw, 1980] Ralston, A., Shaw, M. "*Curriculum '78 – Is Computer Science Really that Unmathematical?*". Communications of the ACM, 23(2):67-70. February 1980.

[Ramamoorthy y Sheu, 1988] Ramamoorthy, C., Sheu, P. "*Object-Oriented Systems*". IEEE Expert, 3(3). Fall 1988.

[Rand, 1979] Rand, A. "*Introduction to Objectivist Epistemology*". New American Library, 1979.

[Randell, 1998] Randell, B. "*Memories of the NATO Software Engineering Conference*". In *Anecdotes Column*, James E. Tomayko (Editor). IEEE Annals of the History of Computing, 20(1):51-54. January-March 1998.

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Rayside y Campbell, 2000] Rayside, D., Campbell, G. T. "*Aristotle and Object-Oriented Programming: Why Modern Students Need Traditional Logic*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 237-244. ACM Press. 2000.

[Redwine, 1985] Redwine, S. T. "*Software Technology Maduration*". In Proceedings of the 1st International Conference on Software Engineering, Washington D. C. (USA). IEEE, 1985.

[Reenskaug et al., 1996] Reenskaug, T., Wold, P., Lehne, O. A. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

[Reynolds y Fox, 1996] Reynolds, C., Fox, C. "*Requirements for a Computer Science Curriculum Emphasizing Information Technology Subject Area: Curriculum Issues*". In Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 247-251. ACM. 1996.

[Rivas et al., 1997] Rivas, E., DeSilva, D., McDaniel, T., Atkinson, C. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January 1997.

[Roberts et al., 1999] Roberts, E., Shackelford, R., LeBlanc, R., Denning, P. J. "*Curriculum 2001: Interim Report from the ACM/IEEE-CS Task Force*". In Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, SIGCSE'99. (March 24-28, 1999, New Orleans, LA - USA). Pages 343-344. ACM. 1999.

[Robillard y Robillard, 1998] Robillard, P. N., Robillard, M. "*Improving Academic Software Engineering Projects: A Comparative Study of Academic and Industry Projects*". Annals of Software Engineering, 6:343-363. 1998.

[Rosson y Carroll, 1996] Rosson, M. B., Carroll, J. M. "*Scaffolded Examples for Learning Object-Oriented Design*". Communications of the ACM, 39(4):46-47. April 1996.

[Rumbaugh, 1987] Rumbaugh, J. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications – OOPSLA'87. (October 4 - 8, 1987, Orlando, FL USA). Pages 466-481. ACM Press, 1987.

[Rumbaugh, 1988] Rumbaugh, J. "*Controlling Propagation of Operations Using Attributes on Relations*". In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications – OOPSLA'88. (September 25 - 30, 1988, San Diego, CA USA). Pages 285-296. ACM Press, 1988.

[Rumbaugh, 1996] Rumbaugh, J. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., Booch, G. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

[SAC, 1967] President's Science Advisory Commission. "*Computers in Higher Education*". Washington DC: The White House. February 1967.

[Sánchez et al., 1998] Sánchez, P., Letelier, P., Ramos, I., Pastor, O. "*OASIS 3.0: Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Actas de las III Jornadas de Trabajo MENHIR, Moros, B. y Sáez, J. (editores) (13 y 14 de noviembre de 1998, Murcia - Spain). Páginas 63-74. 1998.

[Schmauch, 1994] Schmauch, C. "*ISO9000 for Software Developers*". IEEE Computer Society Press, 1994.

[Scragg et al., 1994] Scragg, G., Baldwin, D., Koomen, H. "*Computer Science Needs and Insight-Based Curriculum*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 150-154. ACM. 1994.

[Sernadas et al., 1989] Sernadas, A., Fiadeiro, J., Sernadas, C., Eric, H.-D. "*The Basic Building Blocks of Information Systems*". In *Information Systems Concepts*. North Holland, Namur, 1989.

[Shackelford y LeBlanc, 1994] Shackelford, R. L., LeBlanc, R. J. "*Integrating 'Depth First' and 'Breadth First' Models of Computing Curricula*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 6-10. ACM. 1994.

[Sharp et al., 2000] Sharp, H., Robinson, H., Woodman, M. "*Software Engineering: Community and Culture*". IEEE Software, 17(1):40-47. January/February 2000.

[Shaw, 1984] Shaw, M. "*Abstraction Techniques in Modern Programming Languages*". IEEE Software, 1(4). October 1984.

[Shaw, 1985] Shaw, M. (Editor). "*The Carnegie-Mellon Curriculum for Undergraduate Computer Science*". Springer-Verlag, 1985.

[Shaw, 1990] Shaw, M. "*Prospects for an Engineering Discipline of Software*". IEEE Software, 7(6):15-24. November 1990.

[Shaw, 1992] Shaw, M. "*We Can Teach Software Better*". Computing Research News, 4(4):2-12, September 1992.

[Shaw, 1998] Shaw, M. "*A Profession of Software Engineering: Is There a Need? YES; Are We Ready? NO*". In Proceedings of the ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering, SIGSOFT'98. (Nov. 3-5, 1998, Lake Buena Vista, FL). Pages 207-208, ACM Press, 1998.

[Shaw, 2000] Shaw, M. "*Software Engineering Education: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 371-380. ACM Press, 2000.

[Shaw y Garlan, 1996] Shaw, M., Garlan, D. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996.

[Shaw y Tomayko, 1991] Shaw, M., Tomayko, J. E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August 1991.

[Shlaer y Mellor, 1992] Shlaer, S., Mellor, S. "*Object Lifecycles: Modeling the World in States*". Yourdon Press, 1992.

[Simons et al., 1991] Simons, B., Frailey, D. J., Turner, A. J., Zweben, S. H., Denning, P. J. "*An ACM Response: The Scope and Directions of Computer Science*". Communications of the ACM, 34(10):121-131. October 1991

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May 1987.

[Smith y Tockey, 1988] Smith, M., Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

[Sommerville, 1985] Sommerville, I. "*Software Engineering*". 2nd edition. Addison-Wesley, 1985.

[Sommerville, 1989] Sommerville, I. "*Software Engineering*". 3rd edition. Addison-Wesley, 1989.

[Sommerville, 1996] Sommerville, I. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

[Sommerville, 2001] Sommerville, I. "*Software Engineering*". 6th edition. Addison-Wesley, 2001.

[Spivey, 1989] Spivey, J. M. "*The Z Notation. A Reference Manual*". International Series in Computer Science. Prentice-Hall International, 1989.

[Stevens et al., 2000] Stevens, K. T., Henry, J., Lawhead, P. B., Lewis, J., Bland, C., Peters, M. J. "*Using Large Projects in a Computer Science Curriculum*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 399-400. ACM Press. 2000.

[Stillings et al., 1987] Stillings, N., Feinstein, M., Garfield, J., Rissland, E., Rosenbaum, D., Weisler, S., Baker-Ward, L. "*Cognitive Science: An Introduction*". The MIT Press, 1987.

[SUN, 2002] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 12/04/2002]. March 2002.

[Sutcliffe, 1995] Sutcliffe, A. "*Human Computer Interaction*". Macmillan, 1995.

[Sutcliffe y Maiden, 1998] Sutcliffe, A., Maiden, N. "*The Domain Theory for Requirements Engineering*". IEEE Transactions on Software Engineering, 24(3): 174-196. March 1998.

[Toffler y Toffler, 1990] Toffler, A., Toffler, H. "*The Third Wave*". Bantam Books, 1990.

[Tardieu et al., 1986] Tardieu, H., Rochfeld, A., Coletti, R. "*La Méthode Merise. Tomo 1. Principes et Outils*", Editions d'Organisation, 1986.

[Tewari y Friedman, 1992] Tewari, R., Friedman, F. "*The Impact of Object-Oriented Software Engineering in the Introductory Computer Science Curriculum*". In addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 289-292. ACM. 1992.

[Tobin y Tippins, 1993] Tobin, K., Tippins, D. "*Constructivism as a Referent for Teaching and Learning*". In Tobin, K. (ed.). *The Practice of Constructivism in Science Education*. Pages 3-21. AAAS Press, Washington D. C. 1993.

[Tomayko, 1987] Tomayko, J. E. "*Teaching a Project-Intensive Introduction to Software Engineering*". Technical Report CMU/SEI-87-TR-20 (ESD-TR-87-171), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August 1987.

[Tomayko, 1998] Tomayko, J. E. "*Forging a Discipline: An Outline History of Software Engineering Education*". Annals of Software Engineering, 6:3-18, 1998.

[Tomayko, 2000] Tomayko, J. E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

[Tucker et al., 1991a] Tucker, A. B., Barnes, B. H., Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*Computing Curricula 1991*". ACM Press. February 1991.

[Tucker et al., 1991b] Tucker, A. B., Barnes, B. H., Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*A Summary of the ACM/IEEE-CS Joint*

Curriculum Task Force Report. Computing Curricula 1991". Communications of the ACM, 34(6):68-84. June 1991.

[Tucker et al., 1996] Tucker, A. B., Astrachan, O., Bruce, K., Cupper, R., Denning, P., Drysdale, S., Horton, T., Kelemen, C., McGeoch, C., Patt, Y., Proulx, V., Rada, R., Rasala, R., Roberts, E., Rudich, S., Stein, L., Van Loan, C. "*Strategic Directions in Computer Science Education*". ACM Computing Surveys, 28(4):836-845. December 1996.

[Tucker y Barnes, 1991] Tucker, A. B., Barnes, B. H. "*Flexible Design: A Summary of Computing Curricula 1991*". IEEE Computer, 24(11):56-66, November 1991.

[Tucker y Wegner, 1994] Tucker, A. B., Wegner, P. "*New Directions in the Introductory Computer Science Curriculum*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 11-15. ACM. 1994.

[Tymann et al., 1994] Tymann, P. T., Lea, D., Raj, R. K. "*Developing an Undergraduate Software Engineering Program in a Liberal Arts College*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 276-280. ACM. 1994.

[Vaitkevitchius, 1999] Vaitkevitchius, R. "*Pedagogical Pattern #25. BASE-and-Supplementary-Languages in Lectures (BSLL)*". In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp25.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Vaughn, 2000] Vaughn, R. B. Jr. "*Software Engineering Degree Programs*". Crosstalk, The Journal of Defense Software Engineering, 13(3):7-9. March 2000.

[Walker y Schneider, 1996] Walker, H. M., Schneider, G. M. "*A Revised Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 39(12):85-95. December 1996.

[Ward, 1989] Ward, P. T. "*How to Integrate Object Orientation with Structured Analysis and Design*". IEEE Software, 6(2):74-82. March/April 1989.

[Ward y Mellor, 1985] Ward, P., Mellor, S. "*Structured Development for Real-Time Systems*". Vols. 1-3. Yourdon Press, 1985.

[Warnier, 1974] Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

[Wasserman, 1996] Wasserman, A. "*Toward a Discipline of Software Engineering*". IEEE Software, 13(6):23-31. November/December 1996.

[WebE, 2001] "*Actas del I Taller sobre Ingeniería del Software Orientada al Web (Web Engineering*". Almagro, Ciudad Real, 22 de noviembre de 2001. http://www.dlsi.ua.es/webe01. [Última vez visitado, 7/1/2002]. 2001.

[Wegner, 1990] Wegner, P. "*Concepts and Paradigms of Object-Oriented Programming*". OOPS Messenger, 1(1). August 1990.

[Weinberg, 1971] Weinberg, G. F. "*The Psychology of Computer Programming*". Van Nostrand Reinhold, 1971.

[West, 1997] West, D. "*Hermeneutic Computer Science*". Communications of the ACM, 40(4):115-116. April 1997.

[Wielinga et al. 1991] Wielinga, B. J., Schreiber, A. T., Breuker, J. A. "*KADS: A Modeling Approach to Knowledge Engineering*". Technical Report ESPRIT Project P5248 KAD-II, 1991.

[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wirfs-Brock y Johnson, 1990] Wirfs-Brock, R., Johnson, R. E. "*Surveying Current Research in Object-Oriented Design*". Communications of the ACM, 33(9):104-124. September 1990.

[Woodman et al., 1996] Woodman, M., Davies, G., Holland, S. "*The Joy of Software – Starting with Objects*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 88-92. ACM. 1996.

[Yonezawa y Tokoro, 1987] Yonezawa, A., Tokoro, M. "*Object-Oriented Concurrent Programming: An Introduction*". In *Object-Oriented Concurrent Programming*. Cambridge, MA: The MIT Press, 1987.

[Yourdon, 1989] Yourdon, E. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Yourdon Inc., 1993] Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions, 1993.

[Yourdon y Constantine, 1975] Yordon, E., Constantine, L. "*Structured Design*". 1st edition. Prentice Hall, 1975.

[Yourdon y Constantine, 1979] Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

[Yourdon y Constantine, 1989] Yordon, E., Constantine, L. "*Structured Design*". 2nd edition. Yourdon Press, 1989.

[Zelkovitz et al., 1979] Zelkovitz, M. V., Shaw, A. C., Gannon, J. D. "*Principles of Software Engineering and Design*". Prentice-Hall, 1979.

Capítulo 5
Programación Docente

En este capítulo se concreta la planificación docente de las asignaturas "Ingeniería del Software", "Análisis de Sistemas" y "Administración de Proyectos Informáticos" tomando como base los fundamentos expuestos en el capítulo anterior referentes al marco histórico y conceptual de la materia Ingeniería del Software, en la que se engloban estas asignaturas. En la elaboración del programa se han tenido en cuenta, además de las recomendaciones recogidas en las descripciones del cuerpo de conocimiento de la Ingeniería del Software y en los diferentes currículos internacionales, los siguientes puntos:

- *La formación previa de los posibles alumnos.*
- *El Plan de Estudios vigente.*
- *Los programas de asignaturas similares de otras Universidades españolas.*

Antes de introducir la programación de las asignaturas correspondientes al perfil de la plaza se van a relacionar las mismas con otras asignaturas de la unidad docente de "Ingeniería del Software y Orientación a Objetos" cuyo plan de calidad se incluye en el apéndice A de esta memoria. Seguidamente se presentan, para cada asignatura, los objetivos generales que se pretenden alcanzar, seguidos de la programación teórico-práctica que se propone para su consecución. Para cada tema de teoría se presenta su duración aproximada, el programa, los objetivos y la bibliografía, tanto básica como complementaria.

5.1 Objetivos del Programa Docente

Los objetivos del programa propuesto se dividen en dos grupos: *objetivos generales* y *objetivos específicos*.

5.1.1 Objetivos generales

Los objetivos generales que se formulan son válidos para las asignaturas *Ingeniería del Software, Análisis de Sistemas* y *Administración de Proyectos Informáticos,* que, de una forma directa, recogen los créditos de la materia de Ingeniería del Software tanto en la titulación Ingeniería Técnica en Informática de Sistemas como en la titulación Ingeniería Informática en la Universidad de Salamanca.

El **objetivo global** es producir profesionales que puedan resolver de forma sistemática y ordenada la producción de software de calidad, que respondan a las necesidades y exigencias de las organizaciones, además de ser capaces de evaluar las nuevas tecnologías o comprender cómo pueden aplicarse de la mejor forma posible a la práctica del desarrollo del software.

El estudiante al finalizar esta formación debe ser capaz de:

- Identificar y establecer las fases y etapas que constituyen el desarrollo de un sistema de información.
- Identificar y modelar los requisitos del nuevo sistema a construir.

Se procura potenciar en los alumnos las capacidades para:

- “Aprender a aprender” y “aprender a pensar”.
- Desarrollar un espíritu crítico y una actitud abierta ante todo tipo de cambios que puedan afectar a la sociedad en la que vive y, en especial, a los cambios científico-técnicos de su especialidad.
- Fomentar actitudes y adquirir técnicas para un eficaz trabajo en equipo.
- Desarrollar actitudes de curiosidad intelectual y rigor científico.
- Basar en criterios deontológicos su futuro comportamiento en el ejercicio de la profesión.
- Estimular el perfeccionamiento profesional y la ampliación de estudios.

Para conseguir estos objetivos se propone un programa diseñado para proporcionar al estudiante un cuerpo de conocimiento – que incluya la cobertura de las actividades y

herramientas del proceso de desarrollo de proyectos, sus aspectos y los productos que elabora – y una experiencia en el desarrollo de un sistema software que les permita aplicar los conocimientos adquiridos en las clases teóricas.

5.1.2 Objetivos específicos

Los objetivos específicos de las asignaturas aquí presentadas se obtienen de los objetivos establecidos en la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca [García et al., 2000a], que se catalogan en tres apartados: *conceptos teóricos*, *aspectos prácticos* y *habilidades personales*.

Los objetivos de la unidad docente en el apartado de los conceptos teóricos son:

- **T1** Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
- **T2** Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.
- **T3** Importancia de los requisitos en el ciclo de vida del software.
- **T4** Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
- **T5** Especificaciones formales de requisitos.
- **T6** Método de análisis/diseño estructurado.
- **T7** Método de análisis/diseño orientado a objetos.
- **T8** Diseño de la interfaz gráfica de usuario.
- **T9** Estudio y comprensión de los fundamentos del diseño de sistemas software.
- **T10** Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.
- **T11** Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.
- **T12** Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.
- **T13** Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

Los objetivos de la unidad docente en el apartado de los aspectos prácticos son:

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de la Orientación a Objetos.

P3 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

P5 Programación orientada a objetos.

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

P7 Aprendizaje y manejo de forma práctica de plataformas, entornos de desarrollo, lenguajes de programación... de alta repercusión en el desarrollo de sistemas software en la actualidad.

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

P9 Construcción de sistemas software de entidad superior a una práctica de laboratorio, a ser posible partiendo de unas especificaciones reales obtenidas de *clientes* y/o *usuarios* reales.

Los objetivos de la unidad docente en el apartado de habilidades personales son:

H1 Mejora de la expresión oral.

H2 Mejora en la redacción de documentos técnicos.

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

H4 Capacitar a los alumnos para el trabajo en grupo.

La unidad docente de Ingeniería del Software y Orientación a Objetos se ha creado para dar cobertura a las titulaciones de Ingeniería Técnica en Informática de Sistemas (I.T.I.S) (Plan de 1997) e Ingeniero en Informática (I.I) (Plan de 1998) impartidas en la Universidad de Salamanca.

En estas titulaciones las asignaturas que mejor se ajustan a los objetivos marcados en la unidad docente se recogen en la Tabla 5.1. Dentro de estas asignaturas se encuentran las tres que son objeto de este Proyecto Docente, siendo el resto aquéllas con las que tienen una relación más estrecha.

Las dependencias e interrelaciones entre estas asignaturas se muestran en la Figura 5.1. En el establecimiento de estas dependencias se han tenido en cuenta el factor tiempo, que claramente establece el orden lógico en el que se van a cursar las asignaturas, como las aportaciones a la unidad docente de las mismas, en forma de contenidos y objetivos a conseguir.

Además de las asignaturas propias de la unidad docente, se incluye la asignatura de Diseño de Bases de Datos. Esto se debe a que es la asignatura en la que se introducen los modelos de datos conceptuales y lógicos (diagramas entidad/relación y modelos relacionales típicamente), lo que supone una importante base, a la vez que una descarga, para la asignatura de Ingeniería del Software donde estos modelos serán utilizados de forma práctica sin necesidad de tener que incluirlos en la parte teórica de la asignatura.

Los objetivos identificados se reparten en las asignaturas de la unidad docente como se indica en la Tabla 5.2, señalando de una manera especial aquéllos que son cometido de las asignaturas objeto de este Proyecto Docente.

Asignatura	Titulación	Curso	Carácter	Créditos
Interfaces Gráficas	I.T.I.S	2º	Optativa	6 (3T + 3P)
Ingeniería del Software	**I.T.I.S**	**3º**	**Obligatoria**	**6 (4,5T + 1,5P)**
Programación Orientada a Objetos	I.T.I.S	3º	Optativa	6 (3T + 3P)
Proyecto	I.T.I.S	3º	Obligatoria	9 (9P)
Análisis de Sistemas	**I.I**	**1º**	**Troncal**	**9 (6T + 3P)**
Administración de Proyectos Informáticos	**I.I**	**2º**	**Troncal**	**9 (6T + 3P)**
Sistemas de Información	I.I	2º	Troncal	9 (9P)
Proyecto	I.I	2º	Troncal	6 (6P)

Tabla 5.1. Asignaturas que componen la unidad docente

	Obj. Teóricos	Obj. Prácticos	Hab. Personales
Interfaces Gráficas	T8	P6	H1, H2, H3
Ingeniería del Software	**T1, T2, T3, T4, T6, T7, T9**	**P1, P2, P4**	**H1, H2, H3, H4**
Programación Orientada a Objetos	T7, T9	P2, P5	H1, H2, H3, H4
Proyecto I.T.I.S		P9	H1, H2, H3
Análisis de Sistemas	**T3, T4, T5, T7, T12, T13**	**P1, P2, P4**	**H1, H2, H3, H4**
Administración de Proyectos Informáticos	**T10, T11, T12**	**P4, P8**	**H1, H2, H3, H4**
Sistemas de Información		P7	H1, H2, H3
Proyecto I.I		P8, P9	H1, H2, H3

Tabla 5.2. Reparto de los requisitos en las asignaturas de la unidad docente

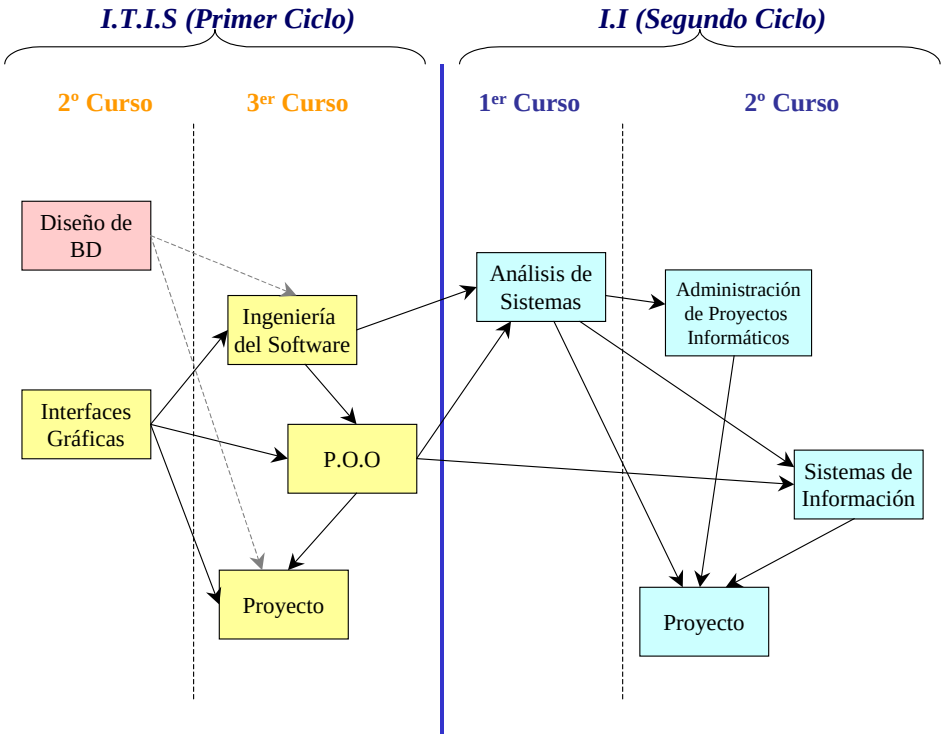


Figura 5.1. Dependencias entre las asignaturas que forman la unidad docente

5.2 Programa de la asignatura Ingeniería del Software

La asignatura *Ingeniería del Software* es la clave de la formación de los alumnos de la *Ingeniería Técnica en Informática de Sistemas* en materias de Ingeniería del Software.

Asignatura	Ingeniería del Software (obligatoria)
Créditos	4,5T + 1,5P
Estudios	I.T.I.S
Plan	B.O.E de 4-11-1997
Curso	3º
Cuatrimestre	1º
Responsable	Francisco José García Peñalvo (fgarcia@.usal.es)
Página web de la asignatura	http://tejo.usal.es/~fgarcia/docencia.html

Tabla 5.3. Ficha de la asignatura Ingeniería del Software

Como ya se ha comentado previamente en este Proyecto Docente, esta asignatura se imparte en el quinto cuatrimestre (o lo que es lo mismo, en el primer cuatrimestre del tercer y último curso de la titulación Ingeniería Técnica en Informática de Sistemas) dentro del Plan de Estudios del B.O.E de 4 de noviembre de 1997 [BOE, 1997].

Esta asignatura está dotada de 6 créditos (**4,5 teóricos** y **1,5 prácticos**), lo que conduce al principal problema de la misma: *un número escaso de créditos para la cantidad de materia que puede tratarse bajo este epígrafe*. Este problema se aborda minimizando los contenidos que se solapan con otras asignaturas de la titulación, descargando temas *avanzados* en las asignaturas afines del segundo ciclo y enfocando la asignatura optativa de *Programación Orientada a Objetos* como una continuación de la asignatura de *Ingeniería del Software*, aunque centrada en los aspectos de diseño e implementación dentro del paradigma objetual.

En la elaboración del programa se ha optado por una estrategia cuyos ejes básicos son:

1. Mantener los temas fundamentales que tradicionalmente se imparten en este tipo de asignaturas en otras Universidades, tanto españolas como extranjeras, así como los derivados de las recomendaciones de los currículos internacionales y los diferentes cuerpos de conocimiento de la Ingeniería del Software.
2. Se parte de los siguientes prerrequisitos:
    a. El alumno debe estar familiarizado con la teoría y la práctica del diseño y codificación en lenguajes procedurales (por ejemplo C [Kernighan y Ritchie, 1988]). Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos**.
    b. El alumno debe tener los conocimientos y la práctica necesaria en la creación de modelos de información conceptuales y lógicos, en concreto, dominio del modelo entidad-relación, paso al modelo relacional y normalización. Estos conceptos se adquieren en la asignatura de segundo curso **Diseño de Bases de Datos** (asignatura troncal en el Plan de Estudios vigente).
    c. Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura optativa **Interfaces Gráficas** del segundo curso.
3. No dejar que en ningún caso el alumno pierda de vista que el desarrollo de todo sistema software debe abordarse con un proceso sistemático que englobe los procedimientos, las técnicas y las herramientas necesarias.

4. Hacer hueco a las, cada día más demandadas, técnicas de desarrollo orientadas a objetos, dentro de un currículo dominado por el paradigma estructurado.
5. Dentro de los procesos del ciclo de vida, esta asignatura se decanta por los procesos de desarrollo y, dentro de éstos, por las actividades de análisis de requisitos y diseño.
6. Aunque la asignatura se fundamenta en la transmisión de conocimiento técnico, no se puede (ni se quiere) perder la oportunidad de hacer que los alumnos desarrollen y potencien otras habilidades más generales, pero de importancia capital en su futuro como profesionales: *acostumbrarse a consultar bibliografía (especialmente en inglés), haciendo hincapié en la importancia que tiene leer con cuidado para sintetizar, escribir y modelar de forma adecuada* [Jackson, 1995]; *escribir documentos técnicos que describan los diferentes elementos software que se crean a lo largo de un proyecto* [Deveaux et al., 1999] *cuidando los estándares de documentación* [Gersting, 1994; McCauley et al., 1996], *sin que ello signifique dar la espalda a las reglas gramaticales y de estilo que ofrece un lenguaje tan rico como el castellano* [Vaquero, 1999]; *desarrollar una capacidad de comunicación oral adecuada* [McDonald y McDonald, 1993; Fell et al., 1996].
7. Llegar a un equilibrio entre la teoría y la práctica [Glass, 1996], de forma que una base teórica bien establecida sea el fundamento adecuado para la aplicación práctica de la Ingeniería del Software. Pero en ningún caso permitir que la primera aproximación a la Ingeniería del Software carezca de una de estas dos partes [Stevens, 2001].

5.2.1 Programa de la parte teórica

La parte teórica de la asignatura *Ingeniería del Software* está orientada a satisfacer aquellos objetivos teóricos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**T1, T2, T3, T4, T6, T7** y **T9**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a] (incluido en el Apéndice A de este Proyecto Docente e Investigador).

T1	Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
T2	Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.
T3	Importancia de los requisitos en el ciclo de vida del software.
T4	Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
T6	Método de análisis/diseño estructurado.
T7	Método de análisis/diseño orientado a objetos.
T9	Estudio y comprensión de los fundamentos del diseño de sistemas software.

Tabla 5.4. Objetivos del programa de teoría de la asignatura Ingeniería del Software

5.2.1.1 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de nueve temas, organizados en cuatro unidades docentes, como se muestra en la Tabla 5.5.

Presentación de la asignatura (1 Hora)
Unidad Didáctica I: Conceptos básicos (6 Horas)
Tema 1. Introducción a la Ingeniería del Software (6 Horas)
Unidad Didáctica II: Paradigma estructurado de desarrollo (23 Horas)
Tema 2. Análisis y especificación de requisitos (11 Horas)
Tema 3. Análisis estructurado (2 Horas)
Tema 4. Diseño del software (6 Horas)
Tema 5. Diseño estructurado (4 Horas)
Unidad Didáctica III: Introducción al paradigma objetual (13 Horas)
Tema 6. Introducción a la Orientación a Objetos (6 Horas)
Tema 7. UML (6 Horas)
Tema 8. Visión general de la metodología OMT (1 Hora)
Unidad Didáctica IV: Miscelánea (2 Horas)
Tema 9. Herramientas CASE (2 Horas)

Tabla 5.5. Estructura del programa de teoría de la asignatura Ingeniería del Software (4,5 créditos)

La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve exposición de lo que es la Ingeniería del Software y la visión que se pretende conseguir desde la asignatura. Para que esto sea efectivo, los alumnos deben contar con el programa de la asignatura, resumido en la guía académica que se les entrega con la matrícula [GAFC-USAL,

2001], y totalmente actualizado en la página web de la asignatura (http://tejo.usal.es/~fgarcia/docencia.html).

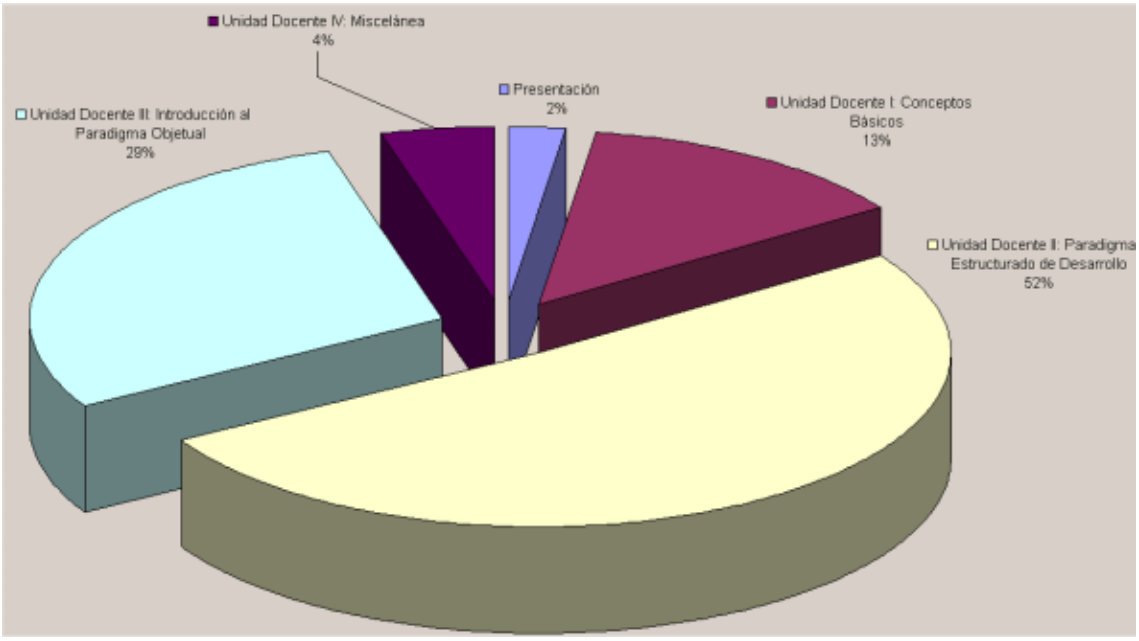


Figura 5.2. Reparto porcentual de las horas de teoría de Ingeniería del Software entre las unidades didácticas

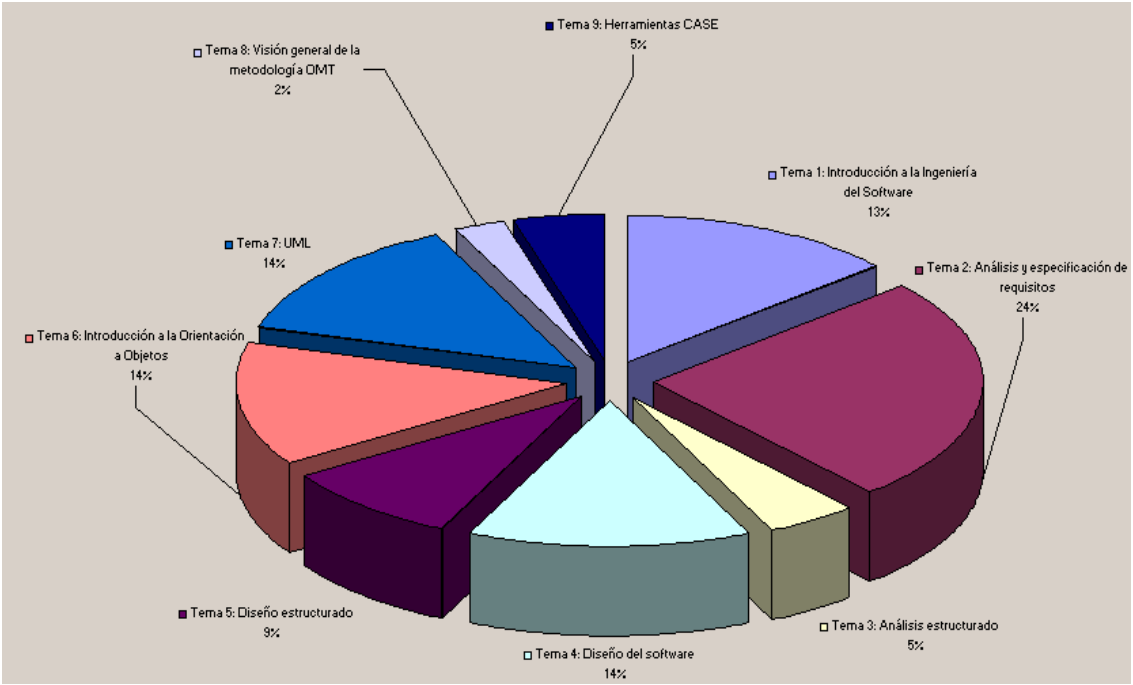


Figura 5.3. Reparto de las horas de teoría de Ingeniería del Software entre los temas

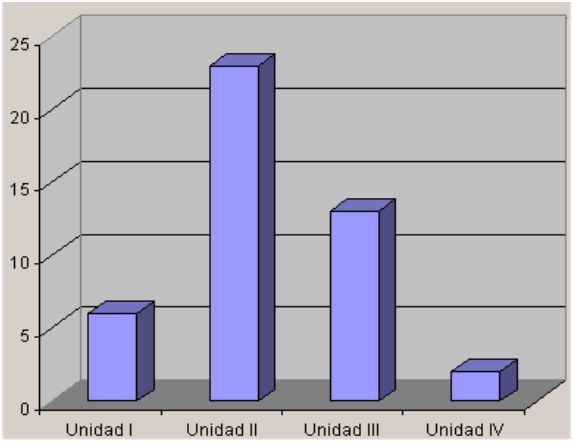


Figura 5.4. Distribución temporal por unidades didácticas

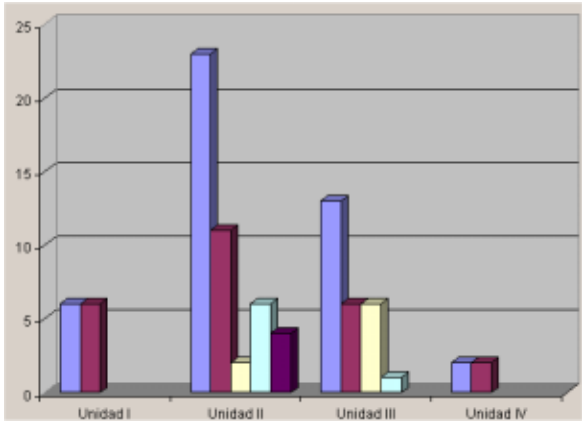
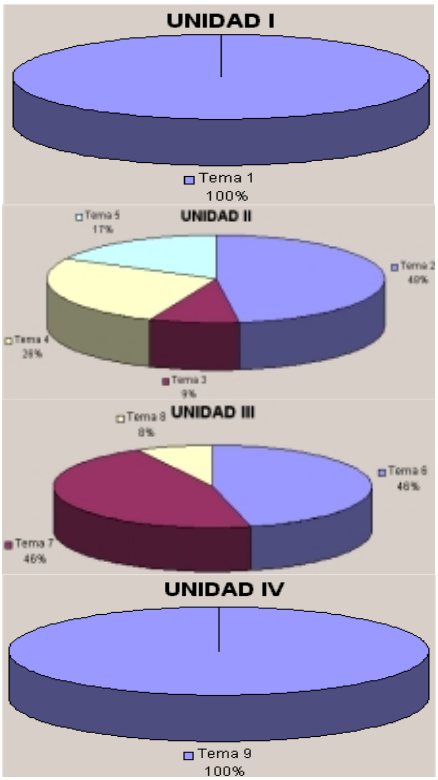


Figura 5.5. Distribución temporal por unidades didácticas y temas de cada una de ellas

En la Figura 5.2 se presenta el reparto porcentual de las unidades didácticas de la asignatura *Ingeniería del Software*, mientras que la Figura 5.3 refleja el reparto porcentual de los diferentes temas. La Figura 5.4 refleja cuantitativamente el número de horas dedicadas a cada una de las unidades didácticas y la Figura 5.5 presenta la distribución en horas de los temas de cada unidad didáctica, comparándolo con la duración total de la unidad didáctica a la que pertenece (primera barra de cada unidad). A continuación se presenta porcentualmente la presencia de cada tema dentro de su unidad didáctica.



Tema 1. Introducción a la Ingeniería del Software (6 H)

Tema 2. Análisis y especificación de requisitos (11 H)
Tema 3. Análisis estructurado (2 H)
Tema 4. Diseño del software (6 H)
Tema 5. Diseño estructurado (4 H)

Tema 6. Introducción a la Orientación a Objetos (6 H)
Tema 7. UML (6 H)
Tema 8. Visión general de la metodología OMT (1 H)

Tema 9. Herramientas CASE (2 H)

En la Tabla 5.6 se presenta la correspondencia existente entre el programa de teoría y los objetivos teóricos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T1, T2, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T9, H3
Tema 5	T6, H3
Tema 6	T3, T7, H3
Tema 7	T7, H3
Tema 8	T7, H3
Tema 9	P4, H3

Tabla 5.6. Correspondencia entre el temario teórico y los objetivos teóricos de la asignatura

La **Unidad Didáctica IV**: ***Miscelánea***, es difícil de impartir en la realidad por limitaciones de tiempo. El **Tema 9**, dedicado a las herramientas CASE, es factible introducirlo a la vez que se realiza la **Práctica 4**. Otros posibles temas candidatos a ser tratados en esta unidad miscelánea, podrían ser: *pruebas del software, mantenimiento, reingeniería, calidad del software, gestión de proyectos...*

Estos y otros temas pueden ser objeto de actividades complementarias, que se llevarán a cabo dependiendo de diversos factores, entre los que cabe citar *la disponibilidad de tiempo para hacerlas efectivas* y *el interés y la colaboración de los propios alumnos*. Las actividades complementarias a realizar pueden caer dentro de alguno de los siguientes grupos:

- Seminarios impartidos sobre temas específicos.
- Trabajos voluntarios realizados por los alumnos.
- Conferencias invitadas.
- Talleres de trabajos realizados por los alumnos sobre temas de Ingeniería del Software.

Desarrollo de las clases de teoría

Las clases de teoría se llevan a cabo utilizando una variante de la clase magistral, donde el profesor se apoya en un retroproyector y en la pizarra para el desarrollo de los nueve temas que componen el temario.

Los alumnos cuentan de antemano con las transparencias de los temas, no teniendo que tomar apuntes en el sentido clásico del término, pudiendo prestar atención a las explicaciones, completando las transparencias con las notas que cada uno crea oportuno. Además, permite que el alumno que lo desee intervenga en cualquier momento para hacer una pregunta o solventar una duda y no, como en el dictado de apuntes, para pedir que se repita una frase.

El uso que de las transparencias hace el que suscribe este Proyecto Docente, es el de guión de clase. De forma, que recojan los puntos fundamentales que se van a desarrollar, más esquemas, gráficos y definiciones de términos. Esto obliga a que la clase no se convierta en una mera lectura de las transparencias, y se asemeje más a una conferencia centrada en el tema abordado en cada clase.

Esta forma de abordar las clases teóricas tiene la ventaja de que obliga al alumno interesado por la asignatura a prestar atención a las explicaciones y a completar el material distribuido por el profesor con la bibliografía de consulta básica o las lecturas complementarias que se recomiendan a lo largo de cada uno de los temas (por este motivo, aunque hay unos apuntes desarrollados por el profesor [García, 1999], éstos no le son facilitados a los alumnos).

La elección de esta forma de impartir clases viene *forzada* por dos factores fundamentales:

- *La infraestructura con la que se cuenta*. Una clase con capacidad para recoger los alumnos matriculados, cuyo número ha rondado los dos últimos cursos la cifra de 140 alumnos.
- *La gran cantidad de conceptos que se han de impartir*. Es la primera asignatura que, en su Plan de Estudios, afronta el desarrollo de software desde una perspectiva global a través de su ciclo de vida, y no centrada en los algoritmos, las bases de datos o las redes por ejemplo.

Las ventajas que se obtienen con este método docente se pueden resumir en los siguientes puntos:

- Permite al docente controlar el *tiempo de la clase*, siendo lo suficientemente flexible para que el profesor pueda sortear los posibles imprevistos en la planificación del calendario, haciendo hincapié en los conceptos más importantes y pasando más someramente por los temas más sencillos o de menor trascendencia.
- Requiere una atención constante del alumno, para seguir las explicaciones del profesor.
- Obliga al alumno a consultar la bibliografía y las lecturas recomendadas para completar el material facilitado por el profesor, orientado a servir de base para las clases, pero a todas luces insuficiente para preparar la asignatura.

Como inconvenientes cabe citar:

- Obliga al profesor a realizar diferentes cambios de ritmo, de tono... para no caer en la monotonía del pasar transparencias.

- Si el alumno *desconecta* de las explicaciones no sacará provecho de su asistencia a clase, lo que es muy factible por el relax que supone no tener que copiar apuntes.
- Si el alumno confunde las transparencias facilitadas con *toda la verdad sobre la asignatura,* estará en desventaja para preparar y comprender la asignatura.

Evaluación de la parte teórica

La forma principal de evaluar la parte teórica de esta asignatura es mediante la realización de una prueba escrita. Aunque también se puede aumentar la nota final realizando trabajos teóricos o participando en seminarios (aunque no incida esta nota para aprobar un examen suspenso, ya sea de teoría o de prácticas).

En la Figura 5.6 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota del examen de teoría y los trabajos voluntarios en dicha nota.

Si (***Teoría*** ≥ 4,75) **y** (Práctica ≥ 5.0)
    Nota Final = (***Teoría****0,5) + (Práctica*0,5) + ***Nota trabajos***
Si no
    Ø
Fin si

Figura 5.6. Influencia de la nota en la parte teórica en la nota final de Ingeniería del Software

A la hora de elaborar el examen de la asignatura se tienen en cuenta los siguientes aspectos:

- El examen se divide en dos partes que hay que aprobar por separado para superar la prueba. La primera parte se centra en la evaluación de conceptos desarrollados en la asignatura, mientras que la segunda es un conjunto de supuestos prácticos, en la que aparecen algunas de las técnicas de modelado estudiadas.
- La primera parte del examen está compuesta por un conjunto de cuestiones de respuesta abierta y corta, donde se prima evaluar la capacidad de asimilación, de comprensión y de relacionar los conceptos desarrollados en la asignatura, sobre la capacidad memorística del alumno.
- Los supuestos de la segunda parte buscan comprobar que los modelos realizados en la práctica obligatoria han sido asimilados por todos los integrantes del grupo, así como evaluar el dominio de otras técnicas de modelado menos utilizadas en las prácticas.

Bibliografía básica de referencia

Desde esta asignatura, así como desde las otras que forman este Proyecto Docente e Investigador, se entiende que el conocimiento es tanto el que se posee como el que se sabe conseguir. Por este motivo, se fomenta el manejo de libros y otros recursos bibliográficos.

Así, junto con el programa de la asignatura se le facilita a los alumnos una lista con la bibliografía básica que pueden (y deben) consultar para preparar y dominar la asignatura de Ingeniería del Software.

A la hora de elaborar esta lista se tienen en cuenta los siguientes factores:

- No se trata de distribuir un listado exhaustivo de todos los títulos que el profesor conoce o maneja para preparar sus clases, sino una lista representativa que trate todo el temario en global (para los aspectos concretos ya se recomendarán lecturas complementarias en cada tema).
- En la elección de los títulos influye su disponibilidad para los alumnos, bien en la biblioteca o a través el profesor.
- El idioma, de forma que si un libro está traducido al español prevalecerá sobre el original. Esto es debido a que, aunque los alumnos deben acostumbrarse al manejo de bibliografía en inglés, sienten una aversión generalizada por los textos que no están español.

La lista de títulos que se les propone como bibliografía básica de consulta es:

- **Booch, G.** "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996.
- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.
- **Durán, A., Bernárdez, B.** "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.
- **OMG.** "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.

- **Piattini, M. G., Daryanani, S. N.** "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.
- **Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.
- **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.
- **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.
- **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.
- **Sommerville, I.** "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002.
- **Yourdon, E., Constantine, L. L.** "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993.

5.2.1.2 Desarrollo comentado del programa de teoría

En este epígrafe se va a desarrollar con mayor grado de detalle el programa de la parte teórica de la asignatura Ingeniería del Software. Este programa se ha dividido en cuatro partes o unidades didácticas.

Una primera que introduce la terminología básica que se maneja en la asignatura, en especial el concepto de ciclo de vida.

Una segunda donde se desarrollan las características del paradigma de desarrollo estructurado, en concreto de las fases de análisis y diseño estructurado.

Una tercera, en la que se realiza una introducción a la orientación a objetos, centrada en el modelo de objetos, en UML y en una sucinta presentación de la metodología OMT, génesis de otras avanzadas.

Por último, una cuarta donde, dependiendo del tiempo de que se disponga, se tratan brevemente otros temas relacionados con la Ingeniería del Software.

Es obvio que, debido a los créditos asignados a la asignatura, no es posible extenderse en todas las partes por igual. En algunos casos la explicación se verá reducida a una breve referencia o comentario, mientras que en otros, que se consideran más representativos y de mayor actualidad, se profundiza en sus explicaciones.

Cada parte esta dividida en una serie de temas, subtemas y epígrafes. Cada uno de los temas va a ser descrito siguiendo el siguiente patrón de documentación:

- **Título:** Indica de forma concisa la denominación comúnmente asignada al tema correspondiente.
- **Descriptores:** Consiste en la enumeración de una serie de palabras clave, que ayudan a conocer el contenido de los temas tratados. Representan el papel de *índices* que facilitan las búsquedas de los conceptos tratados en cada uno de los temas, o en varios temas.
- **Objetivos:** Indica los objetivos que se buscan con la inclusión en el programa de cada uno de los temas.
- **Contenido:** Se señalan los subtemas y eventualmente epígrafes que se tratan dentro del tema considerado.
- **Resumen:** Elabora un resumen explicativo, a modo de epítome, de los diferentes conceptos que se desarrollan en la unidad temática.
- **Bibliografía:** Se citan para cada uno de los temas las referencias bibliográficas, clasificadas en cada caso en dos categorías, a saber:
    - *Referencias básicas para el tema tratado.* Se trata de las citas básicas donde mejor se tratan los epígrafes propios del tema.
    - *Referencias complementarias al tema tratado.* En este caso se trata de citas bibliográficas que bien puede utilizar el profesor para profundizar en alguno de los epígrafes tratados, o bien pueden aconsejarse a los alumnos para ilustrar algún aspecto del tema o para complementar el tema tratado.

En la Tabla 5.7 se detalla la estructura de la parte teórica de la asignatura Ingeniería del Software.

Unidad Didáctica I: Conceptos básicos (6 Horas)

Tema 1. Introducción a la Ingeniería del Software (6 Horas)

1.1 Conceptos generales (10 Minutos)

1.2 Sistemas de información (20 Minutos)

1.2.1 Definición

1.2.2 Objetivos de un sistema de información

1.2.3 Elementos de un sistema de información

1.2.4 Estructura de un sistema de información

1.3 Ingeniería del Software (20 Minutos)

1.3.1 Definición

1.3.2 Elementos de la Ingeniería del Software

1.3.3 Objetivos de la Ingeniería del Software

1.4 Ciclo de vida del software (1 Hora)

1.4.1 Definición

1.4.2 Objetivos del ciclo de vida

1.4.3 Descripción general del ciclo de vida del software

1.4.4 Curva del ciclo de vida del proyecto

1.5 Paradigmas de la Ingeniería del Software (3 Horas)

1.5.1 Modelo primitivo

1.5.2 Modelo barroco

1.5.3 Modelo clásico

1.5.4 Modelo estructurado

1.5.5 Modelo de prototipos

1.5.6 Modelo evolutivo

1.5.7 Modelo incremental

1.5.8 Modelo espiral

1.5.9 Estándar ISO 12207

1.5.10 Ciclo de vida para desarrollos OO

1.6 Metodologías (1 Hora)

1.6.1 Introducción

1.6.2 Definición

1.6.3 ¿Qué cubren las metodologías?

1.6.4 Objetivos de las metodologías

1.6.5 Características deseables en una metodología

1.6.6 Metodología versus método

1.6.7 Clasificación

Unidad Didáctica II: Paradigma estructurado de desarrollo (23 Horas)

Tema 2. Análisis y especificación de requisitos (11 Horas)

2.1 Introducción al análisis (1 Hora)

2.1.1 Generalidades

2.1.2 Definición de análisis

2.1.3 Requisitos

2.1.4 Objetivo

2.1.5 Tareas

2.1.6 Los principios del análisis

2.2 Especificación de requisitos del software (30 Minutos)

2.2.1 Definición

2.2.2 Características de una E.R.S

2.2.3 Estructura de una E.R.S

2.3 Técnicas de especificación (30 Minutos)

2.3.1 Generalidades

2.3.2 Técnicas gráficas

2.3.3 Técnicas textuales

2.3.4 Marcos o plantillas

2.3.5 Enfoque de modelado

2.4 Modelado funcional (7 Horas)

2.4.1 Introducción

2.4.2 DFD

2.4.3 DFDs nivelados

2.4.4 Diccionario de datos

2.4.5 Especificación de procesos

2.5 Modelado de la información (10 Minutos)

2.5.1 Introducción

2.5.2 Diagrama de estructura de datos

2.6 Modelado del comportamiento (80 Minutos)

2.6.1 Introducción

2.6.2 Lista de eventos

2.6.3 Diagrama de transición de estados

2.7 Balanceo de modelos (30 Minutos)

2.7.1 Balanceo de modelos

2.7.2 Técnicas matriciales

Tema 3. Análisis estructurado (2 Horas)

3.1 Introducción (10 Minutos)

3.2 Enfoque de modelado clásico (20 Minutos)

3.2.1 Los cuatro modelos del sistema

3.2.2 Los problemas del enfoque clásico

3.3 Enfoque de modelado de Yourdon (90 Minutos)

3.3.1 Introducción

3.3.2 Modelo esencial

3.3.3 Modelo de implantación

Tema 4. Diseño del software (6 Horas)

4.1 Introducción (15 Minutos)

4.1.1 Importancia del diseño en el ciclo de vida de un producto

4.1.2 Diseño dentro de la Ingeniería del Software

4.1.3 Evolución del diseño

4.2 Proceso de diseño (20 Minutos)

4.2.1 Introducción

4.2.2 Definición de diseño

4.2.3 Presentación de las actividades del diseño

4.3 Actividades de diseño (40 Minutos)

4.3.1 Diseño arquitectónico

4.3.2 Diseño de los datos

4.3.3 Diseño de procedimientos

4.3.4 Diseño de la interfaz

4.4 Fundamentos de diseño (2 Horas)

4.4.1 Introducción

4.4.2 Abstracción

4.4.3 Refinamiento sucesivo
4.4.4 Modularidad
4.4.5 Arquitectura del software
4.4.6 Estructura del programa
4.4.7 Partición estructural
4.4.8 Estructura de datos
4.4.9 Procedimiento del software
4.4.10 Ocultación de la información

4.5 Diseño modular (2 Horas y 45 Minutos)

4.5.1 Introducción
4.5.2 Definición de módulo
4.5.3 Tamaño de los módulos
4.5.4 Criterios y reglas para la evaluación de la modularidad
4.5.5 Complejidad de los módulos
4.5.6 Independencia modular

Tema 5. Diseño estructurado (4 Horas)

5.1 Introducción (10 Minutos)
5.2 Diagrama de estructuras (50 Minutos)
5.3 Estrategias de diseño (3 Horas)

5.3.1 Análisis de transformación
5.3.2 Análisis de transacción

Unidad Didáctica III: Introducción al paradigma objetual (13 Horas)

Tema 6. Introducción a la Orientación a Objetos (6 Horas)

6.1 Introducción y evolución de la Orientación a Objetos (1H)

6.1.1 Situación actual
6.1.2 Áreas de aplicación
6.1.3 Evolución de la Orientación a Objetos
6.1.4 Ventajas y desventajas
6.1.5 Orientación a Objetos versus reutilización del software

6.2 Modelo objeto (4 Horas y 30 Minutos)

6.2.1 Tipos de paradigmas de programación
6.2.2 Definición de modelo objeto
6.2.3 Concepto de objeto
6.2.4 Concepto de clase
6.2.5 Abstracción
6.2.6 Encapsulamiento
6.2.7 Modularidad
6.2.8 Jerarquía
6.2.9 Mensajes
6.2.10 Tipos
6.2.11 Polimorfismo
6.2.12 Enlaces y asociaciones
6.2.13 Relaciones entre clases

6.3 Análisis y diseño orientados a objetos (30 Minutos)

6.3.1 Introducción

6.3.2 Análisis orientado a objetos

6.3.3 Diseño orientado a objetos

Tema 7. UML (6 Horas)

7.1 Introducción (30 Minutos)

7.1.1 ¿Por qué modelar?

7.1.2 Concepto de lenguajes de modelado

7.1.3 Modelos en el paradigma objetual

7.1.4 Necesidad de un estándar

7.1.5 ¿Qué es UML?

7.1.6 Lo que UML no es

7.1.7 Origen y evolución de UML

7.2 Una visión general de UML (30 Minutos)

7.2.1 Consideraciones generales

7.2.2 Las vistas

7.2.3 Los diagramas

7.2.4 Elementos de modelado

7.2.5 Mecanismos generales

7.3 Diagramas de estructura (2 Horas)

7.3.1 Definición

7.3.2 Diagramas de clase

7.4 Diagramas de interacción (1 Hora)

7.4.1 Definición

7.4.2 Diagramas de secuencia

7.4.3 Diagramas de colaboración

7.5 Casos de uso (2 Horas)

7.5.1 Introducción

7.5.2 Definición

7.5.3 Notación en UML

7.5.4 Relaciones entre casos de uso

7.5.5 Construcción de los casos de uso

7.5.6 Descripción de los casos de uso

7.5.7 Transición hacia los objetos

Tema 8. Visión general de la metodología OMT (1 Hora)

8.1 Introducción

8.2 Fases

8.3 Análisis

8.3.1 Definición del problema

8.3.2 Modelo de objetos

8.3.3 Modelo dinámico

8.3.4 Modelo funcional

8.4 Diseño del sistema

8.5 Diseño de los objetos

Unidad Didáctica IV: Miscelánea (2 Horas)
Tema 9. Herramientas CASE (2 Horas)
9.1 Introducción (30 Minutos)
9.2 Componentes de una herramienta CASE (10 Minutos)
9.3 Clasificación de las herramientas CASE (20 Minutos)
9.4 Integración de CASE (1 Hora)

Tabla 5.7. Estructura detallada del programa de teoría de Ingeniería del Software

Unidad Didáctica I: Conceptos Básicos

Objetivo genérico

Como su propio nombre indica, esta parte constituye una introducción necesaria para situar al alumno en el ambiente en el que se va a desarrollar su actividad.

Por otra parte, el discente debe tomar conciencia de que su trabajo profesional ha de ejecutarse con rigor y, por tanto, ha de estar sujeto al empleo de un método de Ingeniería adaptado, en concreto, a la Ingeniería del Software.

Se pretende que el estudiante comprenda que está obligado a integrarse en proyectos interdisciplinares, con el objetivo y responsabilidad de sistematizarlos, para su coordinación en un trabajo, o conjunto de trabajos concretos que se realizarán con la asistencia de computadores. Así mismo, debe ser consciente que su labor implica la comunicación con diversas personas, cada una de las cuales jugará un rol determinado en el sistema de información en que se esté trabajando, siendo muchas veces necesario adaptarse a su visión del problema para que esa comunicación sea efectiva y redunde en una correcta solución del mismo.

El desarrollo de sistemas y aplicaciones, y la promoción en las organizaciones del uso masivo de las Tecnologías de la Información, suponen hoy un compromiso de eficiencia. Cada vez se exige una mayor interactividad entre las posiciones del organigrama en sus aspectos funcionales, para mejorar la cooperación y la integración de datos. Todo ello conlleva un aumento de la complejidad, que debe reducirse con el empleo de estándares de Ingeniería que puedan ser considerados como patrones de diferentes niveles y que aseguren el desarrollo de proyectos con la calidad, idoneidad, viabilidad y economía precisa.

Esta unidad supone el 13% de la asignatura, estando compuesta por un único tema, **Introducción a la Ingeniería del Software**, que se detalla a continuación.

Tema 1: ***Introducción a la Ingeniería del Software***

Descriptores

Software; aplicaciones del software; crisis del software; sistemas de información; tecnologías de información; ciclo de vida; paradigmas de la Ingeniería del Software; proceso software; metodología; método.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T1** y **T2** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
- Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir a los alumnos en el planteamiento del ejercicio de la actividad de ingeniería.
- Establecer las características de los sistemas software.
- Establecer los conceptos de ciclo de vida del desarrollo del software, bajo los distintos planteamientos que se han ido desarrollando.
- Destacar la necesidad del empleo de metodología en el desarrollo del software, como única forma de desarrollar software profesional y de calidad.
- Establecer el desarrollo de software como un proceso interdisciplinar, que implica la colaboración del usuario, y que se ve favorecido por el conocimiento que tenga el ingeniero del software del dominio del problema.

Contenidos

1.1 Software
1.2 Sistemas de información
1.3 Ingeniería del Software
1.4 Ciclo de vida del software
1.5 Paradigmas de la Ingeniería del Software
1.6 Metodologías

Tabla 5.8. Contenidos del tema 1 del temario teórico de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE4 Software processes** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Engineering Process** [El Emam, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

En la primera parte se introducen los términos y conceptos que se manejan en el resto de temas, a la par que se comienza una reconducción del alumno, mediatizado por un culto excesivo a la tecnología, para que termine considerando que la tecnología no es un fin en sí misma, sino un medio para la construcción de los sistemas de información de las organizaciones.

En una segunda parte del tema, se estudia el ciclo de vida del software y del desarrollo del software, diferenciando, en una primera aproximación, las tres fases generales que aparecen en la construcción de todo producto software: *definición*, *desarrollo* y *mantenimiento*. Posteriormente, se estudian las características de los paradigmas de la Ingeniería del Software (también denominados modelos de ciclo de vida) más representativos.

La exposición que de los diferentes modelos de ciclo de vida se hace, conduce al alumno desde los modelos secuenciales, totalmente desaconsejables, hasta los modelos evolutivos e incrementales, más acordes con las herramientas de desarrollo actuales y con la orientación a objetos. Como final de esta parte se presenta *el estándar ISO 12207* [ISO/IEC, 1995], como el marco estándar donde encuadrar los diferentes procesos que aglutinan las actividades del ciclo de vida del software, y se hace una breve reseña a los *modelos de ciclo de vida en los desarrollos orientados a objetos*, presentado como ejemplo concreto el modelo fuente [Henderson-Sellers y Edwards, 1990].

La tercera y última parte del tema está dedicada a la introducción de las metodologías de desarrollo, como contraposición al desarrollo anárquico y artesanal de aplicaciones, tan relacionado con la tan nombrada *crisis del software*.

Al clasificar las metodologías se hace una comparativa entre las características de aquéllas centradas en el paradigma estructurado frente a las metodologías orientadas al objeto.

Bibliografía básica

Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulos 3 y 4).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 1, y 2).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 3 y 8).

Yourdon, E. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993. (Capítulo 5).

Bibliografía complementaria

"*Anecdotes/stories about Software Engineering*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/anecdote. [Última vez visitado, 28-1-2000]. July 1993.

"*Computer Horror Stories*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/horror. [Última vez visitado, 28-1-2000]. July 1993.

"*Origin of Term 'Software Engineering'*". http://www.cs.queensu.ca/FAQs/comp.software-eng/archive/SEorigin. [Última vez visitado, 28-1-2000]. July 1993.

"*Software Engineering Questions and Answers*". http://www.qucis.queensu.ca/FAQs/comp.software-eng/questions.html. [Última vez visitado, 28-1-2000]. 1998.

Amescua Seco, A. de, García Sánchez, L., Martínez Fernández, P. y Díaz Pérez, P. "*Ingeniería del Software de Gestión. Análisis y Diseño de Aplicaciones*". Paraninfo, 1995.

Andreu, R., Ricart, J., Valor, J. "*Estrategia y Sistemas de Información*". 2ª Ed. McGraw-Hill (*serie de management*), 1996.

Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

Baber, R. L. "*Comparison of Electrical 'Engineering' of Heaviside's Times and Software 'Engineering' of our Times*". IEEE Annals of the History of Computing, 19(4):5-17. 1997.

Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, p. 20, 1992.

Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". ACM Software Engineering Notes, 11(4):22-42. 1986.

Boehm, B. W. "*Software Engineering Economics*". Prentice Hall, 1981.

Boehm, B. W., Bose, P. "*A Collaborative Spiral Software Process Model Based on Theory W*". In Proceedings of the Int'l Conf. Software Process. IEEE Computer Society Press. Pages, 59-68. 1994.

Boehm, B. W., Egyed, A., Kwan, J., Port, D., Shah, A., Madachy, R. "*Using the WinWin Spiral Model: A Case Study*". IEEE Computer, 31(7):33-44, July 1998.

Brereton, P., Budgen, D., Bennet, K., Munro, M., Layzell, P., Macaulay, L., Griffiths, D., Stannett, C. "*The Future of Software*". Communications of the ACM, 42(12):78-84. December 1999.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

CERN. "*STING Software Engineering Glossary*". http://dxsting.cern.ch/sting/glossary.html. April 1997.

Conger, S. "*The New Software Engineering*". Course Technology, 1994.

Chandra, J., March, S. T., Mukherjee, S., Pape, W., Ramesh, R., Rao, H. R., Waddoups, R. O. "*Information Systems Frontiers*". Communications of the ACM, 43(1):71-79. January 2000.

Davis, A. M. "*Fifteen Principles of Software Engineering*". IEEE Software, 11(6):94-96,101, November/December 1994.

Dedene, G., Snoeck, M. "*MERODE: A Model-Driven Entity-Relationship Object-Oriented Development Method*". ACM Software Engineering Notes, 19(3):51-61. July 1994.

DoD. "*SRI Reuse Glossary*". http://sw-eng.falls-church.va.us/ReuseIC/policy/glossary/glossary.htm, December 1995.

Emery, J. C. "*Sistemas de Información para la Dirección. El Recurso Estratégico y Crítico*". Ed. Díaz de Santos, 1990.

Fayad, M. E., Laitinen, M., Ward, R. P. "*Software Engineering in the Small*". Communications of the ACM, 43(3):115-118. March 2000.

Frakes, W. B., Fox, C., Nejmeh, B. A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

Gaitero Gordillo, D. "*Metodología Métrica. Un Enfoque Práctico*". Everest Multimedia, 1997.

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International, 1991.

Glass, R. L. "*The Software Criris… Not?*". IEEE Computer, 27(4):104. April 1994.

Gray, L. "*ISO/IEC 12207 Software Lifecycle Processes*". Crosstalk. The Journal of Defense Software Engineering, 9(8). August 1996.

Gutiérrez, I., Medinilla, N. "*Contra el Arraigo de la Cascada*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. P Botella, J. Hernández y F. Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999.

Henderson-Sellers, B. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Henderson-Sellers, B., Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990.

Horan, P. "*Software Engineering - A Field Guide***".** Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science*, Van Nostrand Reinhold, p. 1218. 1993.

Jacobson, I. "*Building Without Blueprints*". Object Magazine, 7(9): 71-72. November 1997.

Kruchten, P. "*A Rational Development Process*". Crosstalk, 9(7):11-16. July 1996.

Leaney, J. "*Software Engineering - An Introductory Tutorial*". University of Technology, Sydney. http://www.ee.uts.edu.au/~jrleaney/setut.htm. October 1995.

Lions, J. L. "*ARIANE 5 Flight 501 Failure*". Report by the Inquiry Board. http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html. [Última vez visitado, 28-1-2000]. París, 19 Julio 1996.

Lucero, J. L. "*Gestión de la Calidad del Software*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Abril 1996.

Lucero, J. L., Ramos, M. Á. "*Gestión de Proyectos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Enero-Febrero 1996.

Lucero, J. L., Ramos, M Á. "*Dirección de Departamentos Informáticos*". Notas del curso impartido en la Demarcación de ALI C-L en Valladolid por IEE. Febrero-Marzo 1996.

Monforte, M. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

Moore, J. "*ISO 12207 and Related Software Life-Cycle Standards*". http://www.acm.org/tcs/lifecycle.html. [Última vez visitado, 12-3-2000]. 1996.

Muller, P.-A. "*Modelado de Objetos con UML*". Ediciones Gestión 2000, 1997.

National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Piattini Velthuis, M. G. "*Ciclos de vida para Sistemas Orientados a Objetos*". Cuore, (7):6-11. Septiembre, 1995.

Raccoon, L. B. S. "*Fifty Years of Progress in Software Engineering*". ACM Software Engineering Notes, 22(1):88-104. January 1997.

Rumbaugh, J. "*What Is a Method*". JOOP. October 1995.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

Scacchi, W. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). October 1987.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Singh, R. "*The Software Life Cycle Processes Standard*". IEEE Computer, 28(11):89-90. November 1995.

Telefónica. "*MARTE: Metodología Armonizada de Telefónica*". Telefónica. 1998.

TOA. "*A Comparison of Object-Oriented Methodologies*". The Object Agency, Inc. 1995.

Tomayko, J. E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

Unidad Didáctica II: Paradigma Estructurado de Desarrollo

Objetivo genérico

Aunque la incorporación, cada vez mayor, a las empresas de nuevos titulados en Informática está influyendo de forma decisiva para que éstas vayan adoptando poco a poco los nuevos avances tecnológicos, este proceso es lento y conservador.

En un contexto donde un gran número de empresas y organizaciones no utilizan ningún tipo de metodología, y de las que las utilizan, una gran parte de ellas emplean alguna basada en los planteamientos del paradigma estructurado, los nuevos titulados deben estar preparados para incorporarse a los proyectos que se desarrollen bajo este paradigma, mantener el enorme parque de aplicaciones legadas y estar en disposición de participar en la migración metodológica que, hacia la orientación a objetos, van llevando a cabo cada vez más empresas.

Además, los currículos de las Ingenierías en Informática de la Universidad española son, en general, muy conservadores; tomando como centro de referencia este paradigma desde que el alumno cursa sus primeras asignaturas propias de Informática.

Así pues, los objetivos genéricos de esta unidad docente son *conocer la evolución del análisis y el diseño en el paradigma estructurado,* así como *conocer y manejar las técnicas del análisis y diseño estructurado orientado a procesos y a datos*.

Esta unidad docente ocupa el 52% del temario de teoría de la asignatura, estando compuesta por cuatro temas (**Análisis y especificación de requisitos**, **Análisis estructurado, Diseño del software** y **Diseño estructurado**), que se detallan a continuación.

Tema 2: ***Análisis y Especificación de Requisitos***

Descriptores

Análisis del sistema, análisis de requisitos, requisito, obtención (elicitación) de requisitos, especificación de requisitos, ERS, modelo, modelado funcional, modelado de información, modelado del comportamiento, DFD, descomposición en procesos, flujos de datos, entidades externas, almacenes de datos, extensiones de los DFD para sistemas en tiempo real, diccionario de datos, miniespecificación, diagrama entidad-relación, diagrama de transición de estados, balanceo de modelos.

Objetivos

Este tema se orienta a satisfacer los objetivos **T3** y **T4** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en la problemática de la obtención, gestión, análisis, documentación y especificación de los requisitos de un sistema software.
- Presentar los diferentes tipos y las distintas vistas de los requisitos, en concreto distinguir entre los requisitos del cliente (Requisitos-C) y los requisitos del desarrollador (Requisitos-D) [Brackett, 1990].
- Capacitar al estudiante para comprender y realizar un análisis utilizando técnicas estructuradas, en las que se le introduce en sus aspectos teóricos y prácticos más relevantes.
- Lograr que el alumno entienda las ventajas de utilizar técnicas de especificación gráficas, que facilitan la obtención de modelos y favorezcan la comunicación con clientes y/o usuarios.

Contenidos

2.1 Introducción al análisis
2.2 Especificación de requisitos del software
2.3 Técnicas de especificación
2.4 Modelado funcional
2.5 Modelado de la información
2.6 Modelado del comportamiento dinámico
2.7 Balanceo de modelos

Tabla 5.9. Contenidos del tema 2 del temario teórico de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE5 Software requirements and specification** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Requirements** [Sawyer y Kotonya, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El tema se ha dividido en siete apartados principales. El primero de ellos sirve para realizar una presentación de lo qué se entiende por análisis, sirviendo de enlace con lo introducido en el tema anterior al hablar del ciclo de vida del software.

Antes de entrar a definir qué se entiende por análisis y por requisito, debe diferenciarse qué se entiende por análisis del sistema y por análisis de requisitos. Completa esta introducción un repaso a las tareas y principios fundamentales de esta fase del ciclo de vida del software.

El segundo apartado de este tema se centra en el documento por excelencia de esta fase, la Especificación de Requisitos del Software (ERS), que engloba tanto los requisitos del cliente como los requisitos del desarrollador, aunque hay corrientes metodológicas que abogan por la división de la ERS en dos documentos, el primero de ellos se centraría en los requisitos del cliente, denominándose *Documento de Requisitos del Sistema – DRS* [Durán y Bernárdez, 2001b], mientras que el segundo recogería los requisitos del desarrollador, denominándose *Documento de Análisis del Sistema - DAS* [Durán y Bernárdez, 2001a]. La metodología Métrica 3.0 [MAP, 2001] es otro ejemplo en el que se tienen dos documentos separados para los requisitos-C y los requisitos-D.

El tercer apartado se dedica a clasificar las diferentes técnicas de especificación según dos criterios diferentes: *su modo de representación* (gráficas, textuales, marcos y matriciales) y *su enfoque de modelado* (funcional, datos y comportamiento), donde este último será el que se siga en el resto del tema para presentar las técnicas más extendidas.

En el cuarto apartado se presenta la técnica más representativa del modelado funcional, y del análisis estructurado orientado a procesos, el *Diagrama de Flujo de Datos – DFD*, incluyendo las técnicas accesorias que completan la información que en él aparece: el *Diccionario de Datos* y la *Especificación de Procesos*. Aunque se presentan las notaciones de estos diagramas según Yourdon [Yourdon, 1989], Tom De Marco [DeMarco, 1979], Gane y Sarson [Gane y Sarson, 1981] y Métrica 3.0 [MAP, 2001], la notación seguida es la de Yourdon.

Para la especificación de sistemas en tiempo real se presentan las extensiones a la notación de DFDs de Yourdon realizadas por Ward y Mellor [Ward y Mellor, 1985] y por Hatley y Pirbhai [Hatley y Pirbhai, 1987].

El apartado cinco se dedica al modelado de la información, haciendo hincapié en que dentro del paradigma estructurado la técnica por excelencia es el *Diagrama Entidad-Relación – DER* [Chen, 1976], pero no se explica esta técnica porque se considera que fue objeto de estudio detallado en la asignatura del 2º curso **Diseño de Bases de Datos**, aunque se reitera su importancia y su utilización en la presente asignatura.

El sexto apartado está dedicado al modelado de comportamiento dinámico, donde, y de nuevo por la falta de tiempo, se estudian los *Diagramas de Transición de Estados – DTE* [Harel, 1987], como la técnica más utilizada en los sistemas donde el tiempo es un factor crítico. Esta técnica sirve para especificar los estados globales de un sistema o como técnica de especificación de un proceso de control.

Por último, el séptimo apartado establece, a forma de recetario, una serie de recomendaciones para controlar la eliminación de incongruencias entre los tres modelos fundamentales del sistema (funcional, datos y comportamiento), siendo una aportación interesante la utilización de técnicas matriciales.

Bibliografía básica

Durán, A., Bernárdez, B. "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.

Hatley, Derek J., Pirbhai, I. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulos 6 y 7).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 11, y 12).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 5 y 6).

Ward, P. T., Mellor, S. J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

Yourdon, E. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993. (Capítulos 9, 10, 11, 13 y 14).

Bibliografía complementaria

Abernethy, K., Kelly, J. C. "*Comparing Object-Oriented and Data Flow Models – A Case Study*". In Proceedings of the 1992 ACM Computer Science 20th annual conference on Communications, CSC '92. (March 3-5, 1992, Kansas City, MO - USA). Pages 541-547. ACM. 1992.

Battini, C., Ceri, S., Navathe, S. B. "*Conceptual Database Design: An Entity Relationship Approach*". Benjamin/Cummings, 1992.

Boehm, B., Port, D. "*When Models Collide: Lessons from Software Systems Analysis*". IEEE IT Professional, 1(1):49-56. January/February 1999.

Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January 1990.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Burns, T., Fong E., Jefferson, D., Knox, R., Mark, L., Reedy, C., Reich, L., Roussopoulos, N., Truszkowski, W. "*Reference Model for DBMS Standardization*". SIGMOD RECORD, 15(1). March 1986.

Conger, S. "*The New Software Engineering*". Course Technology, 1994.

Champeaux, D. de (moderator), Constantine, L., Jacobson, I., Mellor, S., Ward, P., Yourdon, E. "*PANEL: Structured Analysis and Object Oriented Analysis*". In Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA90/ECOOP90. (October 21 - 25, 1990, Ottawa Canada). Pages 135-139. ACM, 1990.

Chance, B. D., Melhart, B. E. "*How to Develop Better System Requirements*". IEEE IT Professional, 1(3):70-72. May/June 1999.

Christel, M. G., Kang, K. C. "*Issues in Requirements Elicitation*". Technical Report CMU/SEI-92-TR-12 (ESC-TR-92-012). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). September 1992.

Davis, A. M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

Durán Toro, A., Bernárdez Jiménez, B., Toro Bonilla, M., Ruiz Cortés, A. "*An Object-Oriented Model and a CASE Tool for Software Requirements Management and*

Documentation". In Proceedings of the 4th Workshop MENHIR. F. J. García and J. M. Marqués editors. (May 6-7, 1999, Sedano, Burgos – Spain). Pages 6-10. 1999.

Durán Toro, A., Bernárdez Jiménez, B., Toro Bonilla, M., Ruiz Cortés, A. "*Elicitación de Requisitos de Usuario Mediante Plantillas y Patrones de Requisitos*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. P. Botella, J. Hernández y F. Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 183-194. 1999.

Ebert, C. "*Dealing with Nonfunctional Requirements in Large Software Systems*". Annals of Software Engineering, 3:367-395. 1997.

Ellison, K. S. "*Developing Real-Time Embedded Software in a Market-Driven Company*". John Wiley & Sons, 1994.

Gaitero Gordillo, D. "*Metodología Métrica. Un Enfoque Práctico*". Everest Multimedia, 1997.

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Gardarin, G. "*Dominar las Bases de Datos*". Ediciones Gestión 2000, 1993.

Hansen, G. W., Hansen, J. V. "*Diseño y Administración de Bases de Datos*". 2ª Edición. Prentice Hall, 1997.

Hawryszkiewycz, I. T. "*Introducción al Análisis y Diseño de Sistemas con Ejemplos Prácticos*". Anaya Multimedia, 1990.

Hofmann, H. F. "*Requirements Engineering*". Technical Report 93.05. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. March 1993.

Hsia, P., Kung, D., Sell, C. "*Software Requirements and Acceptance Testing*". Annals of Software Engineering, 3:291-317. 1997.

Jackson, M. "*The Meaning of Requirements*". Annals of Software Engineering, 3:5-21. 1997.

Jones, T. H., Song, I.-Y., Park, E. K. "*Ternary Relationship Decomposition and Higher Normal Form Structures Derived from Entity Relationship Conceptual Modeling*". In Proceedings on 1996 ACM on Computer science conference, CSC '96. (Feb. 16-18, 1996, Philadelphia, PA - USA). Pages 96-104. ACM. 1996.

Kowal, J. A. "*Behavior Models. Specifying User's Expectations*". Prentice-Hall International, 1992.

Lerch, F. J., Ballou, D. J., Harter, D. E. "*Using Simulation-Based Experiments for Software Requirements Engineering*". Annals of Software Engineering, 3:345-366. 1997.

Maiden, N., Gizkis, A. "*Where Do Requirements Come from*". IEEE Software, 18(5):10-12. September-October 2001.

Matheron, J.-P. "*Merise - Metodología de Desarrollo de Sistemas. Casos Prácticos*". Paraninfo, 1990.

Miguel, A. de, Piattini, M. "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al ModeloRelacional*". Ra-ma, 1993.

Miguel, A. de, Piattini, M. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Nuseibeh, B., Easterbrook, S. "*Requirements Engineering: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 35-46. ACM Press, 2000.

Oberg, R., Probasco, L., Ericsson, M. "*Applying Requirement Management with Use Cases*". Rational Software White Paper. Technical Paper TP505. Version 1.4. 2000.

Paternò, F. "*Model-Based Design and Evaluation of Interactive Applications*". Springer-Verlag, 2000.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Pohl, K. "*Requirements Engineering: An Overview*". Encyclopedia of Computer Science and Technology, 36, 1997. Disponible en http://sunsite.informatik.rwth-aachen.de/CREWS/reports96.htm.

Raghavan, S., Zelesnik, G., Ford, G. "*Lecture Notes on Requirements Elicitation*". Educational Materials. CMU/SEI-94-EM-10. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March 1994.

Rolland, C., Prakash, N. "*From Conceptual Modelling to Requirements Engineering*". Annals of Software Engineering, 10:151-176. 2000.

Ramesh, B., Stubbs, C., Powers, T., Edwards, M. "*Requirements Traceability: Theory and Practice*". Annals of Software Engineering, 3:397-415. 1997.

Sawyer, P, Kotonya, G. "*Software Requirements*". Chapter 2 in [Abran et al., 2001].

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

SEI Requirements Engineering Project. "*Requirements Engineering and Analysis. Workshop Proceedings*". Technical Report CMU/SEI-91-TR-30 (ESD-TR-91-30). Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). December 1991.

Senn, J. A. "*Análisis y Diseño de Sistemas de Información*". 2ª Edición. McGraw-Hill, 1992.

Sibley, E. H. "*Entity-Life Modeling and Structured Analysis in Real-Time Software Design – A Comparison*". Communications of the ACM, 32(12):1458-1466. December 1989.

Silva, M. "*Las Redes de Petri: En la Automática y la Informática*". Editorial AC, libros científicos y técnicos. Madrid. 1985.

Sommerville, I., Sawyer, P. "*Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering*". Annals of Software Engineering, 3:101-130. 1997.

Spence, I., Probasco, L. "*Traceability Strategies for Managing Requirements with Use Cases*". Rational Software White Paper. Version 1.0. 1998.

Zave, P. "*Classification of Research Efforts in Requirements Engineering*". ACM Computing Surveys, 29(4):315-321, 1997.

Tema 3: ***Análisis estructurado***

Descriptores

Análisis estructurado, enfoque clásico, métodos de los estímulos de Yourdon, modelo esencial, modelo ambiental, modelo de comportamiento, modelo de implantación, diagrama de contexto, diagrama de sistema, lista de acontecimientos.

Objetivos

Este tema está orientado a satisfacer el objetivo **T6** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño estructurado.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Señalar la diferencia entre la postura mantenida inicialmente por los partidarios de la descomposición funcional estricta, defendida sobretodo por el método de **Gane-Sarson** [Gane y Sarson, 1981], y el método de los estímulos que es sostenido por **Yourdon** [Yourdon, 1989].
- Familiarizar al alumno con la aplicación de un método sistemático en la aplicación de las técnicas estructuradas estudiadas en el tema anterior.

Contenidos

3.1 Introducción
3.2 Enfoque de modelado clásico
3.3 Enfoque de modelado de Yourdon

Tabla 5.10. Contenidos del tema 3 del temario teórico de Ingeniería del Software

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Requirements** [Sawyer y Kotonya, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El presente tema se divide en tres apartados. En el primero de ellos se define el concepto de análisis estructurado, que surge como una consecuencia natural de los principios que inspiraron la programación estructurada.

El segundo apartado se presenta el enfoque clásico, totalmente descendente, con un enfoque de descomposición funcional estricto, con sus cuatro modelos (el físico actual, el lógico actual, el lógico nuevo y el físico nuevo) así como también sus problemas.

El tercer apartado se dedica al método de Yourdon. Este método, también denominado método de los estímulos de Edward Yourdon, parte de la elaboración de una lista de eventos o estímulos que el sistema recibe de las entidades externas. Los estímulos provienen del exterior, o eventualmente de otros subsistemas relacionados con el que se está modelando, y se corresponden con los flujos de entrada. El sistema responde a cada uno de ellos mediante la realización de un proceso.

En sistemas complejos el método da lugar a la aparición de un gran número de procesos que son de difícil interpretación. Se realiza entonces lo que se denomina refinamiento ascendente (*upward leveling*), procedimiento consistente en agrupar los procesos afines en otro más general, ocultando en los procesos los almacenes correspondientes a los subprocesos que se agrupan. Este proceso es iterativo y se realiza tantas veces como sea necesario. El objetivo es evitar que existan en el diagrama un número excesivo de procesos. Esta es la visión ascendente (*bottom up*) del método Yourdon. Si alguno de los procesos que resultan de la lista de acontecimientos, no es primitivo, se puede refinar descendentemente, presentando el enfoque descendente de este método.

El método de los estímulos resulta algo extraño desde un planteamiento puramente estructurado, ya que parece contradecir la propia esencia del paradigma, pero, visto objetivamente, puede parecer más avanzado y se acerca más a algunas de las técnicas empleadas posteriormente.

Bibliografía básica

Yourdon, E. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993. (Capítulos 17, 18, 19, 20 y 21).

Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

Bibliografía complementaria

Miller, G. A. "*The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*". The Psychological Review, Vol. 63: 81-97, 1956. Also available at http://www.well.com/user/smalin/miller.html.

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Tema 4: ***Diseño del software***

Descriptores

Diseño arquitectónico, diseño de datos, diseño de procedimientos, diseño de la interfaz, abstracción, refinamiento sucesivo, modularidad, arquitectura del software, estructura de programa, partición estructural, estructura de datos, procedimiento del software, ocultación de la información, diseño modular, módulo, complejidad ciclotómica, independencia modular, cohesión, acoplamiento.

Objetivos

Este tema está orientado a satisfacer el objetivo **T9** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Estudio y comprensión de los fundamentos del diseño de sistemas software. El diseño es una actividad fundamental en la construcción de cualquier artefacto, lo cual debe extenderse para el software, debiendo prestar a estos temas la suficiente atención en los currículos de Informática [Rasala, 1997; Budgen, 1999].

De manera más concreta se pueden enunciar los siguientes objetivos:

- Conseguir que el alumno capte la importancia del diseño en la obtención de un software de calidad.
- Inculcar en el estudiante la necesidad de que los diseños deben respetar una serie de principios, como única forma de abordar un trabajo de forma profesional, alejado de la artesanía que tradicionalmente se venía utilizando en la construcción de los sistemas software.
- Buscar la independencia modular como criterio de calidad, desarrollando unos módulos altamente cohesionados, con bajo acoplamiento, y que hagan acopio del principio de ocultación de la información, comunicándose a través de unas interfaces precisas y de pequeño tamaño.

Contenidos

4.1 Introducción
4.2 Proceso de diseño
4.3 Actividades de diseño
4.4 Fundamentos de diseño
4.5 Diseño modular

Tabla 5.11. Contenidos del tema 4 del programa teórico de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE1 Software design** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Design** [Tremblay, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Este tema se ha divido en cinco apartados principales, intentando transmitir los principios y conceptos generales del diseño del software, necesarios para obtener software de calidad.

El primer apartado realiza una introducción que justifica la fase de diseño como necesaria para la calidad de los productos software, situándolo en el contexto del ciclo de vida. También se presenta la evolución del diseño desde la década de los setenta hasta el final de la década de los noventa.

En el segundo apartado se define el proceso de diseño y se enumeran las actividades que conlleva este proceso según diferentes autores. Ya en el tercer apartado se explican con un mayor grado de detalle las actividades relacionadas con el diseño arquitectónico, con el diseño de datos, con el diseño de los procedimientos y con el diseño de la interfaz.

El apartado cuatro se dedica a los fundamentos del diseño: *abstracción*, *refinamiento sucesivo* [Wirth, 1971], *modularidad, arquitectura de software*, *estructura del programa*, *partición estructural*, *estructura de datos*, *procedimiento del software* y *ocultación de la información* [Parnas, 1972].

Por último, el quinto apartado se dedica al diseño modular, como base para buscar una independencia modular efectiva, esto es, la calidad del diseño y, en definitiva, del producto final está directamente relacionada con un conjunto de módulos lo más independientes posibles. Existen dos medidas cualitativas de la independencia modular que son la *cohesión* y el *acoplamiento*. La cohesión es la medida del grado de relación que guardan todas las sentencias incluidas en un módulo, mientras que el acoplamiento es utilizado para medir el grado de independencia entre módulos, siendo el objetivo que se produzca en la menor medida posible.

Bibliografía Básica

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999. (Capítulo 3).

Miguel, A. de, Piattini, M. "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997. (Capítulo 8).

Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 8).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 13, 14, 15 y 22).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 10, 12, 14 y 15).

Bibliografía complementaria

Appleton, B D. "*A Software Design Specification Template*". http://www.enteract.com/~bradapp/docs/sdd.html. 1997.

Bell, D., Morrey, I., Pugh, J. "*Software Engineering. A Programming Approach*". 2nd Edition. Prentice Hall, 1992.

Booch, G. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996.

Budgen, D. "*Introduction to Software Design*". SEI Curriculum Module SEI-CM-2-2.1. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January 1989.

Date, C. J. "Introducción a los Sistemas de Bases de Datos". 7ª Edición, Addison-Wesley, 2001.

Davis, W. S. "*Herramientas CASE. Metodología Estructurada para el Desarrollo de los Sistemas*". Paraninfo, 1992.

Fairley, R. "*Ingeniería del Software*". McGraw Hill, 1988.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Graham, I. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Guttag, J. "*Abstract Data Types and the Development of Data Structures*". Communications of the ACM, 20(6):396-404. June 1977.

Hix, D., Hartson, H. R. "*Developing User Interfaces: Insuring Usability through Product and Process*". John Wiley & Sons, 1993.

Hofmann, H. F., Pfeifer, R., Vinkhuyzen, E. "*Situated Software Design*". Technical Report. Institut für Imformatik der Universität Zürich. Institute of Informatics, University of Zurich. Winterthurerstr, 190. CH-8057, Zurich. 1993.

Hofmeister, C., Nord, R., Soni, S. "*Applied Software Architecture*". Addison-Wesley, Object Technology Series, 2000.

Kaman Sciences Corporation. "*A State of the Art Report: Software Design Methods*". Kaman Sciences Corporation. March 1994.

Lores, J. (Editor). "*Curso Introducción a la Interacción Persona-Ordenador. El Libro Electrónico*". http://griho.udl.es/ipo/libroe.html. Enero, 2002.

Marqués Corral, J. M. "*Diseño de Interfaces de Usuario*". Apuntes de la asignatura Ingeniería del Software II de la Ingeniería Técnica de Informática de Sistemas. Universidad de Valladolid.

Newman, M., Lamming, G. "*Interactive System Design*". Addison-Wesley, 1995.

Parnas, D. L. "*Designing Software for Ease of Extension and Contraction*". IEEE Transactions on Software Engineering, SE-5(2):128-138. March 1979.

Parnas, D. L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972.

Paternò, F. "*Model-Based Design and Evaluation of Interactive Applications*". Springer-Verlag, 2000.

Peña Marí, R. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Ed. Prentice-Hall, 1998.

Perrochon, L., Mann, W. "*Inferred Designs*". IEEE Software, 16(5):46-51. September/October 1999.

Rivero Cornelio, E. "*Bases de Datos Relacionales*". Paraninfo, 1991.

Rumbaugh, J. "*Bridging the Gap. Building Complex Systems by Leveling and Layering*". Rose Architect Magazine, 2(2). Winter, 1999.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Shaw, M., Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

Shneiderman, B. "*Designing the User Interface. Strategies for Effective Human-Computer Interaction*". 2nd Edition. Addison-Wesley, 1992.

Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

Wirth, N. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April 1971.

Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

Tema 5: ***Diseño estructurado***

Descriptores

Diagrama de estructuras, tabla de interfaz, estrategias de diseño, análisis de transformación, análisis de transacción, centro de transformación, centro de transacción.

Objetivos

Este tema está orientado a satisfacer el objetivo **T6** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño estructurado.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Estudiar la transformación de los modelos realizados en análisis a sus correspondientes en diseño. En el caso particular del tema de los DFDs a los diagramas de estructuras y del modelo entidad/relación a un modelo relacional normalizado.
- Distinguir entre las estrategias de diseño de transformación y transacción a la hora de transformar los DFDs en diagramas de estructuras.

Contenidos

5.1 Introducción
5.2 Diagrama de estructuras
5.3 Estrategias de diseño

Tabla 5.12. Contenidos del tema 5 del programa de teoría de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE1 Software design** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Design** [Tremblay, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El objetivo principal de este tema es explicar la forma correcta de pasar del análisis al diseño, haciendo hincapié en el uso de los denominados diagramas de estructuras que provienen de los diagramas de flujo de datos, es decir, en el diseño arquitectónico. Esto es así porque se entiende que el alumno ya está familiarizado de forma práctica con el resto de las actividades de diseño. Así, en la asignatura de **Diseño de Bases de Datos** ha estudiado el paso del modelo conceptual al modelo lógico de datos y la teoría de la normalización, en la asignatura de **Interfaces**

Gráficas se ha profundizado en el diseño de interfaces gráficas de usuario, y en las asignaturas de programación ha practicado el diseño de algoritmos previamente a su codificación.

En el primero de los tres apartados en que se organiza el tema se hace una introducción al diseño estructurado, enlazando con lo estudiado en el tema anterior, y justificando (tal como se ha hecho en el párrafo previo) la orientación del mismo hacia el diseño arquitectónico.

En el segundo apartado se estudia la técnica fundamental del diseño arquitectónico dentro del paradigma estructurado, los *diagramas de estructura*, también conocidos por el término en inglés *structure chart*, o por *diagrama de estructura de cuadros de Constantine* [MAP, 1995; MAP, 2001].

En la tercera, y última, parte de este tema se estudian las estrategias de diseño, que son aquéllas que se emplean para la transformación del modelo funcional basado en DFDs en los diagramas de estructuras que constituyen el diseño arquitectónico.

Lo primero sobre lo que se llama la atención es lo *artificial* de este proceso, que obliga a cambiar de modelo subyacente en la técnica de modelado: *de los procesos del DFD* se pasa *a la jerarquía de módulos*.

Se distinguen dos tipos de flujos de datos: *el flujo de transformación* y *el flujo de transacción*. El flujo de transformación reproduce fielmente el patrón entrada/proceso/salida, de manera que las decisiones las toman los módulos de ámbito superior y las operaciones se llevan a cabo por los módulos de orden inferior. El flujo de transacción establece dos módulos principales, uno que analiza la transacción a ser tratada y otro que encamina hacia el módulo que la trata. En estos sistemas se tienen diferentes caminos de acción, independientes los unos de los otros, de forma que la transacción servirá de elemento discriminador para determinar el camino a elegir.

La existencia de dos tipos de flujos de datos origina dos estrategias de diseño, la primera de ellas, el análisis de transformación, se aplica en aquellos sistemas donde la característica fundamental sea de flujo de trasformación, mientras que la segunda, el análisis de la transacción, se aplica en los sistemas cuya característica principal sea de transacción.

Bibliografía básica

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 8).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 14).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 10, 12, 14 y 15).

Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.

Bibliografía complementaria

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Page-Jones, M. "*The Practical Guide to Structured Systems Design*". 2nd edition. Prentice Hall, 1988.

Unidad Didáctica III: Introducción al paradigma objetual

Objetivo genérico

El objetivo que se plantea esta unidad docente es la introducción al alumno en el enfoque de análisis y diseño orientado a objetos para el desarrollo de software, cuyo fin es la construcción de modelos de objetos computacionalmente implantables, obtenidos por observación del mundo real, respetando y reproduciendo sus agentes y vínculos.

Aunque el mayor esfuerzo dentro de este paradigma se ha hecho en los lenguajes de programación, es, sin embargo, en las fases de análisis y diseño donde su empleo puede dar lugar a unos resultados más brillantes.

Es importante insistir en que una buena técnica de diseño retrasa todo lo posible los detalles de implementación, ya que de esta forma se gana flexibilidad. Lo que realmente importa es definir el problema, captar sus requisitos y ser capaces de plantear una solución a las necesidades de información, que se acople de la manera más natural posible a la organización y a las responsabilidades de los usuarios.

También debe resaltarse que es esencial evitar los errores en las primeras fases del desarrollo, porque su influencia sobre la calidad del sistema es fundamental.

Debe presentarse, como uno de los aspectos más importantes del nuevo paradigma, que supone una forma de pensar distinta acerca del problema que se trata de resolver, incorporando de una forma más natural las características relevantes de los objetos del mundo real. Se trata en suma de modelar los objetos del negocio, o del domino del problema, es decir, los conceptos familiares a los usuarios. Estos conceptos no sólo implican estructura de datos, sino que además llevan asociado el comportamiento, que representa la responsabilidad de la entidad objetualizada ante los vínculos con otros objetos. Por tanto, es el usuario el que, como conocedor de la realidad del dominio del problema, aclara el comportamiento que se espera que tengan los objetos del modelo.

Ahondando en el punto anterior, y desde la perspectiva del análisis orientado a objetos, lo más importante es plantearse el problema bajo el enfoque del usuario, y no bajo el planteamiento meramente informático o tecnológico. Esto significa que el esfuerzo en la tarea del desarrollo del software debe hacerse más en la identificación de los objetos pertenecientes al dominio del problema, que en la definición de los objetos relacionados con la aplicación desde el punto de vista de su entorno de implementación.

El paso al diseño y a la implementación es más natural que en el paradigma estructurado, simplemente por el mero hecho de que el modelo subyacente a todas las fases del ciclo vital del software es el mismo, el modelo objeto, llegándose a lo que se denomina un proceso de construcción de software sin costuras [Nerson, 1992]. Así, el paso de los modelos de análisis a los de diseño no conlleva una transformación, como en el paradigma estructurado, sino que se hace por elaboración de los primeros, esto es, los objetos del dominio del problema se refinan, perfilando más sus características, y se añaden los objetos de interfaz, de utilidad y de aplicación [Monarchi y Puhr, 1992].

La inclusión de esta unidad docente en el programa de la asignatura de Ingeniería del Software está plenamente justificada desde diversos puntos de vista:

- Los métodos de desarrollo de software orientados a objetos están contemplados en los diferentes currículos y cuerpos de conocimiento que se han tenido en cuenta.
- Son las referencias que abogan por un enfoque orientado a objetos para la asignatura de Ingeniería del Software [Donadi, 1992; Jacobson et al., 1993; Bruegge y Dutoit, 2000].
- La evolución que están sufriendo los métodos y las herramientas de desarrollo, cada día más asentadas sobre la tecnología de objetos, soportando los ciclos de vida iterativos e incrementales, de desarrollo rápido de aplicaciones, paradigmas visuales... todo ello bastante alejado del paradigma estructurado.

Esta unidad docente ocupa el 29% del temario de teoría de la asignatura, estando compuesta por tres temas (**Introducción a la orientación a objeto**, **UML** y **Visión general de la metodología OMT**), que se detallan a continuación.

Tema 6: ***Introducción a la Orientación a Objetos***

Descriptores

Tecnología de objetos, reutilización del software, modelo objeto, abstracción, encapsulamiento, modularidad, jerarquía, paso de mensajes, tipo, polimorfismo, clase, objeto, atributo, método, estado, comportamiento, identidad, análisis orientado a objetos, diseño orientado a objetos.

Objetivos

Este tema está orientado a satisfacer los objetivos **T3** y **T7** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en los conceptos fundamentales de la tecnología de objetos, explicándole las diferencias que presenta esta manera de desarrollar software respecto a los planteamientos estructurados.
- Insistir en que los objetos representan elementos del mundo real dentro del dominio del problema que se debe resolver, siendo por tanto, familiares al usuario. Los objetos llevan incorporado el comportamiento habitual que el experto en el dominio espera de ellos.
- Destacar dentro de los conceptos fundamentales, el especial relieve que tienen para la obtención de un software de calidad los conceptos de clasificación (jerarquía) y encapsulación.
- Señalar el aumento de productividad que se alcanza gracias al polimorfismo y la herencia, pero también el significado avanzado de estos conceptos en las fases de análisis y diseño.
- Recalcar la trascendencia de la orientación a objetos en el análisis de aplicaciones, ya que su faceta más conocida es en la fase de codificación.
- Comentar y discutir los conceptos básicos del *modelo objeto* basándose en las aportaciones de varios autores, y el modo en el que pueden soslayarse las posibles limitaciones tanto conceptuales, como de notación.
- Introducir el concepto de reutilización del software, su importancia para el aumento de la productividad en el desarrollo del software y su relación con la

orientación a objetos. En este sentido debe hacerse hincapié en que la reutilización del software está presente en todo el ciclo de vida, no sólo en la implementación, debiendo empezar a pensar en ella en las primeras fases del ciclo de vida.

Contenidos

6.1 Introducción y evolución de la Orientación a Objetos
6.2 Modelo objeto
6.3 Análisis y diseño orientados a objetos

Tabla 5.13. Contenidos del sexto tema del programa de teoría de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE1 Software design** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Design** [Tremblay, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Este tema, como se aprecia en la Tabla 5.13, se divide en tres apartados principales. El primero de ellos sirve como introducción de la tecnología de objetos. Aunque a lo largo de la asignatura ya se ha hecho mención al paradigma objetual (y los alumnos han manejado de forma intuitiva los objetos en la asignatura de Interfaces Gráficas de segundo curso), este apartado supone la "presentación oficial" de la orientación a objetos en su currículo.

En este primer apartado se justifica el auge de la tecnología de objetos para que, con una manera diferente de enfocar el desarrollo del software, se pueda hacer frente a la continua demanda social de software de mayor calidad, complejidad y sencillez de manejo.

Se discuten los beneficios potenciales, las ventajas y los inconvenientes (mitos mayormente) y peligros de la orientación a objetos. Dentro de los beneficios que se pueden lograr con el uso de la tecnología de objetos se hace mucho hincapié en la reutilización del software, primero para resaltar su importancia en la consecución de sistemas de software de calidad realizados con alta productividad, segundo para aclarar que la reutilización del software es ortogonal a la orientación a objetos (y a cualquier otro método de desarrollo) aunque este paradigma facilita enormemente tanto el *desarrollo para reutilización* como el *desarrollo con reutilización*, y tercero que la reutilización no es un fin sí misma, sino una forma de enfrentarse de una forma competitiva al desarrollo de software de calidad [García, 2000].

El segundo apartado se dedica a la presentación del modelo objeto, como denominador común de todos los conceptos y principios que debe cumplir un paradigma de desarrollo para

ser considerado orientado a objetos. Aunque no existe un acuerdo completo entre los especialistas, se va a exponer lo que, a juicio de Grady Booch [Booch, 1994], constituyen los elementos fundamentales de un modelo objeto. En la explicación de los mismos tendrán cabida diferentes perspectivas, aunque predominan las cercanas al modelo objeto subyacente en UML [OMG, 2001c], al ser éste un lenguaje de modelado considerado como estándar por OMG, y que será objeto de estudio en el tema 7 de esta asignatura.

El tercer apartado se dedica a introducir las fases de análisis y diseño en los desarrollos orientados a objetos. Se enfatiza como la fase de obtención de requisitos es independiente de si se va a desarrollar orientado a objetos o estructurado, igual que el énfasis de la fase de análisis sigue siendo el *qué hacer* y la misión del diseño modelar el *cómo hacerlo*.

En la orientación a objetos el software se organiza como una colección de objetos discretos, los cuales contienen datos y comportamiento; manteniéndose este modelo a lo largo de todo el ciclo de vida, lo que permite una transición mucho más suave entre las diferentes fases de éste, a la vez que se facilita la utilización de modelos de ciclo de vida iterativos e incrementales.

De forma explícita se comentan diversas actividades a realizar en las fases de análisis y diseño para favorecer el desarrollo para reutilización, lo que se presenta como una máxima para todo ingeniero del software.

Bibliografía básica

Booch, G. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996. (Capítulos 1, 2, 3 y 4).

Graham, I. "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996. (Capítulos 1, 7 y 8).

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999. (Capítulos 5, 7, 14, 15, 22, 23 y 24).

OMG. "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001. (Capítulo 2, 3 y Glosario).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 10).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 20, 21 y 22).

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998. (Capítulos 1, 3 y 4).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 12).

Bibliográfica complementaria

Berard, E. V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc., 1996.

Booch, G. "*Objectifying Information Technology*". Rational Software Corporation. http://www.rational.com. 1998.

Budd, T. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

Champeaux, D., Lea, D., Faure, P. "*Object-Oriented System Development*". Addison Wesley, 1993.

Durán Toro, A., Bernárdez Jiménez, B. "*Metodología para el Análisis de Requisitos de Sistemas Software. (versión 2.2)*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, diciembre de 2001.

Engels, G., Groenewegen, L. "*Object-Oriented Modeling: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 103-104. ACM Press, 2000.

Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman). Pages 2-8. IEEE Computer Society Press, 1987.

García Peñalvo, F. J., Pardo Aguilar, C. "*Introducción al Análisis y Diseño Orientado a Objetos*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(2):64-70. Febrero, 1998.

Girow, A. "*Objects and Binary Relations*". Object Currents, 1(6), SIGS Publications, June 1996.

Girow, A. "*Binary Relations Approach to Building Object Database Model*". Object Currents, 1(11), SIGS Publications, November 1996.

Henderson-Sellers, B. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Joyanes Aguilar, L. "*Programación Orientada a Objetos*". 2ª Edición. Osborne McGraw-Hill. 1998.

Kaindl, H. "*Difficulties in the Transition from OO Analysis to Design*". IEEE Software, 16(5):94-102. September/October 1999.

Krueger, C. W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June 1992.

Lewis, T. "*The Dark Side of Objects*". IEEE Computer, 27(12):6-7. December 1994.

Liskov, B. "*Data Abstraction and Hierarchy*". SIGPLAN Notices, Vol. 23(5), 1988.

Martin, J., Odell, J. J. "*Métodos Orientados a Objetos: Conceptos Fundamentales*". Prentice Hall, 1997.

Martin, J., Odell, J. J. "*Métodos Orientados a Objetos: Consideraciones Prácticas*". Prentice Hall Hispanoamericana, 1997.

Monarchi, D. E., Puhr, G. I. "*A Research Typology for Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):35-47. September 1992.

Nerson, J.-M. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September 1992.

Oestereich, B. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

Parnas, D. L. "*On the Criteria To Be Used in Descomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972.

Piattini Velthuis, M. G. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

Rine, D. "*Object-Oriented Technology and Software Reuse*". IEEE Computer, 26(7):6. July 1993.

Robinson, P. J. "*Hierarchical Object-Oriented Design*". Object-Oriented Series. Prentice-Hall, 1992.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Snyder, A. "*The Essence of Objects: Concepts and Terms*". IEEE Software, 10(1):31-42. January/February 1993.

Sutherland, J. "*Object World Tutorial - Object Design Tutorial*". 1997.

Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice Hall, 1990.

Yourdon, E., Argila, C. "*Case Studies in Object Oriented Analysis & Design*". Yourdon Press Computing Series. Prentice Hall, 1996.

Yourdon, E., Whitehead, K., Toman, J., Oppel, K., Nevermann, P. "*Mainstream Objects. An Analysis and Design Approach for Business*". Yourdon Press, 1995.

Tema 7: *UML*

Descriptores

UML, lenguaje de modelado, modelo, vista, diagrama, elemento de modelado, diagrama estático de estructura, diagrama de clase, clase, atributo, método, asociación, adorno de asociación, cardinalidad, calificador, agregación, composición, generalización/especialización, paquete, diagrama de interacción, diagrama de secuencia, diagrama de colaboración, mensaje, caso de uso, actor, relación entre casos de uso.

Objetivos

Este tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Entendimiento de qué es y qué no es UML, así como presentación somera de sus orígenes e historia.
- Toma de contacto inicial con el vocabulario, las reglas, formas de empleo de UML y, en general, con el manejo de la simbología y el adecuado alcance de los conceptos.
- Iniciación en el modo de aplicar los modelos de UML para resolver los problemas implícitos en el desarrollo del software.
- Dominio del modelado estructural a un nivel básico.
- Introducción del concepto de caso de uso y de su importancia en las corrientes metodológicas actuales, bien como herramienta de extracción y documentación de los requisitos del sistema software, bien como herramienta para modelar el comportamiento esencial de un sistema o subsistema en conjunción con otros diagramas (colaboración, secuencia, transición de estados...).

Contenidos

7.1 Introducción
7.2 Una visión general de UML
7.3 Diagramas de estructura
7.4 Diagramas de interacción
7.5 Casos de uso

Tabla 5.14. Contenidos del séptimo tema del programa de teoría de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE1 Software design** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Design** [Tremblay, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El tema se ha dividido en cinco apartados principales, que tienen como cometido introducir al alumno en el modelado orientado a objetos a través de UML. No se pretende que el alumno obtenga un dominio completo de UML, sino que perciba los conceptos básicos, basados en los diagramas de clase, de casos de uso y de secuencia. Posteriormente, en la asignatura **Análisis de Sistemas**, del primer curso del segundo ciclo, se completarán los conceptos aquí introducidos.

El primer apartado, como bien indica su nombre, es una introducción a los orígenes y cometido de UML. Se justifica el porqué de la necesidad de un estándar dentro de los lenguajes de modelado, UML en este caso, desde el punto de vista de lo que supone para las organizaciones donde se desarrolla software.

El modelado no es exclusivo para los grandes sistemas [Moitra, 1999]. Sin embargo, es una verdad absoluta que el sistema más grande y complejo conlleva el más importante esfuerzo de modelado, y esto es así porque se *construyen modelos de los sistemas complejos ante la incapacidad humana de entender completamente como son éstos.*

El modelado de sistemas en la Ingeniería Informática necesita un lenguaje común, de la misma manera que otras ingenierías disponen de lenguajes *normalizados*. Así, existen lenguajes para la industria de la construcción, la industria eléctrica... Los matemáticos disponen de un lenguaje universal, y además formal, de modelado. Pero no es una cuestión del grado de formalismo del modelo (muchos esquemas informales resultan adecuados en su aspecto expresivo para los fines de diseño y descripción que con ellos se persiguen). Es más importante la necesidad de resolver una carencia que tenían las actividades de modelado en el ámbito informático: *las técnicas empleadas carecían de un lenguaje estándar*.

UML es un **lenguaje**, porque tiene un vocabulario y unas reglas para combinar sus términos con el propósito de lograr la comunicación. Es un **lenguaje de modelado**, porque su vocabulario y sus reglas se dirigen a la representación conceptual y física de un sistema. UML es un **estándar** porque constituye una forma común de expresar las especificaciones para el software, habiéndose admitido como tal por la comunidad de la orientación a objetos (estándar

de facto) y porque ha sido reconocido de esta manera por un organismo internacional cualificado como es OMG (estándar oficial).

En lo tocante a la evolución histórica de UML puede decirse que los lenguajes de modelado orientados a objetos hacen su aparición entre la mitad de la década de los setenta y finales de la década de los ochenta, con una orientación claramente metodológica, conjugada con un nuevo género de lenguajes de programación orientados a objetos e impulsada por el incremento constante de la complejidad de las aplicaciones. Los metodólogos, superando su exclusividad para la programación, comienzan la aproximación del paradigma objetual a las tareas del análisis y diseño. El número de métodos crece desde poco más de diez en el año 1989 a más de cincuenta en 1994 [García y Pardo, 1998].

Muchos usuarios interesados en la orientación a objetos, no encuentran un método que cubra completamente sus necesidades, lo que motiva la aparición de métodos de mayor entidad metodológica, normalmente resultado de la evolución o de la fusión de los ya existentes.

Mediada la década de los noventa, los responsables directos de tres de los métodos de desarrollo orientado a objetos de mayor difusión, Grady Booch, James Rumbaugh e Ivar Jacobson, unen sus esfuerzos para unificar sus métodos, comenzando por la definición de un lenguaje de modelado.

Oficialmente los trabajos para crear UML comienzan en 1994. Al comienzo el proyecto se centró en la unificación de los métodos de Booch [Booch, 1994] y OMT [Rumbaugh et al., 1991], saliendo a la luz la versión 0.8 del llamado Método Unificando en octubre de 1995 [Booch y Rumbaugh, 1995]. Con la incorporación de Ivar Jacobson, se pospone la definición del método, poniendo el énfasis en el lenguaje de modelado, surgiendo la versión 0.9 de UML en junio de 1996 [Booch et al., 1996], la primera versión de UML. En noviembre de 1997, la versión 1.1 de UML [Rational et al., 1997] se adopta como estándar por OMG, siendo la versión 1.4 la que actualmente se encuentra en vigor [OMG, 2001c].

En el apartado segundo del tema se hace un recorrido rápido por UML, diferenciándose las vistas, los diagramas, los elementos de modelado y los mecanismos generales.

El apartado tres se dedica a estudiar los diagramas de estructura en general, aunque el énfasis se pone en el estudio de los diagramas de clase en particular, con un nivel de detalle tal que no se entra en disquisiciones avanzadas. Un diagrama de clase puede utilizarse desde diferentes perspectivas [Fowler y Scott, 2000]: *conceptual* (el diagrama de clase representa los conceptos en el dominio del problema que se está estudiando), *especificación* (el diagrama de

clase refleja las interfaces de las clases, pero no su implementación) e *implementación* (representa las clases tal cual aparecen en el entorno de implementación).

El cuarto apartado se dedica al estudio de los diagramas de interacción, que son modelos que describen como grupos de objetos colaboran para conseguir algún fin [OMG, 2001c]. Una interacción es, por tanto, un comportamiento que comprende un conjunto de mensajes intercambiados entre objetos dentro de un contexto. En UML se definen dos tipos de diagramas de interacción, los diagramas de secuencia y los diagramas de colaboración.

El quinto y último apartado del tema se dedica a los *casos de uso*. Los casos de uso, introducidos inicialmente por Ivar Jacobson [Jacobson, 1987], se pueden utilizar para la identificación y documentación de los requisitos funcionales de un sistema software, así como para el modelado de las interacciones entre el sistema y los actores en los escenarios identificados. Un caso de uso es una descripción de un conjunto de secuencias de acciones que ejecuta el sistema, y que producen un resultado útil para el actor productor del evento que arranca el citado caso de uso.

Bibliografía básica

Booch, G., Rumbaugh, J., Jacobson, I. "*El Lenguaje Unificado de Modelado*". Object Technology Series. Addison Wesley, 1999. (Capítulos 1, 2, 4, 5, 6, 7, 8, 9 y 10).

Fowler, M., Scott, K. "*UML Gota a Gota*". Addison Wesley, 1999. (Capítulos 1, 3, 4 y 5).

Muller, P.-A. "*Modelado de Objetos con UML*". Ediciones Gestión 2000, 1997. (Capítulo 3).

OMG. "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001. (Capítulo 2, 3 y Glosario).

Rumbaugh, J., Jacobson, I., Booch, G. "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.

Bibliografía complementaria

Bell, A. E., Schmidt, R. W. "*UMLoquent Expression of AWACS Software Design*". Communications of the ACM, 42(10):55-61. October 1999.

Berard, E. V. "*Be Careful with 'Use Cases'*". The Object Agency, Inc., 1995.

Blaha, M., Premerlani, W. "*Object-Oriented Design of Database Applications*". Rose Architect, 1(2). January 1999.

Blaha, M., Premerlani, W. "*Implementing UML Models with Relational Databases*". Rose Architect, 1(3). April 1999.

Booch, G. "*Quality Software and the UML*". Object Magazine. March 1997.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Cantor, M. R. "*Object-Oriented Project Management with UML*". Wiley & Sons, 1998.

Cockburn, A. "*Writing Effective Use Cases*". Addison Wesley, 2000.

Collins-Cope, M. "*The Requirements/Service/Interface (RSI) Approach to Use Case Analysis (A Pattern for Structured Use Case Development)*". Ratio Group Ltd. http://www.ratio.co.uk/RSI.htm. 1999.

Conallen, J. "*Modeling Web Application Architectures with UML*". Communications of the ACM, 42(10):63-70. October 1999.

Christerson, M. "*From Use Cases to Components*". Rose Architect, 1(1). October 1998.

D'Souza, D. F., Wills, A. C. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

Díaz, O., Rodríguez, J. J. "*Change Case Analysis*". Journal of Object-Oriented Programming (JOOP), 12(9):9-15,48. February 2000.

Durán Toro, A., Bernárdez Jiménez, B. "*Metodología para el Análisis de Requisitos de Sistemas Software. (versión 2.2)*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Diciembre de 2001.

Durán, A., Bernárdez, B. "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.

Eriksson, H.-E., Penker, M. "*UML Toolkit*". John Wiley & Sons, 1998.

Firesmith, D. G., Henderson-Sellers, B. "*Upgrading OML to Version 1.1: Part 1 Referencial Relationships*". Journal of Object-Oriented Programming (JOOP), 11(3):48-57. June 1998.

Firesmith, D. G., Henderson-Sellers, B. "*Upgrading OML to Version 1.1: Part 2 Additional Concepts and Notation*". Journal of Object-Oriented Programming (JOOP), 11(5):61-67. September 1998.

Firesmith, D., Henderson-Sellers, B., Graham, I., Page-Jones, M. "*OPEN Modeling Language (OML) Reference Manual*". Version 1.0. OPEN Consortium. December 1996.

Fowler, M. "*A Survey of Object-Oriented Analysis and Design Techniques*". Technical-Report, 1997.

Fowler, M. "*Use and Abuse Cases*". Distributed Computing. April 1998.

Fowler, M., Scott, K. "*UML Gota a Gota*". Addison Wesley Longman (Pearson), 1999.

García Peñalvo, F. J., Pardo Aguilar, C. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

García Peñalvo, F. J., Pardo Aguilar, C. "*Diagramas de Clase en UML 1.1*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(3):71-76. Marzo, 1998.

Henderson-Sellers, B. "*Choosing between UML and OPEN*". 1997.

Henderson-Sellers, B. "*OML: Proposals to Enhance UML*". In Proceedings of <<*UML*>>'*98 Beyond the Notation* Conference. (3rd-4th June 98, Mulhouse - France). June 1998.

Henderson-Sellers, B., **Simons, T., Younessi, H.** "*The OPEN Toolbox of Techniques*". Open Series. Addison-Wesley, 1998.

Iturrioz Sánchez, J. I. "*Una Metodología para el Desarrollo de Sistemas de Base de Datos Objeto-Relacional*". Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos. Universidad del País Vasco – Euskal Herriko Unibertsitatea. Junio, 1998.

Jacobson, I. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Jacobson, I., Griss, M., Jonsson, P. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley, 1997.

Kenworthy, E. "*Use Case Modelling. Capturing User Requirements*". http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.html. [Última vez visitado, 2-2-1999]. December 1997.

Kobryn, C. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October 1999.

Kruchten, P. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November 1995.

Larman, C. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Letelier Torres, P., Sánchez Palma, P. "*Análisis y Diseño Orientado a Objetos Usando la Notación UML*". Notas del curso. Valencia, febrero de 2001.

Liberty, J. "*Beginning Object-Oriented Analysis and Design with C++*". Wrox Press Ltd., 1998.

López, N., Migueis, J., Pichon, E. "*Integrar UML en los Proyectos*". Gestión 2000, 1998.

Lunn, K. "*Object Oriented Analysis and Design – Course Notes*". 1997.

Odell, J. J. "*Advanced Object-Oriented Analysis and Design Using UML*". Cambridge University Press. SIGS Books, 1998.

Oestereich, B. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

Ohnjec, V. "*Converging on OOAD Agreement*". Applications Development Trends, 4(2). February 1997.

Rational Software Corporation. "*Inside the Unified Modeling Language - UML*". Multimedia Educational Tool. April 1999.

Rational Software Corporation. "*Unified Modeling Language for Real-Time Systems Design*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 16/6/97]. 1997.

Reenskaug, T., Wold, P., Lehne, O. A. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

Rivas, E., DeSilva, D., McDaniel, T., Atkinson, C. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January 1997.

Rosenberg, D., Scott, K. "*Use Case Driven Object Modeling with UML. A Practical Approach*". Object Technology Series. Addison-Wesley, 1999.

Rumbaugh, J. "*Building Boxes: Subsystems*". Journal of Object-Oriented Programming, 7(6): 16-21. October 1994.

Rumbaugh, J. "*Disinherited! Examples of Misuse of Inheritance*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. February 1993.

Rumbaugh, J. "*Driving to a Solution. Reification and the Art of System Design*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. July 1995.

Rumbaugh, J. "*Getting Started. Using Use Cases To Capture Requirements*". Journal of Object-Oriented Programming (JOOP), 7(5):8-12, 23. September 1994.

Rumbaugh, J. "*Taking Things in Context. Using Composites to Build Models*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. November 1995.

Rumbaugh, J. "*The View from the Front: Changes to UML by the Revision Task Force*". Rose Architect, 1(3). Summer 1999.

Schneider, G., Winters, J. P. "*Applying Use Cases. A Practical Guide*". Object Technology Series. Addison-Wesley, 1998.

Selic, B. "*Turning Clockwise: Using UML in the Real-Time Domain*". Communications of the ACM, 42(10):46-54. October 1999.

Selic, B., Rumbaugh, J. "*Using UML for Modeling Complex Real-Time Systems*". Technical Report. March 1998.

Stevens, P., **Pooley, R.** "*Utilización de UML en Ingeniería del Software con Objetos y Componentes*". Addison-Wesley, 2002.

Texel, P. P., Williams, C. B. "*Use Cases Combined with Booch, OMT UML. Process and Products*". Prentice Hall, 1997.

Vadaparty, K. "*Use Cases - Basics*". Journal of Object-Oriented Programming (JOOP), 12(9). February 2000.

Warmer, J., Kleppe, A. "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley. Object Technology Series, 1999.

Whitlock, D. "*The Unified Modeling Language*". http://watson2.cs.binghamton.edu/~dwhitloc/uml/paper/part1.html. 1999.

Zhang, D. D. "*Use Case Modeling for Real-time Application*". In Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'99. (27 - 29 January 1999, Santa Barbara, California – USA). Pages 56-64. IEEE Computer Society, 1999.

Zhao, L., Foster, T. "*Modeling Roles with Cascade*". IEEE Software, 16(5):86-93. September/October 1999.

Tema 8: ***Visión general de la metodología OMT***

Descriptores

OMT, análisis, modelos del mundo real, requisitos, definición del problema, diseño, arquitectura del sistema, subsistemas.

Objetivos

Este tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Integrar las distintas técnicas aprendidas dentro de una aproximación metodológica como puede ser OMT (*Object Modeling Technique*), aunque la metodología concreta puede variar en próximos cursos, siendo sustituida por ejemplo por UP (*Unified Process*) [Jacobson et al., 1999].
- Recalcar la importancia que tiene para el éxito de un proyecto software partir de una adecuada definición del problema y utilizar un enfoque sistemático para el desarrollo de su solución.

Contenidos

8.1 Introducción
8.2 Fases
8.3 Análisis
8.4 Diseño del sistema
8.5 Diseño de los objetos

Tabla 5.15. Contenidos del tema 8 del programa de teoría de Ingeniería del Software

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El tema se organiza en cinco apartados. El primero de ellos sirve para introducir los orígenes, evolución y objetivos de la metodología OMT (*Object Modeling Technique*). Se hace especial hincapié en diferenciar entre la primera versión de OMT [Rumbaugh et al., 1991] y su evolución hacia OMT-2, menos conocida al quedar eclipsada por la eclosión de UML, que se recoge en diversos artículos de James Rumbaugh en la revista JOOP (*Journal of Object-*

Oriented Programming) [Rumbaugh, 1995a; Rumbaugh, 1995b; Rumbaugh, 1995c; Rumbaugh, 1995d] y posteriormente en su libro *OMT Insights* [Rumbaugh, 1996].

OMT-2 puede considerarse como la precursora de la versión 0.8 del método unificado de Booch y Rumbaugh [Booch y Rumbaugh, 1995].

En el apartado dos se enuncian y se presentan de forma somera las cuatro fases de que consta esta metodología: *análisis, diseño del sistema, diseño de los objetos* e *implementación*.

El tercer apartado se dedica a la fase de análisis, el punto más sobresaliente de esta metodología. Se explica la importancia que tiene la obtención de los requisitos en la etapa de definición del problema. En el proceso de análisis se utilizan como entradas, además de las especificaciones del problema, las entrevistas mantenidas con los usuarios, el conocimiento personal de los ingenieros del software sobre el dominio del problema y la experiencia que se haya adquirido sobre el mundo real ante planteamientos parecidos.

Para la construcción de un modelo de objetos, que va a representar la estructura estática de los datos que se corresponden con el mundo real que se está estudiando, se indica la importancia de construir inicialmente el modelo al más alto nivel de abstracción, incluyendo las asociaciones entre las clases. Se deben posponer a etapas ulteriores los refinamientos de la inclusión de generalización y especialización e incluso algunas estructuras de agregación. Después de obtener el primer modelo surge la necesidad de refinarlo.

El modelo de objetos final difícilmente se alcanzará en una sola aproximación, siendo necesario realizar varias iteraciones. Inclusive alguna de estas iteraciones se realizará después de realizar los otros modelos propios de la fase de análisis.

Una vez conseguido el modelo estático suficientemente descriptivo, y en el que las asociaciones representen el sentido del sistema, el paso siguiente es lograr un modelo dinámico, que regule el flujo de control de los comportamientos. Para este fin debe construirse primero un escenario. Una manera adecuada para ello consiste en plantear el típico diálogo entre el sistema y el usuario. De esta forma se tendrá una idea del comportamiento que se espera del sistema. Conviene insistir en la necesidad de construir no sólo escenarios normales, sino también especiales o correspondientes a situaciones excepcionales.

El modelo funcional propio de OMT ha sido bastante criticado, porque puede llevar a concebir una descomposición funcional del sistema en lugar de seguir un enfoque orientado al objeto. Actualmente [Rumbaugh, 1994], se propone que este modelo consista en la descripción detallada de las operaciones del sistema, utilizando plantillas, y sólo en el caso en que una

operación afecte a varios objetos del sistema, se dibujará el diagrama de flujo de datos orientado a objetos.

El cuarto apartado se centra en el diseño del sistema. En la fase de análisis lo fundamental es determinar lo qué debe hacerse, sin que sea necesario, ni conveniente comprometerse sobre la forma en qué se hará. Sin embargo, durante el diseño es necesario tomar decisiones sobre la forma en qué se resolverá el problema. El diseño del sistema viene a representar la primera aproximación a este respecto.

El quinto y último apartado está dedicado al diseño de los objetos, que es donde se pasa de la orientación del mundo real propia del modelo de análisis a una orientación más propia del mundo de los ordenadores requerida para llevar a cabo una implementación práctica.

Bibliografía básica

Piattini, Mario G., Calvo-Manzano, José A., Cervera Joaquín, Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 10).

Rumbaugh, J. "*OMT Insights*". SIGS Books Publications, 1996.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998. (Capítulos 7, 8, 9, 10 y 11).

Bibliografía complementaria

Amako, K. "*Object Modeling Technique - Summary Note*". http://arkhp1.kek.jp/managers/computing/activities/OO_CollectInfor/Methodologies/OMT/OMTBook/OMTBook.html. [Última vez visitado, 9-2-2000]. June 1995.

Blaha, M., Premerlani, W. "*Object-Oriented Modeling and Design for Database Applications*". Prentice-Hall, 1998.

Brinkkemper, S., Hong, S., Bulthuis, A., van den Goor, G. "*Object-Oriented Analysis and Design Methods a Comparative Review*". http://wwwis.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html. [Última vez visitado, 9-2-2000]. January 1995.

ICON Computing, Inc. "*A Comparison of OOA & OOD Methods*". Icon Computing Inc. http://www.iconcomp.com/papers/comp/. [Última vez visitado, 9-2-2000]. 1995.

Karma, G. M., Casselman, R. S. "*A Cataloging Framework for Software Development Methods*". IEEE Computer, 26(2):34-45. February 1993.

Premerlani, W. J., Blaha, M. R., Rumbaugh, J. E., Varwing, T. A. "*An Object-Oriented Relational Database*". Communications of the ACM, 33(11):99-109. November 1990.

Rumbaugh, J. "*A Private Workspace. Why a Shared Repository Is Bad for Large Projects*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. September 1995.

Rumbaugh, J. "*Getting Started. Using Use Cases To Capture Requirements*". Journal of Object-Oriented Programming (JOOP), 7(5):8-12, 23. September 1994.

Rumbaugh, J. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming, 7(9):6-12. February 1995.

Rumbaugh, J. "*OMT: The Object Model*". Journal of Object-Oriented Programming, 7(8):21-27. January 1995.

Rumbaugh, J. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March 1994.

Rumbaugh, J. "*The OMT Process*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. May 1994.

Rumbaugh, J. "*What Is a Method?*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. October 1995.

Unidad Didáctica IV: Miscelánea

Objetivo genérico

El objetivo que se plantea esta unidad didáctica es la somera presentación de temas adicionales que, relacionados con la Ingeniería del Software, no han tenido cabida en otras unidades didácticas de mayor envergadura por motivos de tiempo.

Los temas que se traten son introducciones para que el alumno sepa de su existencia y pueda profundizar por su cuenta si lo creyese necesario. Incluso podrían sustituirse las horas reservadas a esta unidad docente por la celebración de seminarios dirigidos por el profesor y desarrollados por los alumnos sobre diferentes temas de interés, o por conferencias invitadas.

Siendo realistas, esta unidad didáctica es muy difícil que se llegue a impartir porque, aún cumpliendo perfectamente los tiempos programados para las otras unidades, todos los cursos académicos surgen imprevistos que obligan a perder alguna clase.

Por este motivo, el 4% del temario teórico reservado para esta unidad didáctica, sólo lo ocupa un tema, en este caso dedicado a las herramientas CASE, por ser un tema que no debiera de faltar en el currículo de un ingeniero técnico en informática [Granger y Little, 1996].

Otros candidatos a ocupar este lugar, o a convertirse en seminarios dentro de esta asignatura, pueden ser *prueba del software, calidad del software, gestión de la configuración, mantenimiento, reutilización, reingeniería* o *aspectos legales del software.*

Tema 9: *Herramientas CASE*

Descriptores

Tecnología CASE, repositorio, metamodelo, metadatos, ICASE, IPSE, gestión de la configuración.

Objetivos

Este tema está orientado a satisfacer el objetivo **P4** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir al alumno en las herramientas CASE como apoyo y automatización al proceso de desarrollo de software.
- Presentar la diferente tipología de herramientas CASE que existen en el mercado y la importancia de su integración para conseguir entornos avanzados de Ingeniería del Software.
- Indicar la importancia del intercambio de datos entre diferentes herramientas CASE para favorecer la integración, reutilización e intercambio de información entre proyectos.
- Explicar la arquitectura que se encuentra detrás de una herramienta CASE.

Contenidos

9.1 Introducción
9.2 Componentes de una herramienta CASE
9.3 Clasificación de las herramientas CASE
9.4 Integración de CASE

Tabla 5.16. Contenidos del tema 9 del programa teórico de Ingeniería del Software

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE3 Software tools and environments** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El tema se ha dividido en cuatro apartados principales, todos ellos tratados de una forma introductoria.

En el primer apartado se introduce la tecnología CASE. Se justifica la existencia de estas herramientas para aumentar la productividad y la calidad del software desarrollado, automatizando en el mayor grado posible el proceso software. Igual que la Informática, el software es un activo en prácticamente la totalidad de los dominios de aplicación o negocios actuales, y debe serlo también para la propia Ingeniería del Software.

Se presenta también en este primer apartado la evolución histórica de la tecnología CASE, desde sus orígenes a mediados de la década de los setenta, su eclosión a mediados de los ochenta, su crisis a finales de los ochenta y principio de la década de los noventa, finalizando con su resurgir con mayor madurez y menos mitología a finales de los noventa.

El segundo apartado se dedica a la presentación de los componentes principales de una herramienta CASE: *repositorio, metamodelo, generador de informes, carga/descarga de datos, comprobación de errores* e *interfaz de usuario*.

En el tercer apartado se estudian diferentes clasificaciones de las herramientas CASE, propuestas por otros tantos autores.

El cuarto y último apartado es quizás el de mayor relevancia, ocupándose de los problemas de integración de las herramientas CASE. Así se estudian los dos modelos de integración ya clásicos: *el modelo cebolla* y *el modelo tostadora* [Piattini y Daryanani, 1995], las características de los entornos ICASE (*Integrated CASE*) y los diferentes niveles de integración de las herramientas CASE [Garbajosa y Bonilla, 1995]:

- ***Integración de datos:*** Mide el grado en que los datos generados por una herramienta se hacen accesibles para su uso en otras herramientas. Cabe destacar algunas iniciativas al respecto: **EIA/CDIF** (*Electronic Industries Association's CASE Data Interchange Format*) [EIA CDIF Division, 1996]; **PCTE** (*Portable Common Tool Environment*) [Aldis, 1995]; **XMI** (*XML Metadata Interchange*) [Brodsky, 1999].
- ***Integración de control:*** Facilidad de las herramientas para comunicarse, cooperar, y la relativa sencillez con la que una nueva herramienta puede hacer uso de servicios ya ofrecidos en lugar de duplicarlos en su propio código.
- ***Integración de presentación:*** Homogeneidad y consistencia de la interfaz de usuario, por ejemplo, que en todas las herramientas se acceda a la ayuda de la misma forma, mismas barras de herramientas, mismo uso del ratón…

Bibliografía básica

Fuggetta, A. "*A Classification of CASE Technology*". IEEE Computer, 26(12):25-38. December 1993.

Mc Clure, C. "*CASE: La Automatización del Software*". Ra-ma, 1992.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 19 y Apéndice A).

Piattini, M., Daryanani, S. N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 31).

Sommerville, I. "*Software Engineering*". 5th Edition. Addison-Wesley, 1996. (Capítulos 25, 26 y 27).

Bibliografía complementaria

Aldis, M. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

Fisher, A. S. "*Tecnología CASE. Herramientas de Desarrollo de Software*". 2ª Edición. Anaya Multimedia, 1993.

García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

Iivari, J. "*Why Are CASE Tools Not Used*". Communications of the ACM, 39(10):94-103. October 1996.

Long, F., Morris, E. "*An Overview of PCTE: A Basis for a Portable Common Tool Environment*". Technical Report CMU/SEI-93-TR-1, ESC-TR-93-175. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., March 1993.

Sharma, S., Rai, A. "*CASE Deployment in IS Organizations*". Communications of the ACM, 43(1):80-88. January 2000.

5.2.2 Programa de la parte práctica

La parte práctica de la asignatura de Ingeniería del Software está orientada para satisfacer aquellos objetivos prácticos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**P1, P2** y **P4**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a] (incluido en el Apéndice A de este Proyecto Docente).

P1	Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.
P2	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
P4	Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Tabla 5.17. Objetivos del programa de prácticas de la asignatura Ingeniería del Software

5.2.2.1 Consideraciones iniciales

El objetivo fundamental de las prácticas de la asignatura de Ingeniería del Software es el modelado de sistemas software. Además, se considera que para aprender a modelar es más efectivo la realización de talleres donde se realice un ejercicio y se discuta su solución, en lugar de hacer más hincapié en el manejo de las herramientas CASE; manejo que por otra parte pueden aprender y entrenarse en horas de práctica libre.

Se considera que las 15 horas, que equivalen al crédito y medio de las prácticas, deben repartirse en prácticas presenciales, talleres fundamentalmente, y prácticas libres, que deben aprovechar para hacer los informes de los talleres y la práctica obligatoria.

Para el mejor aprovechamiento de las prácticas presenciales, éstas se llevan a cabo cada 15 días durante dos horas consecutivas.

Una contrariedad es tener que contar con los conocimientos teóricos necesarios para realizar las prácticas. Esto obliga a que el primer taller debe basarse en conocimientos previos, *modelo entidad-relación*, adquiridos en la asignatura de **Diseño de Bases de Datos**, impartida en el segundo curso. También por restricciones temporales, sólo se podrá realizar un taller dedicado a la tecnología de objetos.

5.2.2.2 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte práctica de esta asignatura se ha organizado en seis prácticas, repartidas en tres grupos, como se muestra en la Tabla 5.18.

Talleres (10 Horas)
Práctica 1. Taller de modelado de datos. Repaso al modelo entidad-relación (2H)
Práctica 2. Taller de modelado funcional de sistemas (método clásico) (2H)
Práctica 3. Taller de modelado funcional de sistemas (método de Yourdon) (4H)
Práctica 4. Taller de Orientación al Objeto (2H)
Laboratorio (2 Horas)
Práctica 5. Manejo de una herramienta CASE (2H)
Práctica obligatoria (Prácticas libres)
Práctica 6. Realización de una ERS y un prototipo de una aplicación software

Tabla 5.18. Estructura del programa de prácticas de la asignatura Ingeniería del Software (1,5 créditos)

En la Tabla 5.19 se presenta la correspondencia existente entre el programa de prácticas y los objetivos prácticos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Práctica 1	P1, H1, H2, H4
Práctica 2	P1, H1, H2, H4
Práctica 3	P1, H1, H2, H4
Práctica 4	P2, H1, H2, H4
Práctica 5	P4
Práctica 6	P1, P2, P4, H1, H2, H4

Tabla 5.19. Correspondencia entre el temario de prácticas y los objetivos prácticos de la asignatura

Desarrollo de las clases prácticas (talleres)

Los talleres se llevan a cabo en el aula de teoría. Los alumnos se organizan en grupos de entre cuatro y seis personas, y durante, aproximadamente una hora, de las dos de las que disponen, el grupo discute y desarrolla la solución a un problema cuyo enunciado, previamente, ha sido facilitado a través de la página de la asignatura.

Uno de los grupos se ofrecerá voluntario para presentar la práctica, recibiendo como recompensa 0,75 puntos, sumados a la nota final.

Durante la segunda hora, el grupo desarrolla su solución en la pizarra (si la solución del problema tuviera una gran cantidad de diagramas, se dedican las dos horas para su solución, y se aplaza la discusión para la siguiente clase, utilizándose transparencias).

Tras su exposición, se desarrolla un debate con el resto de los grupos, moderado por el profesor. De esta manera se logra, por una parte, que los alumnos se expresen en público y defiendan su trabajo y, por otra, la corrección pública de los errores cometidos en los modelos.

Con los comentarios, el grupo encargado de la defensa elabora un informe que entrega al profesor, quien, tras comprobar que no contiene errores, lo publicará en la página de la asignatura, sirviendo de material de estudio al resto de los alumnos, así como para promociones futuras.

Evaluación de la parte práctica

La forma principal de evaluar la parte práctica de esta asignatura es mediante la realización de una práctica obligatoria, cuyos requisitos deben obtenerse de *usuarios* o *clientes* reales.

En la Figura 5.7 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota de la práctica obligatoria y los informes de los talleres, realizados voluntariamente.

Si (Teoría ≥ 4,75) **y** (***Práctica*** ≥ 5.0)

Nota Final = (Teoría*0,5) + (***Práctica****0,5) + ***Nota trabajos***

Sino

Ø

Fin si

Figura 5.7. Influencia de la nota en la parte práctica en la nota final de Ingeniería del Software

Bibliografía básica de referencia

De la lista de títulos recomendados, los siguientes pueden ser los más adecuados para ser consultados en la realización de la parte práctica.

- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **Durán, A., Bernárdez, B.** "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.
- **OMG.** "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.

- **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

A los que habría que añadir alguno donde se aborde la creación de modelos entidad-interrelación, como por ejemplo:

- **Date, C. J.** "*Introducción a los Sistemas de Bases de Datos*". 7ª Edición, Addison-Wesley, 2001.
- **Miguel, A. de, Piattini, M.** "*Concepción y Diseño de Bases de Datos. Del Modelo E/R al Modelo Relacional*". Ra-ma, 1993.
- **Miguel, A. de, Piattini, M.** "*Fundamentos y Modelos de Bases de Datos*". Ra-ma, 1997.

5.2.2.3 Desarrollo comentado del programa

En el programa de práctica se distinguen tres bloques. El primero está compuesto por los talleres, donde se va a hacer hincapié en los aspectos de modelado. El segundo sería el laboratorio, donde se puede aprovechar para introducir alguna herramienta CASE, ya sea orientada al paradigma estructurado, como **Easy Case** [Evergreen, 1994], u orientada al paradigma objetual, como **Rational Rose** (http://www.rational.com). El tercero, y último, es el que se ocupa de la práctica obligatoria, por lo tanto práctica no presencial.

Talleres

Los talleres constituyen la parte práctica de la asignatura donde la relación docente-discentes es más importante.

Los objetivos generales de estos talleres se pueden resumir en los siguientes puntos:

- Practicar el modelado de sistemas software tanto en el paradigma estructurado como en el objetual.
- Comentar en público los errores más frecuentes que se cometen a la hora de realizar los modelos.
- Potenciar la comunicación en público de los alumnos.
- Obligar a la realización de informes que documenten su labor.
- Incentivar la participación del alumno en el desarrollo de la asignatura.

Idealmente cada alumno matriculado en la asignatura participará en tres talleres de dos horas de duración cada uno, más en un cuarto de cuatro horas de duración; lo que supone ocupar

aproximadamente el **67%** del tiempo oficial de prácticas en estas actividades. Los talleres a realizar son:

- ***Taller 1:*** Modelo de información – Entidad-Interrelación.
- ***Taller 2:*** Modelo funcional – Enfoque clásico.
- ***Taller 3:*** Modelo funcional – Enfoque de Yourdon.
- ***Taller 4:*** Diagramas de clases.

A continuación se va a desarrollar de una forma más detallada cada uno de ellos.

***Taller 1:* Modelo de información**

Es el primero que se realiza. Sirve como repaso a las técnicas de modelado conceptual de bases de datos relacionales impartidas en la asignatura de **Diseño de Bases de Datos** del segundo curso.

Permite al profesor realizar prácticas mientras se avanza en el temario teórico para poder abordar las prácticas relacionadas con el modelado funcional mediante DFDs.

La técnica de modelado utilizada en este taller es el modelo entidad-interrelación, aunque además del modelo conceptual, se les pide que construyan el modelo lógico (modelo relacional) equivalente al modelo conceptual elaborado, de manera que éste se encuentre en forma normal de Boyce/Codd.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

***Taller 2:* Modelo funcional – Enfoque clásico**

En este segundo taller se pretende que los alumnos se familiaricen con la creación de DFDs por niveles, su balanceo y la identificación de las estructuras inválidas de esta técnica.

Este primer taller dedicado al modelado funcional, se realiza utilizando un enfoque clásico, totalmente descendente, para lo que se requiere un problema relativamente pequeño.

Se justifica un taller de estas características por los siguientes motivos:

- El objetivo fundamental es que el alumno se familiarice con la técnica.
- El alumno está acostumbrado a trabajar con técnicas descendentes.
- Es una buena oportunidad para comentar los problemas del modelado descendente en la creación de DFDs.

El informe final a realizar deberá incluir además el diccionario de datos y la especificación de los procesos primitivos.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Taller 3: **Modelo funcional – Enfoque de Yourdon**

El tercer taller también se dedica al modelado funcional, pero ahora haciendo uso del método de Yourdon [Yourdon, 1989; Yourdon Inc., 1993].

Se pretende que el grupo que defienda la práctica muestre la evolución desde el DFD preliminar hasta el DFD por niveles final. Como esto supone realizar muchos diagramas, y el nivelado ascendente/descendente de este método les resulta más complejo debido a su dependencia de las técnicas descendentes, a este taller se le dedican cuatro horas, dos para la elaboración y dos para la presentación y la discusión.

El informe final a realizar debe incluir además el diccionario de datos y la especificación de los procesos primitivos.

Este taller busca satisfacer el objetivo **P1**, *aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Taller 4: **Diagramas de clases**

El cuarto y último taller se dedica a la orientación a objetos. Por restricciones temporales, cuando se estudian las técnicas orientadas al objeto, UML, queda muy poco tiempo para finalizar la asignatura y, por tanto, para llevar a cabo el taller. Así, hay que decantarse por una técnica, y quizás la más representativa sea el diagrama de clases de UML.

Lo más interesante de esta práctica es que, partiendo de un enunciado pequeño que represente un caso conocido por los alumnos, se vayan identificando las clases, relacionándolas y refinándolas, hasta un punto no demasiado elaborado, pero lo suficiente para que capten la filosofía de trabajo.

El informe final debe incluir la documentación de cada clase y cada relación, utilizando para ello las plantillas recomendadas en [Durán y Bernárdez, 2001a].

Este taller busca satisfacer el objetivo **P2**, *aplicar de forma práctica los conceptos teóricos de Orientación a Objetos*, identificado en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

Laboratorio

De cara a satisfacer el objetivo **P4** de la Unidad Docente de Ingeniería del Software y Orientación a Objetos, *utilización de herramientas CASE para la gestión y desarrollo de sistemas software*, se puede dedicar una jornada de prácticas, aproximadamente el 13% de la parte de prácticas, para introducir alguna herramienta CASE, ya sea dedicada al paradigma estructurado, cada vez más escasas y con licencias muy costosas, como **Easy CASE** [Evergreen, 1994], o dedicadas al paradigma objetual, concretamente a UML, muy abundantes en Internet, ya sea por ser de libre distribución como **ArgoUML** (http://argouml.tigris.org/), o versiones reducidas, como puede ser el caso de **Rational Rose 98** o **Rational Rose 2001** (http://www.rational.com).

Precisamente, para solucionar el problema de las licencias, desde el curso académico 2000-2001, se viene trabajando con herramientas CASE construidas en el Departamento de Informática y Automática de la Universidad de Salamanca, bajo la dirección del que suscribe este Proyecto Docente e Investigador, trabajo que ha estado subvencionado por la Consejería de Educación y Cultura de la Junta de Castilla y León [García, 2001].

Concretamente las herramientas que se han construido en el Departamento (disponibles en la URL http://tejo.usal.es/~fgarcia/docencia/isoftware/case/casetools.html), y que se utilizan en las asignaturas de Ingeniería del Software y en los proyectos de final de carrera son:

- **ADAM CASE v1.1 [García et al., 2000b; García et al., 2001b]:** Herramienta CASE frontal para plataformas **Windows 9x**, **Windows NT** o **Windows 2000**, que facilita el desarrollo de proyectos estructurados y orientados a objetos (utilizando la notación UML). Inicialmente, ADAMCASE soporta un tipo de técnica de modelado para cada tipo de proyecto:
  - Diagramas de Flujo de Datos (DFD), siguiendo la notación propuesta por Edward Yourdon [Yourdon, 1989] en proyectos estructurados.
  - Diagramas de clases, siguiendo la notación propuesta por UML [OMG, 2001c], en proyectos orientados a objetos
- **ER CASE v1.1 [Blasco, 2000]:** Herramienta CASE frontal para plataformas **Windows 9x**, **Windows NT** o **Windows 2000**, que permite la creación de Diagramas Entidad Relación (DER), utilizando la notación de Peter Chen [Chen, 1976], así como su posterior paso a un modelo lógico de datos basado en el modelo relacional.

- **CRC CASE v1.0 [Cuesta, 2001]:** Herramienta CASE frontal cuyo cometido consiste en asistir al desarrollador en una de las tareas más críticas, el descubrimiento de clases.
- **Left CASE v1.0 [García et al., 2001a]:** Herramienta CASE frontal para plataformas GNOME (http://www.gnome.org). Left CASE (http://zarza.usal.es) es un entorno CASE extensible, basado en componentes, de forma que existe un *framework* base o contenedor, que hace las veces de "anfitrión" para los componentes, cada uno de los cuales dará soporte a una técnica diferente de modelado. En la actualidad se dispone de los siguientes componentes:
    - **Left CASE: Componente UML v1.2 [Hernández, 2001]**: Para la realización de diagramas de clase de UML.
    - **Left CASE: Componente DTE v1.0 [Costa, 2001]**: Para la realización de diagramas de transición de estados, siguiendo la notación de Yourdon [Yourdon, 1989].
    - **Left CASE: Componente DER v1.0 [González, 2001]**: Para la realización de diagramas entidad relación extendidos, así como su transformación al correspondiente modelo lógico relacional.
    - **Left CASE: Componente DFD v1.0 [Conde, 2002]**: Para la realización de diagramas de flujo de datos, siguiendo la notación de Yourdon [Yourdon, 1989].

Además, también se trabaja con la herramienta REM (*REquirements Management*) [Durán et al., 2002], definida en la Tesis Doctoral de Amador Durán Toro [Durán, 2000], y que amablemente ha puesto al alcance de toda la comunidad en [Durán, 2002].

Práctica obligatoria

Es la base fundamental para la evaluación de la parte práctica de la asignatura de Ingeniería del Software.

En las primeras semanas del curso se publica en la página de la asignatura en qué consiste y la fecha de entrega.

La práctica debe hacerse en grupos, compuestos por entre tres y cinco alumnos, y ellos mismos deben buscar un *cliente real* a quién hacerle las entrevistas oportunas para la obtención de los requisitos de la práctica.

Esta forma de plantear la práctica obligatoria tiene las siguientes ventajas:

- Se obliga a que los alumnos se acerquen a lo que es el mundo real [Nachbar, 1998]. Deben poner en práctica las técnicas de entrevista, comprobando la dificultad que supone la comunicación con los clientes.
- Evita el plagio de prácticas dado que cada grupo tiene un cliente distinto.
- Potencia el trabajo en grupo. Se les da libertad para que ellos se organicen.
- Se les da libertad para elegir entre el paradigma estructurado o el objetual.

Como inconvenientes se pueden citar:

- Normalmente, el cliente es un familiar de alguno de los miembros del grupo, con lo que la entrevista suele ser realizada por una persona sólo, en lugar de participar todo el grupo.
- Debe ponerse un límite para que el grupo no minimice los requisitos a simples altas, bajas y modificaciones de unas entidades.
- Todos los inconvenientes que tiene el trabajo en grupo, cuando se hace un reparto del problema y no se trabaja en grupo, o cuando alguien decide *camuflarse* en el grupo y dejar que trabajen por él. Pero estos son problemas reales, que ellos mismos deben aprender a sortearlos.

Los productos que cada grupo debe entregar son dos:

1. *Un documento de* ***especificación de requisitos:*** El documento que recoja la especificación de requisitos puede basarse en la estructura de cualquiera de los formatos vistos en la parte teórica de la asignatura, debiendo incluir de forma obligatoria los siguientes aspectos:
    - Una descripción textual describiendo los requisitos (catálogo de requisitos).
    - Modelo funcional.
    - Modelo de datos.
    - Modelo de comportamiento dinámico (si ha lugar).
    - Modelo lógico de datos (modelo relacional en FNBC).
2. *Un* ***prototipo:*** Se debe realizar un prototipo de la aplicación que recoja toda la interfaz de usuario de la aplicación y un mínimo de la funcionalidad especificada. La funcionalidad implementada, así como su complejidad, influirá en la evaluación

de la práctica. El prototipo se realizará utilizando el entorno de desarrollo visual que el grupo de trabajo estime oportuno, con la única restricción de que el prototipo se pueda ejecutar bajo **Windows NT** o **Linux**.

Cada grupo deberá entregar todos los ficheros de su trabajo organizados en documentación y ejecutables, así como la especificación de requisitos impresa. Después de la entrega se publicará un calendario para la defensa por grupos de la práctica.

La defensa de la práctica se realizará en un examen oral, donde los miembros del grupo dispondrán de unos 5 minutos para presentar el prototipo, relacionándolo con el catálogo de requisitos establecido, su viabilidad... a semejanza de una reunión con un cliente que tiene que validar la especificación realizada. Durante aproximadamente otros 15 minutos los integrantes del grupo contestarán a preguntas, sobre las características técnicas de los modelos, realizadas por el responsable de la asignatura.

Al ser un trabajo realizado en grupo, todos los integrantes del mismo recibirán la misma nota. Esto significa que la actuación individual de cada integrante repercutirá en el global del grupo.

Con el fin de promover una mayor motivación hacia el trabajo, y por transitividad hacia la asignatura, la nota final del trabajo dependerá de un baremo impuesto en función de la calidad técnica y de la presentación de los trabajos. De todas formas, con independencia de la nota final, para que la práctica se considere superada se hará uso de la siguiente fórmula:

$$(NotaTécnica*0,8)+(Prototipo*0,1)+(PresentaciónOral*0,05)+(PresentaciónDocumentación*0,05) \geq 5$$

Para la realización de la práctica, se les da libertad sobre los métodos a seguir, siempre y cuando los resultados sean correctos.

Se hace hincapié en la utilización de estándares para realizar los documentos, aunque hasta ahora se les ha dado libertad para elegir los estándares a seguir, en futuros cursos se irán desarrollando guías de estilo y contenido para la asignatura, semejantes a las de [Marqués, 1999] o [Tuya, 2001].

En general, con esta práctica se busca satisfacer en general todos los objetivos prácticos identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos.

De forma más concreta se podían citar los siguientes objetivos específicos:

- Que los alumnos se enfrenten a la especificación (con uso de prototipos) de un caso real.
- Que los alumnos entiendan la diferencia entre un programa y un producto software.

- Que los alumnos pongan en práctica las técnicas explicadas en la parte teórica y en los talleres de la asignatura.
- Que los alumnos participen en un trabajo en grupo, donde hay roles diferenciados, y además debe existir una comunicación intensa entre todos ellos.
- Que los alumnos se acostumbren a documentar el software realizado, cogiendo soltura en la redacción de documentos técnicos.
- Que los alumnos manejen bibliografía especializada para profundizar en los aspectos teóricos que dan soporte a la práctica.
- Que los alumnos desarrollen y mejoren su capacidad de comunicación entre compañeros y ante un cliente real.

5.3 Programa de la asignatura Análisis de Sistemas

La asignatura *Análisis de Sistemas* es una de las dos asignaturas que recogen los créditos troncales de la materia en la titulación de Ingeniero Informático (2º ciclo), junto a la asignatura *Administración de Proyectos Informáticos*. Su propósito fundamental es que los alumnos de esta titulación profundicen en los conocimientos adquiridos en la asignatura *Ingeniería del Software* de primer ciclo, especialmente en lo tocante a métodos de análisis y especificación de requisitos de sistemas orientados a objetos y a técnicas formales de especificación.

Asignatura	Análisis de Sistemas (troncal)
Créditos	6T + 3P
Estudios	Ingeniería Informática (2º ciclo)
Plan	B.O.E de 1-7-1999
Curso	1º
Cuatrimestre	1º y 2º (anual)
Responsables	Francisco José García Peñalvo (fgarcia@usal.es) María N. Moreno García (mmg@usal.es)
Página web de la asignatura	http://lisisu02.usal.es/~mmoreno/analisis.html

Tabla 5.20. Ficha de la asignatura Análisis de Sistemas

Esta asignatura se imparte durante los dos primeros cuatrimestres del primer curso, dentro del Plan de Estudios del B.O.E de 1-7-1999 [BOE, 1999].

Esta asignatura está dotada de ***9 créditos, 6 teóricos*** y ***3 prácticos***. Se orienta como una consolidación y ampliación de los contenidos impartidos en la asignatura *Ingeniería del Software* de la Ingeniería Técnica en Informática de Sistemas, centrándose en los aspectos relacionados con la especificación de requisitos.

Para la elaboración del programa se ha optado por una estrategia sustentada en los siguientes puntos:

1. Se tiene como prerrequisito que el alumno tenga unos conocimientos teóricos y prácticos de los fundamentos de la Ingeniería del Software, de los modelos de ciclo vida del software más difundidos y de los métodos de análisis y diseño estructurado. Sería muy conveniente que el alumno también estuviera familiarizado con las bases del desarrollo del software orientado a objetos.
2. En el caso de la Universidad de Salamanca, los prerrequisitos se cumplen para los alumnos que habiendo obtenido el título en Ingeniería Técnica en Informática de Sistemas en el Plan de Estudios de 1997 [BOE, 1997], decidan cursar los estudios

de segundo ciclo en esta Universidad. Esto es así, porque dicho Plan de Estudios cuenta con la asignatura obligatoria de *Ingeniería del Software* y la asignatura optativa *Programación Orientada a Objetos*, que aseguran los prerrequisitos de esta asignatura.

3. La titulación de Ingeniería Informática comenzó su andadura en la Universidad de Salamanca en el curso académico 1998-1999. Notándose una disparidad elevada en la procedencia de los alumnos. Se tienen alumnos de otras Universidades, aunque la mayoría de ellos habían cursado el primer ciclo en la Universidad de Salamanca, pero en Planes de Estudios anteriores al Plan de Estudios de 1997 (los primeros alumnos que habían cursado este Plan se incorporan al segundo ciclo en el curso 2000-2001, no siendo mayoría hasta el curso académico 2001-2002).

   Esta característica influye en el temario de la asignatura, de forma que para garantizar una uniformidad en la base de conocimientos del alumnado, se diseña con un gran solapamiento con respecto al temario de la asignatura *Ingeniería del Software*, de primer ciclo.

4. El solapamiento antes mencionado va reduciéndose en cada nuevo curso académico, a la par que el número de alumnos procedentes del Plan de Estudios de 1997 de la Ingeniería Técnica en Informática de la Universidad de Salamanca va creciendo. Esto provoca que el temario de la asignatura *Análisis de Sistemas* sufra constantes modificaciones, más en los contenidos que en los temas, aunque como se puede comprobar existen diferencias entre los temas que actualmente se imparten, y que se presentarán a continuación en este Proyecto Docente, y los que inicialmente se propusieron y que quedaron recogidos en el Plan de Calidad de la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a], que se incluye en el Apéndice A de este documento.

5. Aunque los métodos estructurados están presentes en el temario de la asignatura, el énfasis de la asignatura se pone los métodos de desarrollo orientado a objetos, porque se tiene que la demanda es creciente en titulados que posean estos conocimientos [Tewari, 1995], y porque los alumnos ya poseen (o debieran poseer unos conocimientos más sólidos en el manejo de métodos estructurados).

6. Otro apartado importante en el temario de la asignatura lo ocupan los métodos de especificación formal, para que los ingenieros informáticos puedan integrar estos conocimientos con otros productos y procesos software a lo largo del ciclo de vida [Lamsweerde, 2000].

7. Aunque la asignatura se fundamenta en la transmisión de conocimiento técnico, no se puede (ni se quiere) perder la oportunidad de hacer que los alumnos desarrollen y potencien otras habilidades más generales, pero de importancia capital en su futuro como profesionales: *acostumbrarse a consultar bibliografía (especialmente*

en inglés), haciendo hincapié en la importancia que tiene leer con cuidado para sintetizar, escribir y modelar de forma adecuada [Jackson, 1995]; *escribir documentos técnicos que describan los diferentes elementos software que se crean a lo largo de un proyecto* [Deveaux et al., 1999] *cuidando los estándares de documentación* [Gersting, 1994; McCauley et al., 1996], *sin que ello signifique dar la espalda a las reglas gramaticales y de estilo que ofrece un lenguaje tan rico como el castellano* [Vaquero, 1999]; *desarrollar una capacidad de comunicación oral adecuada* [McDonald y McDonald, 1993; Fell et al., 1996].

8. Llegar a un equilibrio entre la teoría y la práctica [Glass, 1996], de forma que una base teórica bien establecida sea el fundamento adecuado para la aplicación práctica de la Ingeniería del Software.

5.3.1 Programa de la parte teórica

La parte teórica de la asignatura *Análisis de Sistemas* está orientada a satisfacer aquellos objetivos teóricos que, habiendo sido identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y al contexto docente de la asignatura (**T3, T4, T5, T6, T7, T12** y **T13**), además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a].

T3	Importancia de los requisitos en el ciclo de vida del software.
T4	Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
T5	Especificaciones formales de requisitos.
T6	Método de análisis /diseño estructurado.
T7	Método de análisis /diseño orientado a objetos.
T12	Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.
T13	Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

Tabla 5.21. Objetivos del programa de teoría de la asignatura Análisis de Sistemas

5.3.1.1 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de diez temas, organizados en cuatro unidades docentes, tal y como se puede apreciar en la Tabla 5.22

La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve presentación del temario de teoría y práctica de la misma, relacionando dichos programas con la perspectiva de la formación de un Ingeniero Informático. Los alumnos ya cuentan con el programa de la asignatura, resumido en la guía académica que se les entrega con la matrícula [GAFC-USAL, 2001], y totalmente actualizado en la página web de la asignatura (http://lisisu02.usal.es/~mmoreno/analisis.html).

Presentación de la asignatura (1 Hora)
Unidad Didáctica I: Definición del proceso de software (7 Horas)
Tema 1. Sistemas de Información (1 Hora)
Tema 2. Modelos avanzados de ciclo de vida (4 Horas)
Tema 3. Prototipos (2 Horas)
Unidad Didáctica II: Métodos de desarrollo (41 Horas)
Tema 4. Métodos estructurados. Técnicas avanzadas (4 Horas)
Tema 5: Metodología MÉTRICA V.3 (5 Horas)
Tema 6. Métodos orientados a objetos (4 Horas)
Tema 7. El lenguaje UML y el proceso unificado (18 Horas)
Tema 8. Técnicas formales de especificación (10 Horas)
Unidad Didáctica III: Desarrollo de sistemas complejos (8 Horas)
Tema 9. Desarrollo de sistemas complejos (8 Horas)
Unidad Didáctica IV: Evolución del software (3 Horas)
Tema 10. Evolución y mantenimiento del software (3 Horas)

Tabla 5.22. Estructura del programa de teoría de la asignatura Análisis de Sistemas

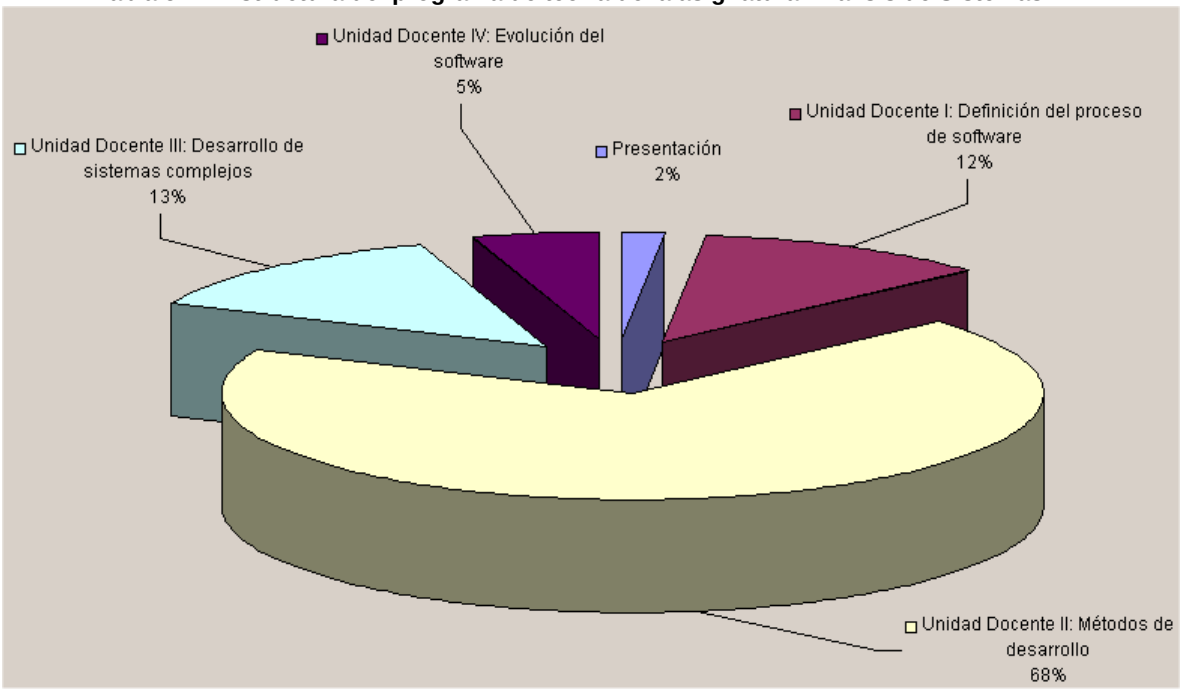


Figura 5.8. Reparto porcentual de las horas de teoría de Análisis de Sistemas entre las unidades didácticas

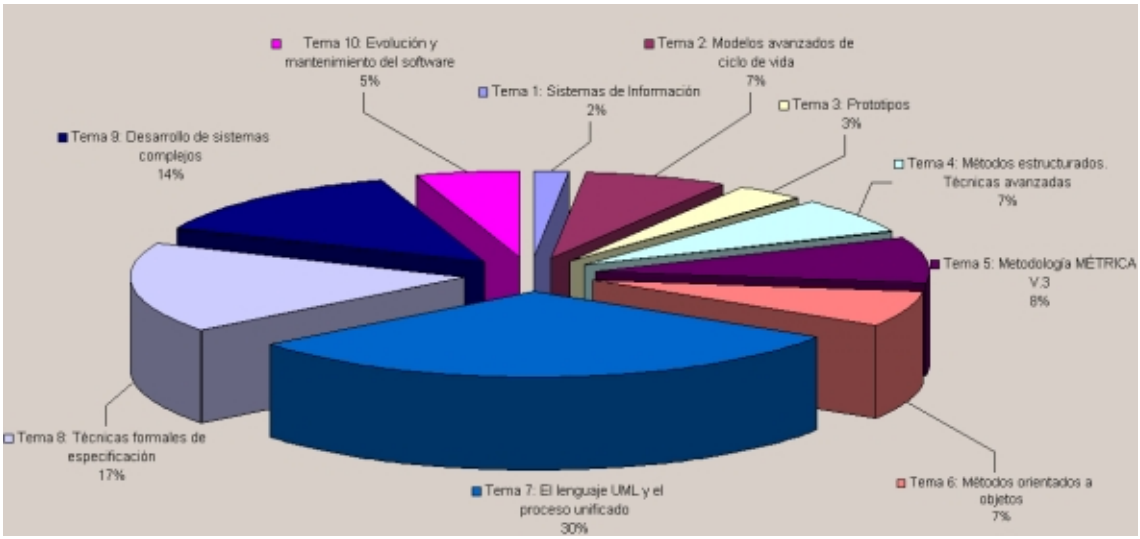


Figura 5.9. Reparto de las horas de teoría de Análisis de Sistemas entre los temas

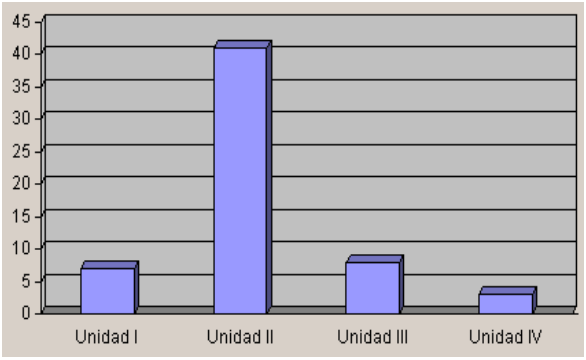


Figura 5.10. Distribución temporal por unidades didácticas

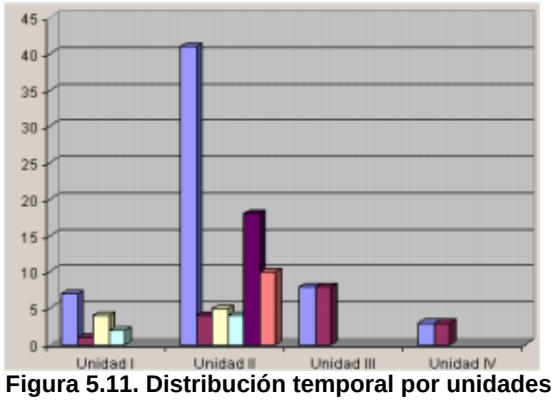
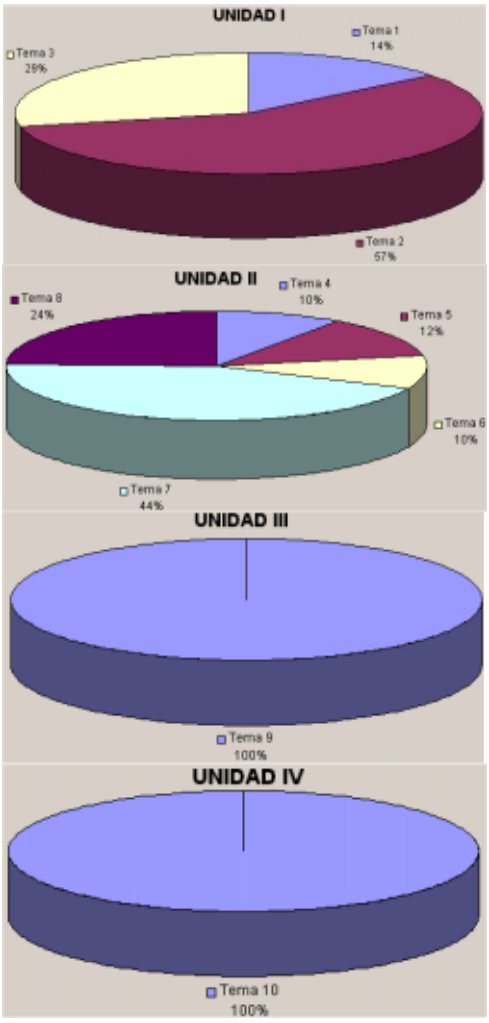


Figura 5.11. Distribución temporal por unidades didácticas y temas de cada una de ellas

En la Figura 5.8 se presenta el reparto porcentual de las unidades didácticas de la asignatura *Análisis de Sistemas*, mientras que la Figura 5.9 refleja el reparto porcentual de los diferentes temas. La Figura 5.10 refleja cuantitativamente el número de horas dedicadas a cada una de las unidades didácticas y la Figura 5.11 presenta la distribución en horas de los temas de cada unidad didáctica, comparándolo con la duración total de la unidad didáctica a la que pertenece (primera barra de cada unidad). A continuación se presenta porcentualmente la presencia de cada tema dentro de su unidad didáctica.



Tema 1. Sistemas de Información (1 H)
Tema 2. Modelos avanzados de ciclo de vida (4 H)
Tema 3. Prototipos (2 H)

Tema 4. Métodos estructurados. T. avanzadas (4 H)
Tema 5: Metodología MÉTRICA V.3 (5 H)
Tema 6. Métodos orientados a objetos (4 H)
Tema 7. El lenguaje UML. Proceso unificado (18 H)
Tema 8. Técnicas formales de especificación (10 H)

Tema 9. Desarrollo de sistemas complejos (8 H)

Tema 10. Evolución y mantenimiento del sw (3 H)

En la Tabla 5.23 se presenta la correspondencia existente entre el programa de teoría y los objetivos teóricos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T3
Tema 2	T3, T4, H3
Tema 3	T3, T4, H3
Tema 4	T6, H3
Tema 5	T6, T7, H3
Tema 6	T7, H3
Tema 7	T7, H3
Tema 8	T5, H3
Tema 9	T13, H3
Tema 10	T12, H3

Tabla 5.23. Correspondencia entre el temario teórico y los objetivos teóricos de la asignatura

Estos temas pueden verse completados por otras actividades complementarias, que se llevarán a cabo dependiendo de diversos factores, entre los que cabe citar *la disponibilidad de tiempo para hacerlas efectivas* y *el interés y colaboración de los propios alumnos*. Las actividades a realizar pueden caer dentro de alguno de los siguientes grupos:

- Seminarios impartidos sobre temas específicos.
- Trabajos voluntarios realizados por los alumnos.
- Conferencias invitadas.
- *Workshop* de trabajos realizados por los alumnos sobre temas de la asignatura.

Desarrollo de las clases de teoría

Las clases de teoría se desarrollan de forma similar a las de la asignatura *Ingeniería del Software*, esto es, utilizando una variante de la clase magistral, donde el profesor se apoya en un retroproyector y en la pizarra para el desarrollo de los siete temas de los que se compone el temario.

Los alumnos cuentan de antemano con las transparencias de los temas para que no tengan que tomar apuntes en el sentido clásico del término, y puedan prestar atención a las explicaciones, completando las transparencias con las notas que cada uno crea oportuno. Además, permite que el alumno que lo desee intervenga en cualquier momento para hacer una pregunta o solventar una duda y no, como en el dictado de apuntes, para pedir que se repita una frase.

Evaluación de la parte teórica

La forma principal de evaluar la parte teórica de esta asignatura es mediante la realización de una prueba escrita. En la Figura 5.12 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura.

Si (***Teoría*** ≥ 4,75) **y** (Práctica ≥ 5.0)
    Nota Final = ((***Teoría****0,6) + (Práctica*0,3)) * 10/9 + Nota trabajos
Sino
    Ø
Fin si

Figura 5.12. Influencia de la nota de la parte teórica en la nota final de Análisis de Sistemas

La parte de teoría se evalúa mediante un examen escrito, formado por cuestiones de respuesta corta y pequeños supuestos prácticos, haciendo un solo examen por convocatoria, esto

es, un examen final en el mes de junio y un examen final extraordinario en el mes de septiembre.

Los trabajos voluntarios presentados obtendrán una puntuación entre 0,5 y 1,5 puntos, que se sumarán a la nota conseguida en los apartados de teoría y de prácticas, siempre y cuando en éstos se haya obtenido la calificación mínima exigida.

Bibliografía básica de referencia

La lista de títulos que se les propone como bibliografía básica de consulta es:

- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **Harry, A.** "*Formal Methods. Fact File*". John Wiley & Sons, 1996.
- **Jacobson, I., Booch, G., Rumbaugh, J.** "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000.
- **Ministerio de Administraciones Públicas.** "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.
- **OMG.** "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.
- **Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.
- **Piattini, M., Villalba, J., Ruiz, F., Bastanchury, T., Polo, M., Martínez, M. Á., Nistal, C.** "*Mantenimiento del Software. Modelos, Técnicas y Métodos para la Gestión del Cambio*". Ra-ma, 2000.
- **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.
- **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.
- **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.
- **Silva, M.** "*Las Redes de Petri en la Automática y la Informática*". Editorial AC, 1985.
- **Sommerville, I.** "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002.
- **Warmer, J., Kleppe, A.** "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley, 1999.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993.

5.3.1.2 Desarrollo comentado del programa de teoría

El Plan de Estudios de Ingeniero en Informática (segundo ciclo) de la Universidad de Salamanca, publicado en el BOE número 156 de 1 de julio de 1999 [BOE, 1999] contiene la siguiente descripción de los contenidos de la asignatura *Análisis de Sistemas*:

> *Análisis y definición de requisitos. Diseño, propiedades y mantenimiento del software. Análisis de aplicaciones.*

En este epígrafe se va a desarrollar con mayor grado de detalle el programa de la parte teórica de la asignatura *Análisis de Sistemas*. Este programa se ha dividido en cuatro partes o unidades didácticas. Dicho programa se desarrolla a lo largo de dos cuatrimestres, en el primero sólo se imparte teoría, durante tres horas a la semana, mientras que en el segundo se imparte una sola hora de teoría y dos de prácticas.

En la primera unidad didáctica, **Definición del proceso de software**, se parte de que el alumno ya ha sido introducido a la Ingeniería del Software cursada en la Ingeniería Técnica en Informática de Sistemas. El objetivo de esta unidad docente se convierte, entonces, en recordar y ampliar los conceptos sobre el de proceso software, así como en recalcar la importancia de los modelos de ciclo de vida.

La segunda unidad didáctica, **Métodos de desarrollo**, profundiza en los métodos de desarrollo software, sucintamente se recordarán los principios que comparten los métodos estructurados y los orientados a objetos. Aunque los puntos más destacados de esta unidad son la metodología Métrica versión 3 [MAP, 2001], UML [OMG, 2001c] y los métodos formales.

La tercera unidad didáctica, **Desarrollo de sistemas complejos**, introduce cómo aplicar los principios de Ingeniería del Software para la definición y desarrollo de sistemas software complejos, tales como sistemas de tiempo real, sistemas web o sistemas para el soporte a la toma de decisiones.

La cuarta, y última, unidad didáctica, **Evolución del software**, presenta la problemática del mantenimiento y la evolución de los sistemas software.

Es obvio que, debido a los créditos asignados a la asignatura, no es posible extenderse en todas los temas por igual. En algunos casos la explicación se verá reducida a una breve referencia o comentario, mientras que en otros, que se consideran más representativos y de mayor actualidad, se profundiza más en las explicaciones.

Igual que en la descripción del temario de Ingeniería del Software, cada parte está dividida en una serie de temas, subtemas y epígrafes; donde todos los temas van a ser descritos siguiendo

el patrón de documentación compuesto por las entradas: *título, descriptores, objetivos, contenido, resumen* y *bibliografía*.

Unidad Didáctica I: Definición del proceso de software (7 Horas)

Tema 1. Sistemas de Información (1 Hora)

1.1 Historia y evolución (10 Minutos)

1.2 Elementos de un sistema de información (10 Minutos)

1.3 Estructura de un sistema de información (10 Minutos)

1.4 Tipos de sistemas de información automatizados (20 Min.)

1.5 Principios generales de sistemas (10 Minutos)

Tema 2. Modelos avanzados de ciclo de vida (4 Horas)

2.1 Concepto de ciclo de vida (10 Minutos)

2.2 Procesos del ciclo de vida (20 Minutos)

2.2.1 Procesos principales

2.2.2 Procesos de soporte

2.2.3 Procesos de la organización

2.3 Modelos de ciclo de vida (3 Horas y 30 Minutos)

2.3.1 Modelo clásico

2.3.2 Variantes del ciclo de vida iterativo

2.3.3 Ciclo de vida en espiral

2.3.4 Modelo de métodos formales

2.3.5 Desarrollo rápido de aplicaciones

2.3.6 Modelo de desarrollo concurrente

2.3.7 Modelos para sistemas orientados a objetos

Tema 3. Prototipos (2 Horas)

3.1 ¿Qué es un prototipo? (10 Minutos)

3.2 Enfoques en la construcción de prototipos (40 Minutos)

3.3 Tipos de prototipos (10 Minutos)

3.4 Usos de los prototipos (15 Minutos)

3.5 Guías para el desarrollo de prototipos (20 Minutos)

3.6 Beneficios de los prototipos (10 Minutos)

3.7 Herramientas para la construcción de prototipos (15 Minutos)

Unidad Didáctica II: Métodos de desarrollo (41 Horas)

Tema 4. Métodos estructurados. Técnicas avanzadas (4 Horas)

4.1 Introducción (20 Minutos)

4.2 Perspectiva histórica (20 Minutos)

4.3 Clasificación de los métodos (20 Minutos)

4.3.1 Métodos orientados a procesos

4.3.2 Métodos orientados a datos

4.4 Principales métodos de desarrollo (1 Hora)

4.4.1 DeMarco
4.4.2 Gane y Sarson
4.4.3 Yourdon y Constantine
4.4.4 MERISE
4.4.5 SSADM
4.4.6 MÉTRICA

4.5 Técnicas de modelado (2 Horas)

4.5.1 Diccionario de datos
4.5.2 Modelado funcional
4.5.3 Modelado de datos
4.5.4 Modelado de comportamiento

Tema 5. Metodología MÉTRICA V.3 (5 Horas)

5.1 Introducción (15 Minutos)
5.2 Estructura de la metodología (15 Minutos)
5.3 Planificación de Sistemas de Información (30 Minutos)
5.4 Desarrollo de Sistemas de Información (3 Horas)

5.4.1 Estudio de la viabilidad del sistema
5.4.2 Análisis del Sistema de Información
5.4.3 Diseño del Sistema de Información
5.4.4 Construcción del Sistema de Información
5.4.5 Implantación y Aceptación del Sistema

5.5 Mantenimiento de Sistemas de Información (30 Minutos)
5.6 Procesos de Interfaz (30 Minutos)

Tema 6. Métodos orientados a objetos (4 Horas)

6.1 El paradigma de la orientación al objeto (2 Horas)

6.1.1 Principios
6.1.2 Objetos
6.1.3 Comunicación entre objetos
6.1.4 Clases
6.1.5 Relaciones entre clases
6.1.6 Jerarquía de clases

6.2 Clasificación de los métodos (30 Minutos)

6.2.1 Métodos dirigidos por los datos
6.2.2 Métodos dirigidos por las responsabilidades
6.2.3 Métodos dirigidos por los casos de uso

6.3 Descripción de algunos métodos (1 Hora y 30 Minutos)

6.3.1 OMT
6.3.2 Método de Booch
6.3.3 Método de Jacobson
6.3.4 OOSA de Shaler/Mellor
6.3.5 SOMA de Graham

Tema 7. El lenguaje UML y el proceso unificado (18 Horas)

7.1 Génesis y evolución de UML (20 Minutos)
7.2 Visión general de UML (10 Minutos)
7.3 Elementos de modelado (10 Minutos)
7.4 Las vistas (20 Minutos)
7.5 Vista estática (6 Horas)
7.5.1 Generalidades
7.5.2 Clasificadores
7.5.3 Relaciones
7.6 Vista de casos de uso (3 Horas)
7.6.1 Generalidades
7.6.2 Actores
7.6.3 Casos de uso
7.7 Vista de máquina de estados (2 Horas)
7.7.1 Generalidades
7.7.2 Máquinas de estados
7.7.3 Eventos
7.7.4 Estados
7.7.5 Transiciones
7.7.6 Estados compuestos
7.8 Vista de actividad (1 Hora)
7.8.1 Generalidades
7.8.2 Diagramas de actividad
7.9 Vista de interacción (3 Horas)
7.9.1 Generalidades
7.9.2 Colaboración
7.9.3 Interacción
7.9.4 Diagrama de secuencia
7.9.5 Activación
7.9.6 Diagrama de colaboración
7.10 Vistas físicas (30 Minutos)
7.10.1 Generalidades
7.10.2 Componentes
7.10.3 Nodos
7.11 Vista de gestión de modelo (30 Minutos)
7.11.1 Generalidades
7.11.2 Dependencias de paquetes
7.11.3 Modelos y subsistemas
7.12 Proceso unificado de desarrollo de software (1 Hora)

Tema 8. Técnicas formales de especificación (10 Horas)

8.1 Introducción (10 Minutos)
8.2 Base matemática de los métodos formales (50 Minutos)
8.3 Lenguajes formales de especificación (10 Minutos)

8.4 Lenguaje Z (1 Hora y 50 Minutos)
8.5 OCL (3 Horas)
8.6 Redes de Petri (1 Hora y 30 Minutos)
8.7 Redes de Petri Coloreadas (2 Horas y 30 Minutos)

Unidad Didáctica III: Desarrollo de sistemas complejos (8 Horas)

Tema 9. Desarrollo de sistemas complejos (8 Horas)

9.1 Sistemas de tiempo real (3 Horas)
9.1.1 Introducción
9.1.2 Características de desarrollo
9.1.3 Aspectos de los sistemas de tiempo real
9.1.4 Análisis de sistemas de tiempo real
9.1.5 Diseño de sistemas de tiempo real
9.1.6 Modelado orientado a objetos

9.2 Sistemas Cliente/Servidor (C/S) (2 Horas)
9.2.1 Estructura de los sistemas C/S
9.2.2 Distribución de componentes de software
9.2.3 Software intermedio (*Middleware*)
9.2.4 Análisis
9.2.5 Diseño

9.3 Ingeniería Web (2 Horas)
9.3.1 Introducción
9.3.2 Análisis
9.3.3 Diseño

9.4 Sistemas Interoperativos (30 Minutos)
9.5 Sistemas para el soporte de decisiones (30 Minutos)

Unidad Didáctica IV: Evolución del software (3 Horas)

Tema 10. Evolución y mantenimiento del software (3 Horas)

10.1 Introducción (20 Minutos)
10.2 Actividades de mantenimiento (20 Minutos)
10.3 El proceso de mantenimiento (20 Minutos)
10.3.1 Estándares
10.3.2 Modelo de proceso de mantenimiento

10.4 Problemas del mantenimiento (40 Minutos)
10.4.1 Problemas técnicos
10.4.2 Problemas de gestión

10.5 Costes de mantenimiento (20 Minutos)
10.6 Herramientas y técnicas (20 Minutos)
10.7 Ingeniería inversa y reingeniería (40 Minutos)

Tabla 5.24. Estructura detallada del programa de teoría de Análisis de Sistemas

Unidad Didáctica I: Definición del proceso de software

Objetivo genérico

Según Roger S. Presuman la Ingeniería del Software debe verse como una tecnología multicapa que tiene a la calidad como principio central y su fundamento en el proceso [Pressman, 2000].

Cuando se trabaja para construir un producto o un sistema, es importante seguir una serie de pasos predecibles, un *mapa de carreteras* que ayude a obtener el resultado oportuno de calidad. El mapa de carreteras a seguir es el "proceso del software".

Los ingenieros de software y sus gestores adaptan el proceso a sus necesidades y entonces lo siguen. Además, las personas que han solicitado el software tienen un papel a desempeñar en el proceso del software.

Como el software, al igual que el capital, es el conocimiento incorporado, y puesto que el conocimiento está inicialmente disperso, el desarrollo del software implícito, latente e incompleto en gran medida, es un proceso social de aprendizaje. El proceso es un diálogo en el que se reúne el conocimiento y se incluye en el software para convertirse en software. El proceso proporciona una interacción entre los usuarios y los diseñadores, entre los usuarios y las herramientas de desarrollo, y entre los diseñadores y las herramientas de desarrollo [tecnología]. Es un proceso interactivo donde la herramienta de desarrollo se usa como medio de comunicación, con cada iteración del diálogo se obtiene mayor conocimiento de las personas involucradas [Baetjer, 1998].

Construir software es, por tanto, un proceso de aprendizaje iterativo, y el resultado, es el conjunto del software reunido, depurado y organizado mientras se desarrolla el proceso.

Ante estas premisas, esta unidad didáctica tiene como objetivo resaltar la importancia del proceso en el desarrollo de un producto software de calidad. Es por ello que se va a situar su contexto, se van a recordar y ampliar los conceptos que ya tiene el alumno sobre los modelos del ciclo de vida del software, y se va a hacer hincapié en los modelos iterativos e incrementales basados en el concepto de prototipado.

Esta unidad supone el 12% de la asignatura, estando compuesta por tres temas, **Sistemas de Información, Modelos avanzados de ciclo de vida** y **Prototipos**, que se detallan a continuación.

Tema 1: ***Sistemas de Información***

Descriptores

Sistema; sistema de información; sistema de información automatizado; elementos de un sistema de información; estructura de un sistema de información; sistema transaccional; sistema de apoyo a las decisiones.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T3** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Repasar los principios generales de los sistemas.
- Detallar la estructura jerárquica de un sistema de información.
- Diferenciar las características de los diferentes tipos de sistemas de información automatizados.

Contenidos

1.1 Historia y evolución
1.2 Elementos de un sistema de información
1.3 Estructura de un sistema de información
1.4 Tipos de sistemas de información automatizados
1.5 Principios generales de sistemas

Tabla 5.25. Contenido del primer tema del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del curso **IS97.1 Fundamentals of Information Systems** del *IS'97* [Davis et al., 1997].

Resumen

El entorno comercial actual es sumamente complejo y competitivo, donde la rapidez de respuesta resulta clave para el éxito de las empresas, por tanto resulta crucial contar con la información adecuada para actuar y tomar las mejores decisiones. Todo esto conduce a las empresas y organismos a crear sus propios sistemas de información, dirigidos a lograr los objetivos de la organización.

Toda empresa, independientemente de sus dimensiones, necesita contar con una infraestructura para llevar a cabo sus actividades. Esta estructura organizativa suele recaer sobre una red de funciones a desarrollar (control y gestión del empleo de los recursos financieros,

comercialización óptima de los productos o servicios, fabricación de los productos, creación de servicios…). No obstante, resultaría muy difícil por no decir imposible el llevar a cabo todas estas tareas sin coordinarlas e interrelacionarlas entre sí. Por este motivo, las organizaciones deben incluir una infraestructura para coordinar los flujos y los registros de información necesarios para desarrollar sus actividades de acuerdo a su planteamiento o estrategia de negocio [Andreu et al., 1996]. El sistema dedicado a este cometido es el que se denomina sistema de información de la empresa, siendo tan importante como lo puedan ser el resto de sistemas (producción, ventas, contabilidad…).

Un error bastante difundido en la actualidad es la confusión entre sistema de información y tecnologías de la información (TI), esto es pensar en sistemas de información en términos de ordenadores, programas e instrumentos sofisticados. Debe tenerse presente que los sistemas de información existen desde el momento en que se crea la primera organización humana. Por todo ello, la implementación actual de los sistemas de información en las organizaciones empleando sofisticadas TI no debe ocultar el concepto original subyacente.

El tema se organiza en cinco apartados. En el primero de ellos se presentan los conceptos generales (sistema, sistema de información y sistema de información automatizado).

En el segundo apartado se revisan los elementos de un sistema de información, mientras que en el tercero se explica la estructura de un sistema de información.

El cuarto apartado revisa las principales características de los diferentes tipos de sistemas de información automatizados, a saber: Sistemas de procesamiento de transacciones, Sistemas de automatización de oficina, Sistemas de manejo de conocimiento, Sistemas de información administrativa o gerencial, Sistemas de apoyo a decisiones, Sistemas de apoyo a decisiones de grupo, Sistemas de apoyo a ejecutivos o de planificación estratégica.

Por último, el quinto apartado repasa los principios de la teoría general de sistemas [Bertalanffy, 1968].

Bibliografía básica

Kendall, K. E., Kendall, J. E. "*Análisis y Diseño de Sistemas*". Prentice Hall, 1997. (Capítulos 1 y 2).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulos 1 y 2).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 2).

Yourdon, E. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993. (Capítulos 2 y 3).

Bibliografía complementaria

Andreu, R., Ricart, J., Valor, J. "*Estrategia y Sistemas de Información*". 2ª Ed. McGraw-Hill (*serie de management*), 1996.

Bertalanffy, L. von. "*General Systems Theory: Foundations, Development, Applications*". George Brazillier, New York, 1968.

Davis, G., Olson, M. "*Sistemas de Información Gerencial*". Mc Graw Hill, 1987.

Leff, A., Pu, C. "*A Classification of Transaction Processing Systems*". IEEE Computer, 24(6):63-76. June 1991.

Tema 2: ***Modelos avanzados de ciclo de vida***

Descriptores

Ciclo de vida; modelo de ciclo de vida; procesos del ciclo de vida; norma ISO 12207-1; modelo clásico; ciclo en V; ciclo de vida iterativo; ciclo de vida en espiral; modelo de desarrollo rápido de aplicaciones; enfoque de sala limpia; modelo de desarrollo concurrente; ciclos de vida orientados a objetos.

Objetivos

Este tema está orientado a satisfacer los objetivos **T3** y **T4** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar diferentes modelos de ciclo de vida que se separan del modelo clásico. Se intenta de esta forma, transmitir una concepción más dinámica de las actividades de construcción y mantenimiento de sistemas de software.
- Establecer el desarrollo de software como un proceso interdisciplinar, que implica la colaboración del usuario, y que se ve favorecido por el conocimiento que tenga el ingeniero del software del dominio del problema.

Contenidos

2.1 Concepto de ciclo de vida
2.2 Procesos del ciclo de vida
2.3 Modelos de ciclo de vida

Tabla 5.26. Contenidos del tema dos del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE4 Software processes** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Process** [El Emam, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

En este tema introductorio se presentan diferentes modelos de ciclo de vida que se separan del modelo clásico. Se intenta de esta forma, transmitir una concepción más dinámica de las actividades de construcción y mantenimiento de sistemas de software.

Antes de pasar a comentar los modelos se ofrecen diferentes definiciones tanto de ciclo de vida como de modelo de ciclo de vida y se describen las actividades que lo componen clasificándolas previamente según la norma ISO 12207-1 [ISO/IEC, 1995].

En el tercer apartado del tema se realiza un repaso rápido a los modelos de ciclo de vida tradicionales (clásico, estructurado [Yourdon, 1989] y prototipos) que servirá de base para la introducción de los modelos evolutivos (en espiral [Boehm, 1988], evolutivo [Land, 1982], incremental [McDermid y Rook, 1993] y modelo de ensamblaje de componentes [Nierstrasz, 1992]) en los que se considera la posible evolución de los requisitos a lo largo del desarrollo. También se contemplan modelos como el de desarrollo concurrente [Davis y Sitaram, 1994] en el que se definen una serie de acontecimientos que disparan transiciones de estado a estado para cada una de las actividades de la Ingeniería del Software o el modelo de desarrollo rápido (RAD) [Martin, 1991] que se fundamenta en el desarrollo de software basado en componentes y el uso de técnicas JAD [Wood y Silver, 1995].

Termina este apartado con el estudio de los modelos de ciclo de vida propuestos específicamente para sistemas orientados a objetos, entre los que destacan el modelo remolino [Rumbaugh, 1992], el modelo *pinball* [Ambler, 1994], el modelo de agrupamiento [Meyer, 1988], el modelo fuente [Henderson-Sellers y Edwards, 1990] y el modelo de ciclo de vida iterativo e incremental, basado en casos de uso, propuesto en UP (*Unified Process*) [Jacobson et al., 1999].

Bibliografía básica

Amescua, A., García, L. Martínez, P., Díaz, P. "*Ingeniería del Software de Gestión*". Paraninfo, 1995. (Capítulo 1).

Jacobson, I., Booch, G., Rumbaugh, J. "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000. (Capítulo 1).

Muller, P. A. "*Modelado de objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997. (Capítulo 4).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 3).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 2).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 3).

Bibliografía complementaria

Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72, May 1988.

Bowen, J., Bahler, D. "*Constraint-Based Software for Concurrent Engineering*". IEEE Computer, 26(1):66-68. January 1993.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Cockburn, A. "*Agile Software Development*". Addison-Wessley, 2001.

Comer, E. "*Alternative Software Life Cycle Models*". En Dorfmann, M., Thayer, R. (eds.): *Software Engineering*, IEEE-CS Press, 1997.

Conger, S. "*The New Software Engineering*". Course Technology, 1994.

Davis, A., Sitaram, P. "*A concurrent Process Model for Software Development*", Software Engineering Notes, ACM Press, 19(2):38-51. February 1994.

Dewan, P., Riedl, J. "*Toward Computer-Supported Concurrent Software Engineering*". IEEE Computer, 26(1):17-27. January 1993.

El Emam, K. "*Software Engineering Process*". Chapter 9 in [Abran et al., 2001].

Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International Editions, 1991.

Gutiérrez, I., Medinilla, N. "*Contra el Arraigo de la Cascada*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. P Botella, J. Hernández y F. Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999.

Harel, D. "*Biting the Silver Bullet. Toward a Brighter Future for System Development*". IEEE Computer, 25(1):8-20. January 1992.

Henderson-Sellers, B. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Henderson-Sellers, B., Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990.

Karlsson, E.-A. (Editor). "*Software Reuse. A Holistic Approach*". Wiley Series in Software Based Systems. John Wiley and Sons Ltd., 1995.

Maciaszek, L. A. "*Requirements Analysis and System Design. Developing Information Systems with UML*". Addison Wesley, 2001. (Capítulos 1 y 3).

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Object Management Group. "*Software Process Engineering Metamodel Specification*". http://www.omg.org. December 2001.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Piattini Velthuis, M. G. "*Ciclos de vida para Sistemas Orientados a Objetos*". Cuore, (7):6-11. Septiembre, 1995.

Rumbaugh, J. "*Over the Waterfall and Into the Whirlpool*", Journal of Object-Oriented Programming (JOOP), 5(2):23-26. May 1992.

Scacchi, W. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). October 1987.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Tema 3: *Prototipos*

Descriptores

Prototipo; prototipo evolutivo; prototipo desechable; prototipo operativo; tipos de prototipos; herramientas de construcción de prototipos.

Objetivos

Este tema está orientado a satisfacer los objetivos **T3** y **T4** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos,* a saber:

- Importancia de los requisitos en el ciclo de vida del software.
- Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Conocer el papel de la construcción de prototipos de software en los diferentes tipos de desarrollo de proyectos.
- Conocer la diferencia entre la construcción de prototipos evolutivos y la de prototipos desechables.

Contenidos

3.1 ¿Qué es un prototipo?
3.2 Enfoques en la construcción de prototipos
3.3 Tipos de prototipos
3.4 Usos de los prototipos
3.5 Guías para el desarrollo de prototipos
3.6 Beneficios de los prototipos
3.7 Herramientas para la construcción de prototipos

Tabla 5.27. Contenidos del tema tres del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE5 Software requirements and specifications** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Requirements** [Sawyer y Kotonya, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Aunque los prototipos ya han sido introducidos con anterioridad como modelo de ciclo de vida, en este tema se pretende profundizar en las razones y circunstancias de su uso, en la mejor forma de llevarlos a cabo y las ventajas que pueden obtenerse de su construcción. Para ello se examinan con detenimiento los dos enfoques de su aplicación, el enfoque evolutivo [Basili y Turner, 1975] en el que se utiliza como alternativa de ciclo de vida y el enfoque desechable [Gomaa, 1981; Davis, 1982; Boehm et al., 1984] en el que su propósito es validar algún aspecto del sistema, sirviendo, en este caso, como herramienta auxiliar a la especificación de requisitos y el diseño. Se hace mención también al enfoque mixto conocido con el nombre de prototipado operativo [Davis, 1992].

Dar a conocer la mayor o menor adecuación de los prototipos a diferentes situaciones, proporcionar una serie de recomendaciones para su desarrollo eficiente, así como examinar las herramientas de apoyo a su construcción son también objetivos de este tema.

Bibliografía básica

Davis, A. M. "*Software Requirements, Objects, Functions and States*". Prentice Hall, 1993. Capítulo 6).

Jacobson, I., Booch, G., Rumbaugh, J. "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000.

Kendall, K. E., Kendall, J. E. "*Análisis y Diseño de Sistemas*". Prentice Hall, 1997. (Capítulo 8).

McConnell, S. "*Desarrollo y Gestión de Proyectos Informáticos*". Mc Graw Hill, 1997. (Capítulo 21).

Muller, P. A. "*Modelado de objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 2).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 3).

Bibliografía complementaria

Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72, May 1988.

Conger, S. "*The New Software Engineering*". Course Technology, 1994.

Connell, J., Shafer L. "*Object-Oriented Rapid Prototyping*". Englewood Cliffs, Yourdon Press, 1995.

Dewan, P., Riedl, J. "*Toward Computer-Supported Concurrent Software Engineering*". IEEE Computer, 26(1):17-27. January 1993.

El Emam, K. "*Software Engineering Process*". Chapter 9 in [Abran et al., 2001].

Gomaa, H., Scott, P. "*Prototyping as a Tool in the Especification of User Requirements*". ACM Software Engineering Notes, 8(2):17-28, 1983.

Gordon, V. S., Bieman, J. M. "*Rapid Prototyping: Lessons Learned*". IEEE Software, 12(1):85-95. January 1995.

Gutiérrez, I., Medinilla, N. "*Contra el Arraigo de la Cascada*". En las actas de las IV Jornadas de Ingeniería del Software y Bases de Datos, JISBD'99. P Botella, J. Hernández y F. Saltor editores. (24-26 de noviembre de 1999, Cáceres - España). Páginas 393-404. 1999.

Harel, D. "*Biting the Silver Bullet. Toward a Brighter Future for System Development*". IEEE Computer, 25(1):8-20. January 1992.

Henderson-Sellers, B. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Henderson-Sellers, B., Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990.

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Myers, B. A. "*User-Interface Software Tools*". ACM Transactions on CHI, 2(1). 1995.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice Hall, 1998.

Piattini Velthuis, M. G. "*Ciclos de vida para Sistemas Orientados a Objetos*". Cuore, (7):6-11. Septiembre, 1995.

Rumbaugh, J. "*Over the Waterfall and Into the Whirlpool*", Journal of Object-Oriented Programming (JOOP), 5(2):23-26. May 1992.

Scacchi, W. "*Models of Software Evolution: Life Cycle and Process*". SEI Curriculum Module SEI-CM-10-1.0. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). October 1987.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Unidad Didáctica II: Métodos de desarrollo

Objetivo genérico

Esta unidad didáctica tiene el objetivo de profundizar en los métodos de desarrollo de software, que fueron introducidos en la asignatura *Ingeniería del Software*, propia de los estudios del primer ciclo.

La Ingeniería del Software es una disciplina muy amplia como para poder tratarla en su amplitud en un solo curso. El primer contacto serio con esta disciplina lo tienen los alumnos de primer ciclo con los créditos troncales u obligatorios que las Universidades Españolas dedican a la Ingeniería del Software.

En el caso de la Universidad de Salamanca, se ha visto en este mismo Proyecto Docente que dichos créditos son 6, y son asumidos por la asignatura *Ingeniería de Software*, impartida en el primer cuatrimestre del tercer curso de la Ingeniería Técnica.

Los conceptos aquí impartidos se centran fundamentalmente en asentar las bases de la disciplina y, siguiendo las directrices del Plan de Estudios en vigor, en los métodos estructurados, dedicando escaso tiempo a los métodos orientados a objetos, que se ven reducidos a una introducción a UML [OMG, 2001c], como lenguaje de modelado, y a OMT [Rumbaugh et al., 1991], como método de desarrollo.

En el segundo ciclo, en el contexto de la asignatura *Análisis de Sistemas*, se cambian las tornas, y el paradigma estructurado queda bastante relegado frente a los métodos orientados a objetos.

Concretamente, esta segunda unidad didáctica, además de suponer el grueso de la asignatura, se dedica exclusivamente a los métodos de desarrollo, repasando tanto los principales métodos estructurados como orientados a objetos, aunque se centra especialmente en tres tópicos: la metodología Métrica versión 3 [MAP, 2001], UML [OMG, 2001c] y los métodos formales [Lamsweerde, 2000]; poniendo en todos ellos un énfasis especial en las fases de elicitación y especificación de requisitos.

Esta unidad didáctica supone el **68%** del programa teórico de la asignatura *Análisis de Sistemas*, estando compuesta por cinco temas: **Métodos estructurados: Técnicas avanzadas**; **Metodología MÉTRICA V.3**; **Métodos orientados a objetos**; **El lenguaje UML y el proceso unificado**; y **Técnicas formales de especificación**, que se detallan a continuación.

Tema 4: ***Métodos estructurados. Técnicas avanzadas***

Descriptores

Métodos estructurados; descomposición funcional de un sistema; perspectiva histórica de los métodos estructurados; clasificación de métodos estructurados; técnicas de modelado; diccionario de datos; diagrama de flujo de datos; ampliaciones de tiempo real en el modelado funcional; especificación de procesos; especificación de control; diagrama entidad/relación; modelo de eventos; historia de la vida de la entidad; diagrama de transición de estados; red de petri.

Objetivos

Este tema está orientado a satisfacer el objetivo **T6** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis /diseño estructurado.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Revisar las principales metodologías estructuradas de desarrollo.
- Profundizar en las técnicas de modelado que el alumno ya conoce.
- Introducir técnicas de modelado avanzadas.

Contenidos

4.1 Introducción
4.2 Perspectiva histórica
4.3 Clasificación de los métodos
4.4 Principales métodos de desarrollo
4.5 Técnicas de modelado

Tabla 5.28. Contenidos del tema cuatro del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE5 Software requirements and specifications** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Requirements** [Sawyer y Kotonya, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El objetivo general de este tema es la revisión de las principales metodologías estructuradas de desarrollo, la profundización en las técnicas de modelado que el alumno ya conoce y la introducción de otras técnicas más avanzadas.

Se han seleccionado aquellos métodos que más relevancia han tenido desde que comenzaron a gestarse las primeras técnicas que intentaban sistematizar y formalizar el proceso de especificación de requisitos. Se incluyen también métodos creados por iniciativa de instituciones gubernamentales de países europeos debido a la repercusión que han tenido en nuestro país.

Para realizar el estudio de las técnicas de especificación se ha tomado como base la clasificación de las mismas según las tres perspectivas que propone Yourdon [Yourdon Inc., 1993] para examinar un sistema: función, información y tiempo. Así se han incluido técnicas de modelado funcional como los diagramas de flujo de datos, considerando principalmente las extensiones introducidas para la especificación de sistemas de tiempo real [Ward y Mellor, 1985; Hatley y Pirbhai, 1987], los diagramas jerárquicos de descomposición funcional y técnicas de especificación de procesos: lenguaje estructurado, árboles de decisión, tablas de decisión, diagramas de acción. En el apartado de modelado de datos se realiza un repaso del modelado entidad-relación y se introduce una técnica muy similar, los diagramas de estructura de datos. En cuanto al modelado del comportamiento dependiente del tiempo se presentan técnicas orientadas a estados y orientadas a eventos. La introducción a las primeras se realiza estudiando las máquinas de estados finitos y sus notaciones usuales, las matrices y los diagramas de transición de estados. Posteriormente, se exponen las extensiones sugeridas por Harel, cuya notación se conoce con el nombre de diagramas de estado [Harel, 1987]. Se finalizan las técnicas de especificación de control con el estudio de las Redes Petri [Petri, 1962; Peterson, 1977; Peterson, 1981] que comenzaron a usarse para representar la sincronización de procesos en la fase de diseño, aunque actualmente se ha extendido su uso a la fase de análisis, ampliándose también la notación original [Coolahan y Roussopoulos, 1983; Bruno y Manchetto, 1986; Silva, 1985].

Además de esas técnicas asociadas a alguna de las tres perspectivas comentadas, se incluyen notaciones que permiten establecer conexiones entre aspectos de más de una de ellas. En este grupo se encuentran las técnicas matriciales y el modelado evento-entidad cuyo diagrama más representativo es el de historia de la vida de las entidades. Este apartado de especificación de requisitos termina con algunas consideraciones sobre la definición de interfaces externas. Finalmente, se comenta la forma de realizar una revisión de las

especificaciones basándose en cuatro aspectos: compleción, integridad, exactitud y calidad [Yourdon, 1985].

Bibliografía básica

Davis, A. M. "*Software Requirements, Objects, Functions and States*". Prentice Hall, 1993. (Capítulos 2 y 4).

Kendall, K. E., Kendall, J. E. "*Análisis y Diseño de Sistemas*". Prentice Hall, 1997. (Capítulos 9, 10, 11, 15, 16 y 18).

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulos 4 y 7).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 12).

Silva, M. "*Las Redes de Petri en la Automática y la Informática*". Editorial AC, 1985.

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 7).

Yourdon, E. "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993. (Capítulos 8, 9, 10, 11, 12, 13 y 14).

Bibliografía complementaria

Amescua, A., García, L. Martínez, P., Díaz, P. "*Ingeniería del Software de Gestión*". Paraninfo, 1995.

Davis, A. M., Jordan, K., Nakajima, T. "*Elements Underlying the Specification of Requirements*". Annals of Software Engineering, 3:63-100. 1997.

Durán Toro, A. "*Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*". Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/publicaciones/tesis.pdf.zip. Septiembre de 2000.

Durán, A., Bernárdez, B. "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.

Gabay, J. "*Aprender y Practicar MERISE*". Masson, 1991.

Gaitero, D. "*Metodología Métrica*". Everest, 1997.

Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International Editions, 1991.

Hatley, D. J., Pirbhai, I. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

Martin, J., McClure, C. "*Structured Techniques: The basis for CASE*". Prentice-Hall, 1988.

Matheron, J.-P. "*Merise - Metodología de Desarrollo de Sistemas. Casos Prácticos*", Paraninfo, 1990.

Nuseibeh, B., Easterbrook, S. "*Requirements Engineering: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 35-46. ACM Press, 2000.

Paternò, F. "*Model-Based Design and Evaluation of Interactive Applications*". Springer-Verlag, 2000.

Peterson, J. "*Petri Nets*". ACM Computing Surveys 9(3):223-252, 1977.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Ward, P. T., Mellor, S. J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Prentice-Hall, 1979.

Tema 5: ***Metodología MÉTRICA V.3***

Descriptores

Metodología Métrica v3; metodología de desarrollo; antecedentes de Métrica; estructura de Métrica v3; planificación de sistemas de información; desarrollo de sistemas de información; estudio de viabilidad del sistema; análisis del sistema de información; diseño del sistema de información; construcción del sistema de información; implantación del sistema de información; mantenimiento del sistema de información; procesos de interfaz.

Objetivos

Este tema está orientado a satisfacer los objetivos **T6** y **T7** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis /diseño estructurado.
- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Conocer la metodología de desarrollo de software propuesta por el Ministerio de Administraciones Públicas, Métrica V3.
- Explicar la doble vertiente estructurada y orientada a objetos que presenta esta versión de la metodología.
- Aprovechar el marco metodológico de Métrica para hacer un repaso integrador de las fases del ciclo de vida, relacionando éstas con las principales técnicas utilizadas.
- Aprovechar las características de Métrica para incidir en la separación de documentación de requisitos y de especificación de requisitos.

Contenidos

5.1 Introducción
5.2 Estructura de la metodología
5.3 Planificación de Sistemas de Información
5.4 Desarrollo de Sistemas de Información
5.5 Mantenimiento de Sistemas de Información
5.6 Procesos de interfaz

Tabla 5.29. Contenidos del tema cinco del programa teórico de Análisis de Sistemas

Al tratar este tema de una metodología concreta no tiene un soporte directo ni en el *Computing Curricula 2001* [ACM/IEEE-CS, 2001], ni en el **SWEBOK** (*Software Engineering Body of*

Knowledge) [Abran et al., 2001], aunque los contenidos estarían relacionados con los tópicos de las áreas **SE1 Software design, SE4 Software processes, SE5 Software requirements and specifications, SE7 Software evolution y SE8 Software Project management** del *Computing Curricula 2001*, y con las áreas de conocimiento **Software Engineering Process** [El Emam, 2001], **Software Requirements** [Sawyer y Kotonya, 2001], **Software Design** [Tremblay, 2001], **Software Construction** [Bollinger et al., 2001], **Software Maintenance** [Pigoski, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK.**

Resumen

La nueva versión de Métrica [MAP, 2001] contempla el desarrollo de Sistemas de Información para las distintas tecnologías que actualmente están conviviendo y los aspectos de gestión que aseguran que un proyecto cumple con sus objetivos en términos de calidad, coste y plazos.

Su punto de partida es la versión anterior de Métrica, de la cual se han conservado la adaptabilidad, flexibilidad y sencillez, así como la estructura de actividades y tareas, si bien las fases y módulos de Métrica 2.1 han dado paso a la división en procesos, más adecuada a la entrada-proceso-salida que se produce en cada una de las divisiones del ciclo de vida de un proyecto. Para cada tarea se detallan los participantes que intervienen, los productos de entrada y de salida así como las técnicas y prácticas a emplear para su obtención.

En la elaboración de Métrica Versión 3 se han tenido en cuenta los métodos de desarrollo más extendidos, así como los últimos estándares de Ingeniería del Software y calidad, además de referencias específicas en cuanto a seguridad y gestión de proyectos. También se ha tenido en cuenta la experiencia de los usuarios de las versiones anteriores para solventar los problemas o deficiencias detectados.

En una única estructura la metodología Métrica Versión 3 cubre distintos tipos de desarrollo: estructurado y orientado a objetos, facilitando a través de interfaces la realización de los procesos de apoyo u organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad y Seguridad.

Se ha ampliado el enfoque de la Planificación de Sistemas de Información (PSI) respecto a Métrica Versión 2.1, incluyendo planificación estratégica y recogiendo las actividades de más alto nivel de la fase PSI de Métrica Versión 2.1. Las actividades restantes de la antigua fase PSI se han incorporado al proceso de Desarrollo de la actual versión. Igualmente, aparece el proceso de Mantenimiento de Sistemas de Información que no estaba contemplado en MÉTRICA Versión 2.1.

Se ha reforzado el ciclo de vida de las pruebas a través del plan de pruebas y se han mejorado los procedimientos de prueba. Se ha dado respuesta formal a problemáticas específicas de diseño con la incorporación de tecnologías tipo cliente/servidor, interfaces de usuario basadas en entornos gráficos...

La estructura de Métrica Versión 3 está basada en el Modelo de Ciclo de Vida de Desarrollo propuesto en la norma ISO 12.207 "*Information Technology - Software Life Cycle Processes*" [ISO/IEC, 1995], distinguiéndose dos tipos de procesos: los procesos principales y los procesos de interfaz.

Los procesos principales son: Planificación de Sistemas de Información (PSI); Desarrollo de Sistemas de Información y Mantenimiento de Sistemas de Información (MSI).

El proceso de Desarrollo de Sistemas de Información se divide a su vez en cinco subprocesos: Estudio de viabilidad del sistema (EVS); Análisis del sistema de información (ASI); Diseño del sistema de información (DSI); Construcción del sistema de información (CSI) e Implantación y aceptación del sistema (IAS).

Los procesos de interfaz son: Gestión de proyectos; Seguridad; Gestión de la configuración y Aseguramiento de la calidad.

Este tema se centra en la descripción de las actividades y tareas de los procesos principales, así como en una somera presentación de los procesos de interfaz.

Bibliografía básica

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Ministerio de Administraciones Públicas. "*MÉTRICA. Versión 3. Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información*". http://www.map.es/csi/metrica3/. [Última vez visitado, 20-10-2001]. 2001.

Bibliografía complementaria

Gaitero, D. "*Metodología Métrica*". Everest, 1997.

Hofmeister, C., Nord, R., Soni, S. "*Applied Software Architecture*". Addison-Wesley, Object Technology Series, 2000.

Ministerio de Administraciones Públicas. "*Euromético v1*". Ministerio de Administraciones Públicas. http://www.map.es/csi/pg5e40.htm. [Última vez visitado, 24-10-2001]. 1997.

Ministerio de Administraciones Públicas. "*Curso de Autoformación – MÉTRICA Versión 3*". Ministerio de Administraciones Públicas, 2001.

Nuseibeh, B., Easterbrook, S. "*Requirements Engineering: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 35-46. ACM Press, 2000.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 9).

Tema 6: ***Métodos orientados a objetos***

Descriptores

Métodos orientados a objetos; objeto; clase, modelo objeto; métodos dirigidos por datos; métodos dirigidos por responsabilidades; métodos dirigidos por casos de uso; OMT; método de Booch; OOSE; diagrama de clase; diagrama de casos de uso; diagramas de actividad.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Ofrecer una visión general de las técnicas que han tenido mayor difusión dentro del paradigma orientado a objetos.
- Repasar los principios del modelo objeto.
- Revisar algunos de los métodos orientados a objetos más sobresalientes y en los que se basan la mayoría de las corrientes actuales.

Contenidos

6.1 El paradigma de la orientación al objeto
6.2 Clasificación de los métodos
6.3 Descripción de algunos métodos

Tabla 5.30. Contenidos del tema seis del programa teórico de Análisis de Sistemas

Resumen

El tercer tema de esta unidad didáctica constituye el primero de los dos dedicados al estudio de los de métodos de desarrollo orientados a objetos. Su finalidad consiste en ofrecer una visión general de las técnicas que han tenido mayor difusión. Después de repasar los principios del paradigma objetual, se centra en los objetos y las clases examinando en ambos casos sus características y relaciones. En el estudio de las características de los objetos no sólo se contemplan las relacionadas con su estructura, sino también aquéllas derivadas de las restricciones de realización, entre las que destaca la persistencia.

La comunicación entre objetos se analiza desde el punto de vista del papel que los objetos desempeñan en la comunicación y desde la perspectiva de los mensajes que permiten establecer esa comunicación, clasificándolos, además, por su forma de sincronización.

El concepto de clase se introduce a partir de la definición previa de objeto, se analizan sus características y se explican los diferentes tipos de relaciones que pueden presentar.

Cuando se aplican métodos orientados a objetos cobra una importancia especial la noción de análisis del dominio [Arango y Prieto-Díaz, 1991; Firesmith, 1993], ya que la identificación, análisis y especificación de capacidades comunes para la reutilización dentro de un dominio de aplicación específico, es más factible si se realiza con los componentes de los modelos orientados a objetos. En este tema se examina el proceso de análisis del dominio y se muestran las actividades que lo componen [Berard, 1992].

En el segundo apartado del tema se clasifican los métodos en tres categorías, se discuten sus características y se explican las técnicas que los diferencian, entre ellas destacan las tarjetas de clase (CRC) [Beck y Cunningham, 1989; Wirfs-Brock et al., 1990] y los diagramas de casos de uso [Jacobson, 1987; Jacobson et al., 1993].

La revisión que se hace de los métodos de desarrollo en el último apartado es mucho más extensa que en el caso de los métodos estructurados debido, no sólo a la importancia que están adquiriendo frente a las técnicas estructuradas, sino también debido a que los conocimientos que el alumno tiene de ellos es mucho menor. Se presta especial atención a la metodología OMT [Rumbaught et al., 1991] y a los tres enfoques de modelado que propone: modelo de objetos, dinámico y funcional. Aunque también se estudian las características del Método de Booch [Booch, 1994], de OOSE/Objectory [Jacobson et al., 1993], del Método de Shaler y Mellor [Shaler y Mellor, 1992], del Método de Coad/Yourdon [Coad y Yourdon, 1991] y de SOMA [Graham, 1994; Graham, 1995].

Bibliografía básica

Barbier, F., Henderson-Sellers, B. "*Object Modelling Languages: An Evaluation and some Key Expectations for the Future*". Annals of Software Engineering, 10:67-101. 2000.

Booch, G. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Diaz de Santos, 1996. (Capítulos 1, 2, 3, 4, 5, 6 y 7).

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Maciaszek, L. A. "*Requirements Analysis and System Design. Developing Information Systems with UML*". Addison Wesley, 2001. (Capítulo 2).

Muller, P. A. "*Modelado de Objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997. (Capítulos 2 y 4).

Graham, I. "*Métodos Orientados a objetos*". Addison-Wesley, 1996. (Capítulo 8).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 10).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 20, 21 y 22).

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998. (Capítulos 1, 3, 4, 7, 8, 9, 10 y 11).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 12).

Bibliografía complementaria

Alonso, F., Segovia F. J. "*Entornos y Metodologías de Programación*". Paraninfo, 1995.

Budd, T. "*Introducción a la Programación Orientada a Objetos*". Addison-Wesley, 1994.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Champeaux, D., Lea, D., Faure, P. "*Object-Oriented System Development*". Addison Wesley, 1993.

Durán Toro, A., Bernárdez Jiménez, B. "*Metodología para el Análisis de Requisitos de Sistemas Software. (versión 2.2)*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, diciembre de 2001.

Engels, G., Groenewegen, L. "*Object-Oriented Modeling: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 103-104. ACM Press, 2000.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

Graham, I., Bischof, J., Henderson-Sellers, B. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February 1997.

Henderson-Sellers, B. "*A Book of Object-Oriented Knowledge. An Introduction to Object-Oriented Software Engineering*". 2nd Edition. Prentice Hall. The Object-Oriented Series, 1997.

Henderson-Sellers, B. "*OPEN Relationships – Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December 1997.

Henderson-Sellers, B., Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

Hofmeister, C., Nord, R., Soni, S. "*Applied Software Architecture*". Addison-Wesley, Object Technology Series, 2000.

Martin, J., Odell, J. J. "*Métodos Orientados a Objetos: Conceptos Fundamentales*". Prentice Hall, 1997.

Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Jézéquel, J.-M., Meyer, B. "*Design by Contract: The Lessons of Ariane*". IEEE Computer, 30(1):129-130. January 1997.

Lea, D. "*Christopher Alexander: An Introduction for Object-Oriented Designers*". Software Engineering Notes, ACM SIGSOFT, 19(1):39-46, January 1994 (Online version: http://g.oswego.edu/dl/ca/ca/ca.html).

Martin, J., Odell, J. J. "*Métodos Orientados a Objetos: Consideraciones Prácticas*". Prentice Hall Hispanoamericana, 1997.

Martin, R. C. "*The Open Closed Principle*". C++ Report, 8(1). January 1996.

Martin, R. C. "*The Liskov Substitution Principle*". C++ Report, 8(3). March 1996.

Martin, R. C. "*The Dependency Inversion Principle*". C++ Report, 8(5). May 1996.

Martin, R. C. "*The Interface Segregation Principle*". C++ Report, 8(7). July-August 1996.

Martin, R. C. "*Granularity*". C++ Report, 8(10). November-December 1996.

Martin, R. C. "*Principles of OOD*". OMA. 1997.

Martin, R. C. "*Stability*". C++ Report, 9(2). February 1997.

McClure, C. "*Experiences from the OO Playing Field*". The Object World Insider, 2(4):1-3. 1996.

Meyer, B. "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice-Hall, 1999.

Nerson, J.-M. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September 1992.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice-Hall, 1998.

Rubin, K. S., Goldberg, A. "*Object Behavior Analysis*". Communications of the ACM 35 (9): 48-62. September 1992.

Rumbaugh, J. "*Building Boxes: Composite Objects*". Journal of Object-Oriented Programming (JOOP), 7(7):12-22. November-December 1994.

Rumbaugh, J. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

Rumbaugh, J. E. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). ACM. Reprinted in ACM SIGPLAN 22(12):466-481. October 1987.

Sakkinen, M. "*Disciplined Inheritance*". In Proceedings of ECOOP'89. Cook, S. editor. Pages 39-56. Cambridge University Press, 1989.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Taivalsaari, A. "*On the Notion of Inheritance*". ACM Computing Surveys, 28(3). 1996.

Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice Hall, 1990.

Yourdon, E., Argila, C. "*Case Studies in Object Oriented Analysis & Design*". Yourdon Press Computing Series. Prentice Hall, 1996.

Yourdon, E., Whitehead, K., Toman, J., Oppel, K., Nevermann, P. "*Mainstream Objects. An Analysis and Design Approach for Business*". Yourdon Press, 1995.

Tema 7: ***El lenguaje UML y el proceso unificado***

Descriptores

UML; génesis y evolución; modelado de objetos; metamodelo; meta-metamodelo; MOF; elementos de modelado; vistas de UML; vista estática; diagrama de clases; clasificador; clase; objeto; interfaz; relación; vista de casos de uso; diagrama de casos de uso; actor; caso de uso; relaciones entre casos de uso; vista de máquina de estados; máquina de estados; estado; evento; transición; vista de actividad; diagrama de actividad; vista de interacción; colaboración; interacción; diagrama de secuencia; diagrama de colaboración; vistas físicas; nodo; componente; vista de gestión de modelos; UP.

Objetivos

Este tema está orientado a satisfacer el objetivo **T7** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Método de análisis/diseño orientado a objetos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar los principales conceptos del lenguaje estándar de modelado UML, sus reglas, sintaxis y semántica.
- Describir sintáctica y semánticamente las vistas de UML.
- Introducir los principios del proceso unificado de desarrollo o UP (*Unified Process*).

Contenidos

7.1 Génesis y evolución de UML
7.2 Visión general de UML
7.3 Elementos de modelado
7.4 Las vistas
7.5 Vista estática
7.6 Vista de casos de uso
7.7 Vista de máquina de estados
7.8 Vista de actividad
7.9 Vista de interacción
7.10 Vistas físicas
7.11 Vista de gestión de modelo
7.12 Proceso unificado de desarrollo de software

Tabla 5.31. Contenidos del tema siete del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE1 Software design** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con las áreas de conocimiento **Software Design** [Tremblay, 2001] y **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Este tema tiene por objeto presentar los principales conceptos del lenguaje estándar de UML, sus reglas, sintaxis y semántica, contemplando las revisiones realizadas a la versión 1.1 [Rational et al., 1997] que se recogen en la documentación de la versión 1.4 [OMG, 2001c]. Otro de los objetivos es el estudio del enfoque de ciclo de vida ligado a UML, el proceso unificado, desarrollado por los mismos autores del lenguaje [Jacobson et al., 1999].

Para la explicación de los diferentes aspectos de UML se ha realizado una estructuración del tema similar a la que sigue Grady Booch en uno de los tres libros [Booch et al., 1999] de la trilogía publicada por de los tres autores del lenguaje (los otros dos libros son [Jacobson et al., 1999] y [Rumbaugh et al., 1999]). En primer lugar se presenta una introducción a UML y posteriormente se describen los modelos: estructural, de comportamiento y arquitectónico.

El tema comienza con una breve exposición sobre la forma en que surgió UML y como ha evolucionado desde que en 1994 James Rumbaugh y Grady Booch crearan el "método unificado" [Booch y Rumbaugh, 1995], cuyo propósito inicial consistía en unificar el gran número de métodos orientados a objetos que habían proliferado hasta entonces. Posteriormente, con la incorporación de Ivar Jacobson, el método unificado evoluciona y, finalmente, se reorienta hacia un lenguaje de modelado.

En el segundo apartado del tema se ofrece una visión de conjunto de UML, explicando sus objetivos, sus características generales y su arquitectura basada en la estructura de metamodelado de cuatro capas, que es la base para alinear el metamodelo de UML con otros estándares basados en cuatro capas, en particular con MOF (*Meta-Object Facility*) de OMG [OMG, 2001b], de forma que el meta-metamodelo de MOF es el meta-metamodelo para el metamodelo de UML. La introducción al lenguaje termina con el estudio del modelo conceptual del lenguaje en el que se examinan los bloques básicos de construcción, las reglas que permiten combinarlos, los distintos tipos de mecanismos comunes y la arquitectura de cinco vistas centrada en los casos de uso.

La introducción al tema permite seguidamente profundizar en los tres tipos de modelado comentados (estructural, de comportamiento y arquitectónico), cada uno de los cuales engloba

una o varias de las vistas descritas por James Rumbaugh en otro de los libros de la trilogía [Rumbaugh et al., 1999]. Dichas vistas (estática, de casos de uso, de máquina de estados, de actividad, de interacción, física y de gestión de modelos) tienen asociados uno o más diagramas de los nueve propuestos en la notación de UML.

El último apartado se dedica al estudio del proceso unificado [Jacobson et al., 1999], método de desarrollo propuesto por los mismos autores de UML, que se adapta especialmente bien al modelado realizado mediante dicho lenguaje. El método plantea un proceso iterativo e incremental, centrado en la arquitectura de cinco vistas explicada en el tercer apartado del tema, y conducido por los casos de uso.

Bibliografía básica

Booch, G., Rumbaugh, J., Jacobson, I. "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

Fowler, M., Scott, K. "*UML Gota a Gota*". Addison Wesley, 1999.

Jacobson, I., Booch, G., Rumbaugh, J. "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000.

Maciaszek, L. A. "*Requirements Analysis and System Design. Developing Information Systems with UML*". Addison Wesley, 2001. (Capítulos 2, 3, 4, 5y 6).

Muller, P. A. "*Modelado de Objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997. (Capítulos 1 y 3).

OMG. "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.

Rumbaugh, J., Jacobson, I., Booch, G. "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.

Bibliografía complementaria

Atkinson, C., Kühne, T., Henderson-Sellers, B. "*To Meta or Not to Meta – That Is the Question*". Journal of Object-Oriented Programming, 2000.

Barbier, F., Henderson-Sellers, B. "*Object Modelling Languages: An Evaluation and some Key Expectations for the Future*". Annals of Software Engineering, 10:67-101. 2000.

Bauer, B. "*UML Class Diagrams and Agent-Based Systems*". In the Proceedings of the International Conference on Autonomous Agents – AGENTS'01. (Montreal, Quebec, Canada, May 28 – June 1, 2001). Pages 104-105. ACM. 2001.

Berard, E. V. "*Be Careful with 'Use Cases'*". The Object Agency, Inc., 1995.

Berkem, B. "*Traceability Management from Business Processes to Use Cases with UML*". Journal of Object-Oriented Programming (JOOP), 12(5):29-34,64. September 1999.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Cantor, M. "*Object-Oriented Project Management with UML*". John Wiley and Sons, 1998.

D'Souza, D. F., Wills, A. C. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

Durán Toro, A., Bernárdez Jiménez, B. "*Metodología para el Análisis de Requisitos de Sistemas Software. (versión 2.2)*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, diciembre de 2001.

Eriksson, H. E., Penker, M. "*UML Toolkit*". John Wiley and Sons, 1998.

Fowler, M., Scott, K. "*UML Distilled. A Brief Guide to the Standard Object Modeling Language*". 2nd edition. Addison Wesley, 2000.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

Graham, I., Bischof, J., Henderson-Sellers, B. "*Associations Considered a Bad Thing*". Journal of Object-Oriented Programming (JOOP), 9(9):41-48. February 1997.

Henderson-Sellers, B. "*OML: Proposals to Enhance UML*". In Proceedings of *<<UML>>'98 Beyond the Notation* Conference. (3rd-4th June 98, Mulhouse - France). June 1998.

Henderson-Sellers, B. "*OPEN Relationships – Compositions and Containments*". Journal of Object-Oriented Programming (JOOP), 10(7):51-55,72. November/December 1997.

Henderson-Sellers, B., **Simons, T., Younessi, H.** "*The OPEN Toolbox of Techniques*". Open Series. Addison-Wesley, 1998.

Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Kobryn, C. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October 1999.

Kruchten, P. "*The 4+1 View Model of Architecture*". IEEE Software, 12(6):42-50. November 1995.

Larman, C. "*UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*". Pearson, 1999.

Letelier Torres, P., Sánchez Palma, P. "*Análisis y Diseño Orientado a Objetos Usando la Notación UML*". Notas del curso. Valencia, febrero de 2001.

Liberty, J. "*Beginning Object-Oriented Analysis and Design with C++*". Wrox Press Ltd., 1998.

López, N., Migueis, J., Pichon, E. "*Integrar UML en los Proyectos*". Gestión 2000, 1998.

Martin, J., Odell, J. J. "*Object-Oriented Methods: A Foundation, UML Edition*". 2nd Edition. Prentice Hall, 1998.

Mattinggly, L., Rao, H. "*Writing Effective Use Cases and Introducing Collaboration Cases*". Journal of Object-Oriented Programming (JOOP), 11(6):77-84,87. October 1998.

Oberg, R., Probasco, L., Ericsson, M. "*Applying Requirement Management with Use Cases*". Rational Software White Paper. Technical Paper TP505. Version 1.4. 2000.

Odell, J. J. "*Advanced Object-Oriented Analysis and Design Using UML*". Cambridge University Press. SIGS Books, 1998.

Oestereich, B. "*Developing Software with UML. Object-Oriented Analysis and Design in Practice*". Object Technology Series. Addison-Wesley, 1999.

OMG. "*Meta Object Facility (MOF) Specification. Version 1.3.1*". Object Management Group Inc. November 2001.

Pollice, G. "*Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming*". Rational Software White Paper, http://www.rational.com. 2001.

Probasco, L. "*The Ten Essentials of RUP. The Essence of an Effective Development Process*". Rational Software White Paper. Technical Paper TP-177. 2000.

Ratcliffe, M., Budgen, D. "*The Application of Use Case Definitions in System Design Specification*". Information and Software Technology 43:365-386. 2001.

Rational Software Corporation. "*Inside the Unified Modeling Language - UML*". Multimedia Educational Tool. April 1999.

Rational Software Corporation. "*Unified Modeling Language for Real-Time Systems Design*". Rational Software Corporation White Papers. http://www.rational.com. [Última vez visitado, 16/6/97]. 1997.

Rivas, E., DeSilva, D., McDaniel, T., Atkinson, C. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January 1997

Rosenberg, D., Scott, K. "*Use Case Driven Object Modeling with UML. A Practical Approach*". Object Technology Series. Addison-Wesley, 1999.

Rumbaugh, J. "*Depending on Collaborations: Dependencies as Contextual Associations*". Journal of Object-Oriented Programming (JOOP), 11(4):5-9. July/August 1998.

Rumbaugh, J. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.

Selic, B. "*Turning Clockwise: Using UML in the Real-Time Domain*". Communications of the ACM, 42(10):46-54. October 1999.

Selic, B., Rumbaugh, J. "*Using UML for Modeling Complex Real-Time Systems*". Technical Report. March 1998.

Smith, J. "*A Comparison of RUP® and XP*". Rational Software White Paper. Technical Paper TP-167. May 2001.

Spence, I., Probasco, L. "*Traceability Strategies for Managing Requirements with Use Cases*". Rational Software White Paper. Version 1.0. 1998.

Stevens, P., **Pooley, R.** "*Utilización de UML en Ingeniería del Software con Objetos y Componentes*". Addison-Wesley, 2002.

Texel, P. P., Williams, C. B. "*Use Cases Combined with Booch, OMT UML. Process and Products*". Prentice Hall, 1997.

Vadaparty, K. "*Use Cases - Basics*". Journal of Object-Oriented Programming (JOOP), 12(9). February 2000.

Warmer, J., Kleppe, A. "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley. Object Technoloy Series, 1999.

Weidenhaput, K., Pohl, K., Jarke, M., Haumer, P. "*Scenarios in System Development: Current Practice*". IEEE Software, 15(2):34–45, March/April 1998.

Whitlock, D. "*The Unified Modeling Language*". http://watson2.cs.binghamton.edu/~dwhitloc/uml/paper/part1.html. 1999.

Zhang, D. D. "*Use Case Modeling for Real-time Application*". In Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS'99. (27 - 29 January 1999, Santa Barbara, California – USA). Pages 56-64. IEEE Computer Society, 1999.

Zhao, L., Foster, T. "*Modeling Roles with Cascade*". IEEE Software, 16(5):86-93. September/October 1999.

Tema 8: ***Técnicas formales de especificación***

Descriptores

Método de especificación formal; técnica de especificación formal; invariante; precondición; poscondición; conjunto; signatura; predicado; término; operador de conjunto; operador lógico; sucesión; lenguaje de especificación formal; Z; extensiones orientadas a objetos de Z; OCL; Redes de Petri; Redes de Petri Coloreadas.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T5** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Especificaciones formales de requisitos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Aplicar las técnicas de especificación formal en el desarrollo del software.
- Discutir el papel de las técnicas de especificación formal en el contexto del desarrollo del software.
- Explicar los beneficios potenciales de los métodos formales, así como sus puntos débiles.
- Utilizar algunas técnicas de especificación formal de amplia aceptación.

Contenidos

8.1 Introducción
8.2 Base matemática de los métodos formales
8.3 Lenguajes formales de especificación
8.4 Lenguaje Z
8.5 OCL
8.6 Redes de Petri
8.7 Redes de Petri Coloreadas

Tabla 5.32. Contenidos del tema ocho del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE10 Formal methods** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Los métodos formales son objeto de controversia. Quienes los propugnan afirman que pueden revolucionar el desarrollo del software. Sus detractores piensan que resultan imposiblemente difíciles. Mientras tanto, para la mayoría de la gente los métodos formales son tan poco familiares que resulta difícil juzgar estos puntos de vista contrapuesto [Hall, 1990]. Lo que parece probado es que los métodos formales permiten incrementar la capacidad de los estudiantes para la resolución de problemas complejos [Sobel, 2000].

Con este tema no se pretende realizar un estudio exhaustivo de las técnicas formales de especificación, sino únicamente presentar una introducción a este tipo de métodos indicando en qué situaciones o en qué ámbitos de la ingeniería de requisitos está justificado su uso. La comprensión de tales métodos requiere la realización de un repaso de los principales conceptos de la teoría de conjuntos y de la sintaxis y semántica de la lógica de predicados; conceptos que serán objeto de estudio en el segundo apartado del tema.

Antes de entrar en el estudio detallado de diferentes lenguajes y técnicas de especificación formales, se dedica el tercer apartado a introducir las características de los lenguajes formales, introduciendo sus componentes primarios: sintaxis, semántica y conjunto de relaciones. En este mismo apartado del tema se establece la clasificación de los métodos en dos grandes grupos: las técnicas basadas en modelos y las basadas en propiedades o especificaciones axiomáticas. Esta clasificación se aprovecha para describir técnicas concretas como las especificaciones funcionales o lógicas correspondientes al primer tipo o las especificaciones axiomáticas encuadradas en el segundo grupo.

Posteriormente se dedican diversos apartados a describir las características esenciales de los lenguajes de mayor difusión como Z [Spivey, 1988; Spivey, 1992], CSP (*Communicating Sequential Processes*) [Hoare, 1985] o VDM (*Vienna Definition Method*) [Bjorner y Jones, 1978; Jones, 1991]. Se hace referencia también a extensiones de estos lenguajes y lenguajes específicos para sistemas orientados a objetos como Object-Z o Z++ [Stepney et al., 1992a; Stepney et al., 1992b], VDM++ [Durr y Katwijk, 1992; Durr y Plat, 1994; IFAD, 1998], OBJ [Goguen y Malcolm, 1997; Goguen et al., 2000; OBJ, 2002] o Maude [Clavel et al., 1996; Clavel et al., 1998; SRI, 2002]. El tema sigue analizando la manera de especificar restricciones formales en el contexto de UML, así se explica con el lenguaje OCL (*Object Constraint Language*) [OMG, 2001c] y se comentan algunas propuestas de formalización [Cañete et al., 1999].

Para terminar este tema se dedican dos apartados a las Redes de Petri, el primero dedicado a las Redes de Petri clásicas [Petri, 1962; Peterson, 1977; Peterson, 1981] y el segundo a las Redes de Petri Coloreadas [Jensen, 1997a; Jensen, 1997b; Jensen, 1997c].

Bibliografía básica

Harry, A. "*Formal Methods. Fact File. VDM and Z*". John Wiley & Sons, 1996. (Capítulos 1, 2, 3, 4 y 6).

Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

Lightfoot, D. "*Formal Specification Using Z*". 2nd edition. Palgrave, 2001.

OMG. "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001. (Capítulo 6).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 25).

Silva, M. "*Las Redes de Petri en la Automática y la Informática*". Editorial AC, 1985. (Capítulos 1, 2, 3 y 4).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 9).

Warmer, J., Kleppe, A. "*The Object Constraint Language. Precise Modeling with UML*". Addison-Wesley, 1999.

Bibliografía complementaria

Bowen, J. P., Hinchey, M. G. "*The Commandments of Formal Methods*". IEEE Computer, 28(4):56-63. April 1995.

Bowen, J. P., Hinchey, M. G. "*Seven More Myths of Formal Methods*". IEEE Software, 12(4):34-41. July 1995.

Cañete, J. M., Galán, F. J., Toro M. "*A Proposal for the Formalization of the OCL Language Based on Algebraic Specifications*". In the proceedings of the 4th Workshop MENHIR, F. J. García, J. M. Marqués (eds.). (Sedano, Burgos, Spain, May 1999). Pages 80-84. 1999.

Ciancarini, P., Mascolo, C. "*Using Formal Methods for Teaching Software Engineering: A Tool-Based Approach*". Annals of Software Engineering, 6:433-453. 1998.

Clarke, E. M., Wing, J. M. et al. "*Formal Methods: State of the Art and Future Directions*". ACM Computing Surveys 28(4):626-643. December 1996.

Craigen, D., Gerhart, S., Ralston, T. "*Formal Methods Reality Check: Industrial Usage*". IEEE Transactions on Software Engineering 21(2):90-98. 1995.

Diller, A. "*Z: An Introduction to Formal Methods*". Wiley, 1995.

Dillon, L. K., Sankar, S. (Guest Editors). "*Special Section on Format Methods in Software Practice*". IEEE Transactions on Software Engineering, 24(1). January 1998.

Ehrig, H., Mahr, B. "*Fundamentals of Algebraic Specification 2*". Springer-Verlag, 1990.

Fraser, M. D., Kumar, K., Vaishnavi, V. K. "*Strategies for Incorporating Formal Specifications in Software Development*". Communications of the ACM, 37(10):74-86. October 1994.

Gerhart, S. L. (Guest Editor). "*IEEE Software Special Volume on Formal Methods*". IEEE Software, 7(5). September/October 1990.

Hall, A. "*Seven Myths of Formal Methods*". IEEE Software, 7(5):11-19. September/October 1990.

Hinchey, M. G., Bowen, J. P. "*Applications of Formal Methods*". Prentice-Hall, 1995.

Jacky, J. "*The Way of Z. Practical Programming with Formal Methods*". Cambridge University Press, 1997.

Jones, C. B. "*Systematic Software Development Using VDM*". 2nd edition. Prentice-Hall, 1991.

Johnson, S. D., Alexander, W. P., Chin, S.-K., Gopalakrishnan, G. "*Report on the 21st Century Engineering Consortium Workshop. A Forum on Formal Methods Education*". http://www.cs.indiana.edu/formal-methods-education/xxiec/report.html. 1999.

van Lamsweerde, A. "*Formal Specification: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 147-159. ACM Press, 2000.

Luqi, Goguen, J. A. "*Formal Methods: Promises and Problems*". IEEE Software, 14(1): 73-85. January/February 1997.

Maibaum, T. "*Mathematical Foundations of Software Engineering: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 161-172. ACM Press, 2000.

Potter, B., Sinclair, J., Till, D. "*An Introduction to Formal Specification and Z*". Prentice-Hall, 1991.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Siddiqi, J., Morrey, I., Hibberd, R., Buckberry, G. "*Understanding and Exploring Formal Specifications*". Annals of Software Engineering, 6:411-432. 1998.

Spivey, J. M. "*Understanding Z: A specification Language and its Formal Semantics*". Cambridge University Press, 1988.

Spivey, J. M. "*The Z Notation: A Reference Manual*". Prentice-Hall, 1992.

Wing, J. M. "*A Specifier's Introduction to Formal Methods*". IEEE Computer, 23(9):8,10-23. September 1990.

Woodcock, J., Davies, J. "*Using Z. Specification, Refinement, and Proof*". Prentice Hall, 1996. On-line version http://www.comlab.ox.ac.uk/igdp/usingz.

Unidad Didáctica III: Desarrollo de sistemas complejos

Objetivo genérico

Esta unidad didáctica tiene el objetivo tratar aspectos concretos que acontecen en el desarrollo de sistemas software específicos y complejos, como son los sistemas de tiempo real, los sistemas cliente/servidor y distribuidos, los sistemas web, los sistemas interoperativos y los soportes para la toma de decisiones.

Esta unidad docente supone el **13%** del programa teórico de la asignatura Análisis de Sistemas, estando compuesta por un único tema: **Desarrollo de sistemas complejos** que se detalla a continuación.

Tema 9: ***Desarrollo de sistemas complejos***

Descriptores

Sistema de tiempo real; sistema cliente/servidor; *middleware*; IDL; CORBA; sistema interoperativo; sistema para el soporte de decisiones; *datawarehouse*; sistema web; ingeniería web; sistema de comercio electrónico; arquitectura n-capas; webapps.

Objetivos

Este tema está orientado a satisfacer el objetivos **T13** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Identificar y discutir diferentes sistemas complejos o especializados.
- Discutir el ciclo de vida y el proceso software en el contexto de los sistemas diseñados para un contexto específico.
- Seleccionar, con la apropiada justificación, las aproximaciones que darán como resultado un desarrollo y mantenimiento eficaz y eficiente de los sistemas complejos.
- Subrayar las principales técnicas asociadas con la definición, diseño e implementación de los sistemas software complejos.

Contenidos

9.1 Sistemas de tiempo real
9.2 Sistemas Cliente/Servidor (C/S)
9.3 Ingeniería Web
9.4 Sistemas Interoperativos
9.5 Sistemas para el soporte de decisiones

Tabla 5.33. Contenidos del tema nueve del programa teórico de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE12 Specialized systems development** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Resumen

El primer apartado de este tema está dedicado a los sistemas de tiempo real. Aunque el diseño de sistemas de tiempo real abarca aspectos del diseño de software convencional, también introduce un nuevo conjunto de criterios y aspectos que constituyen el objetivo de este apartado.

En los sistemas de tiempo real el diseñador debe considerar la función y el rendimiento del hardware y del software, debido a que en estos sistemas se debe responder a sucesos en el tiempo dictados tanto por el hardware como por el software. Se debe tener en cuenta, por tanto, el procesamiento de interrupciones, la tasa de transferencia de datos, las bases de datos, los sistemas operativos, los lenguajes de programación especializados, los métodos de sincronización... Todos estos aspectos se recogen a lo largo del apartado, antes de considerar el proceso de desarrollo.

En cuanto a las particularidades del proceso de desarrollo, se analizan las tres características definidas por W. Everett [Everett, 1995]: el diseño del sistema está limitado por recursos, los sistemas son compactos y funcionan a menudo sin la presencia de un usuario.

También, y en relación con la fase de análisis, se estudian las técnicas y herramientas matemáticas que facilitan el modelado y simulación que requieren los sistemas de tiempo real que permiten establecer aspectos de tiempo y tamaño y las herramientas matemáticas. Se describe el conjunto de herramientas matemáticas propuestas en [McCabe et al., 1985] que están basadas en las técnicas de análisis de flujo de datos y en las que aplica análisis de redes, teoría de colas y grafos y un modelo matemático para obtener el tiempo del sistema y el tamaño del recurso.

Se repasa y completa el estudio realizado en el tema 4 de las extensiones del Análisis Estructurado para Sistemas de Tiempo Real [Ward y Mellor, 1985; Hatley y Pirbhai, 1987]. La propuesta de P. T. Ward y S. J. Mellor incorpora un modelo del sistema que integra en un mismo esquema, Esquema de Transformación, las visiones estática, dinámica y de control mientras que D. J. Hatley y I. Pirbhai amplían la visión de los diagramas de flujo de datos creando diagramas de flujo de control y diagramas de flujo de estructura. Otro enfoque que se presenta en este tema es el de i-Logix [i-Logix, 1989; Harel et al., 1990; Harel y Naamad, 1996; Harel y Politi, 1998] que utiliza una notación que combina tres planteamientos diferentes del sistema: diagrama de actividad, diagrama de módulos y diagrama de estados. Asimismo, se hace referencia a otras técnicas [Liu y Shyamasundar, 1990; Wilson y Krogh, 1990; Zucconi, 1989], aunque ninguno de estos métodos de análisis y diseño está lo suficientemente desarrollado.

En el siguiente subapartado se examinan los principios especializados de diseño sugeridos por Kurki-Suonio [Kurki-Suonio, 1993], tales como la necesidad de definir acciones atómicas, el entrelazado, la necesidad de asumir que la historia del procesamiento es infinita o los principios de sistema cerrado y estructuración del estado.

Se ha considerado oportuno incluir un subapartado en el que se examinan la adaptación de los métodos orientados a objetos al modelado de sistemas de tiempo real, como es el caso de la

especialización de la metodología OOSE que utiliza los casos de uso para capturar los requisitos del sistema de tiempo real asociando atributos de tiempo a una secuencia, y relacionando esos requisitos con el entorno para obtener la estructura de objetos [Jacobson et al, 1993].

El segundo apartado se dedica al estudio de los sistemas cliente/servidor y sistemas distribuidos. En primer lugar se explican las características de los sistemas distribuidos situándolos en el contexto general de la arquitectura cliente/servidor. Posteriormente, se analizan las diferentes configuraciones que se pueden presentar en la distribución de componentes de software en función del grado de distribución de presentación, la lógica, la gestión de datos y las bases de datos. En los subapartados siguientes se busca un enfoque de orientación a objetos, ya que los métodos orientados a objetos se adaptan mejor al desarrollo de este tipo de sistemas. En esta línea se introduce el concepto de software intermedio o *middleware* y los servicios que puede proporcionar (comunicación, información y gestión de componentes), pasando después a analizar las características principales de las tecnologías *middleware* orientadas a objetos CORBA (*Common Object Request Broker Architecture*) [OMG, 2001a] y DCOM (*Distributed Component Object Model*) [Microsoft, 1996]. Los principios y métodos de análisis descritos para otros sistemas son igualmente aplicables al software distribuido o cliente/servidor. Éstos se desarrollan empleando las actividades de Ingeniería del Software clásicas, evolucionando el sistema a partir de un conjunto de requisitos generales para llegar a ser una colección de componentes de software ya validados que han sido implementados diferentes máquinas. Tomando como referencia esta premisa únicamente en el apartado de diseño se sugiere un enfoque que tenga en cuenta las características específicas de estos sistemas, y se aprovechan los apartados de análisis y diseño para introducir el desarrollo de software basado en componentes [Szyperski, 1998].

El tercer apartado de este tema se dedica a los sistemas web o Ingeniería Web [Pressman, 2000, c.29; Ge y Sun, 2000; Ginige y Murugesan, 2001; EC, 2001; IEEE-CS, 2001a; IEEE-CS, 2001b; WebE, 2001]. Los sistemas y aplicaciones basados en la web (*WebApps*) hacen posible que una población extensa de usuarios finales dispongan de una gran variedad de contenido y funcionalidad. La ingeniería web no es un clónico perfecto de la Ingeniería del Software, pero toma prestados muchos de los conceptos y principios básicos de ésta, dando importancia a las mismas actividades técnicas y de gestión. Existen diferencias sutiles en la forma en que se llevan a cabo estas actividades, pero la filosofía primordial es idéntica dado que dicta un enfoque disciplinado para el desarrollo de un sistema basado en computadora.

Existen diversos tipos de aplicaciones web (de información, interactivas, transaccionales, de *workflow*, entornos colaborativos, comunidades en línea, centrales comerciales y portales

[Ginige y Murugesan, 2001]) que se integran cada vez más en pequeñas y grandes compañías, y con mayor frecuencia es más importante la necesidad de construir sistemas fiables, utilizables y adaptables. Por ello, es necesario un enfoque disciplinado para el desarrollo de este tipo de sistemas software. Así en este apartado se van a presentar los pasos a seguir en la construcción sistemática de una aplicación web, el cual es un enfoque genérico que se suaviza con estrategias, tácticas y métodos especializados. El proceso de la ingeniería web comienza con la formulación del problema que pasa a resolverse con las *WebApps*. Se planifica el proyeco y se analizan los requisitos de la aplicación, entonces se lleva a cabo el diseño arquitectónico de interfaces y del navegador. El sistema se implementa utilizando lenguajes y herramientas que están asociados a la web, y entonces comienzan las pruebas. Dado que este tipo de aplicaciones se encuentra en constante evolución, deben establecerse los mecanismos para el control de configuraciones, garantía de calidad y soporte continuado.

La realización de aplicaciones web sin el rigor y método adecuado está provocando unas aplicaciones web pobres y difíciles de mantener y evolucionar, que dan lugar a una crisis de la web, similar en algunos términos a la famosa crisis del software [Ginige y Murugesan, 2001].

Bajo el epígrafe de sistemas cooperativos se encuentra el estudio de un tipo especial de sistemas muy relacionados con los anteriores, ya que se trata de sistemas formados por componentes heterogéneos, autónomos y gestionados localmente, los cuales deben cooperar para proporcionar servicios complejos [Finkelstein, 1998]. Dichos sistemas son generalmente distribuidos y tienen restricciones no funcionales significativas en su operación. El metamodelado es esencial en estos sistemas para gestionar la evolución y adaptación de los mismos. El propósito del metamodelo es definir un conjunto de conceptos clave que se usarán en las fases de diseño y modelado. En este apartado se explican los niveles de modelado de la arquitectura ISO IRDS (*Information Resources Dictionary Standard*) (Standard ISO 10027) [ISO/IEC, 1990]. El estudio de los sistemas interoperativos termina con la descripción de un método de desarrollo de sistemas basado en el modelado contextual de procesos [Pohl et al., 1998].

El último apartado del tema se dedica a los sistemas de apoyo a las decisiones. Estos sistemas poseen muchas características que los diferencian de otros sistemas de información tradicionales. Un sistema de apoyo a las decisiones no automatiza simplemente transformaciones aplicadas sobre los datos, sino que da soporte al proceso de toma de decisiones por medio de presentaciones especiales de la información y el uso de técnicas que permiten extraer información no explícita en conjuntos masivos datos (descubrimiento de patrones, tendencias...). Las técnicas más utilizadas son las asociadas con los sistemas de

gestion de bases de datos como son las de *datawarehouse* o las técnicas de minería de datos [Michalski et al., 1997]. La técnica a aplicar va a depender del tipo de decisión a tomar, algunos de esos tipos se analizan también en este apartado del tema.

Bibliografía básica

Hatley, D. J., Pirbhai, I. A. "*Strategies for Real-Time System Specification*". Dorset House Pub. Co., 1987.

Kendall, K. E., Kendall, J. E. "*Análisis y Diseño de Sistemas*". Prentice Hall, 1997. (Capítulo 12).

Krämer, B., Papazoglou, M., Schmidt, H. W. (Editors). "*Information Systems Interoperativity*". RSP-John Wiley and Sons Inc., 1998.

Presssmasn, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill, 1998. (Capítulo 15).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 12, 27, 28 y 29).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 11 y 13).

Ward, P. T., Mellor, S. J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

Bibliografía complementaria

Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O., Spiteri, M. "*Generic Support for Distributed Applications*". IEEE Computer, 33(3):68-76. March 2000.

Bihari, T. E., Gopinath, P. "*Object-Oriented Real-Time Systems: Concepts and Examples*". IEEE Computer, 25(12):25-32. December 1992.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.

Conallen, J. "*Modeling Web Application Architectures with UML*". Rational Software White Paper. June 1999.

Coulouris, G., Dollimore, J., Kindberg, T. "*Sistemas Distribuidos. Conceptos y Diseño*". Addison Wesley, 2001.

Chin, R. S., Chanson, S. T. "*Distributed Object-Based Programming Systems*". ACM Computing Surveys, 23(1):91-124. March 1991.

D'Souza, D. F., Wills, A. C. "*Objects, Components and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison Wesley, 1999.

Farley, J. "*Java Distributed Computing*". O'Reilly, 1998.

García Peñalvo, F. J., González González, J., Álvarez Navia, I., Moreno García, Mª N., Curto Diego, B., Moreno Rodilla, V. "*Fundamentos para el Desarrollo de Aplicaciones*

Distribuidas Basadas en CORBA". Informe Técnico (DPTOIA-IT-2002-001), Universidad de Salamanca (España). http://tejo.usal.es/inftec/DPTOIA-IT-2002-001.pdf. Febrero, 2002.

Geihs, K. "*Middleware Challenges Ahead*". IEEE Computer, 34(6):24-31. June 2001.

Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. January-March 2001.

Goguen, J. A., Lin, K. "*Web-Based Support for Cooperative Software Engineering*". Annals of Software Engineering, 12:167-191. 2001.

Gomaa, H. "*Designing Concurrent, Distributed, and Real-Time Applications with UML*". Addison-Wesley, Object Technology Series, 2000.

Hendrickson, E., Fowler, M. (Guest Editors). "*Special Issue on Engineering Internet Software*". IEEE Software, 19(2). March/April 2002.

Henning, M., Vinoski, S. "*Programación Avanzada en CORBA con C++*". Addison Wesley, 2002.

Hofmeister, C., Nord, R., Soni, S. "*Applied Software Architecture*". Addison-Wesley, Object Technology Series, 2000.

IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part I*". Vol.8, N1, January-March 2001.

IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part II*". Vol.8, N2, April-June 2001.

Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Lewis, T. G. "*Where Is Client/Server Software Headed*". IEEE Computer, 28(4): 49-55. April 1995.

Michalski, R. S.; Bratko, I., Kubat, M. "*Machine Learning and Data Mining. Methods and Applications*". John Wiley and Sons, 1997.

Mowbray, T. J., Malveau, R. C. "*CORBA Design Patterns*". John Wiley & Sons, 1997.

Object Management Group. "*CORBA 2.6 Specification*". Document formal/01-02-35. 2001.

Osborne, W. M., Chifofsky, E. J. (Guest Editors). "*Maintenance, Reverse Engineering, and Design Recovery Special Issue*". IEEE Software, 7(1). January 1990.

Pastor, O., Abrahão, S., Fons, J. "*Building E-Commerce Applications from Object-Oriented Conceptual Models*". SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-commerce, 2(2):28-36. Spring 2001.

Paternò, F. "*Model-Based Design and Evaluation of Interactive Applications*". Springer-Verlag, 2000.

Pérez Cañellas, J. "*Servicios Web (I)*". Sólo Programadores, Nº85:40-45. Enero, 2002.

Rational Software, Context Integration. "*Building Web Solutions with the Rational Unified Process: Unifying the Creative Design Process and the Software Engineering Process*". Rational Software and Context Integration White Paper. 1999.

Schmidt, D., Stal, M., Rohnert, H., Buschmann, F. "*Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects – Volume 2*". John Wiley & Sons, 2000.

Sevilla Ruiz, D. "*Aplicaciones Distribuidas en Internet/Intranets: De los Sockets a los Objetos Distribuidos*". Proyecto de Final de Carrera. Universidad de Murcia. 1998.

Siegel, J. (Editor). "*CORBA 3 Fundamentals and Programming*". 2nd edition. Jonh Wiley & Sons, 2000.

Stankovic, J. A. "*Misconceptions about Real-Time Computing. A Serious Problem for Next-Generation Systems*". IEEE Computer, 21(10):10-19. October 1988.

Szyperski, C. "*Component Software – Beyond Object-Oriented Programming*". Addison-Wesley, 1998.

Tari, Z., Bukhres, O. "*Fundamentals of Distributed Object Systems. The CORBA Perspective*". John Wiley & Sons, 2001.

Vaughan-Nichols, S. J. "*Web Services: Beyond the Hype*". IEEE Computer, 35(2):18-21. February 2002.

Watson, H., Sprague, R. (Editors). "*Decision Support Systems: Putting Theory into Practice*". Englewoods Cliffs, N.J. Prentice-Hall, 1993.

Unidad Didáctica IV: Evolución del software

Objetivo genérico

Esta cuarta y última unidad didáctica del programa de teoría de Análisis de Sistemas se dedica a uno de los escollos más grandes con los que se encuentra el desarrollo de sistemas de información automatizados, el mantenimiento de los mismos.

El mantenimiento es fase que más recursos económicos consume dentro del desarrollo de un sistema software. El 60% de las inversiones efectuadas por una organización de desarrollo se dedican al mantenimiento, y ese porcentaje sigue creciendo a medida que se produce más software [Hanna, 1993].

El mantenimiento es algo más que la corrección de errores. El mantenimiento se puede definir describiendo las cuatro actividades que se emprenden en un programa para su utilización: mantenimiento correctivo, adaptativo, perfectivo y preventivo o reingeinería. Tan sólo el 20% del esfuerzo de mantenimiento se dedica a la corrección de errores, el 80% restante se dedica a adatar los sistemas existentes a los cambios de su entorno externo, a efectuar las mejoras solicitadas por los usuarios y a rehacer la ingeniería de las aplicaciones para su posterior utilización. Cuando se considera que el mantenimiento abarca todas estas actividades, se justifica por qué absorbe tanto esfuerzo.

El problema y la realidad del mantenimiento del software se recogen en esta cita de Osborne y Chifofsky: “Gran parte del software del que se depende actualmente tiene por término medio entre diez y quince años de antigüedad. Aun cuando estos programas se crearon empleando las mejores técnicas de diseño y codificación conocidas en su época (y la mayoría no lo fueron), se crearon cuando el tamaño de los programas y el espacio de almacenamiento eran las preocupaciones principales. A continuación, se trasladaron a las nuevas plataformas, se ajustaron para adecuarlos a cambios de máquina y sistemas operativos y se mejoraron para satisfacer nuevas necesidades del usuario; y todo esto se hizo sin tener en cuenta la arquitectura global. El resultado son unas estructuras muy mal diseñadas, una mala codificación, una lógica inadecuada, y una escasa documentación de los sistemas de software que ahora se pide que se mantengan en marcha.” [Osborne y Chifofsky, 1990].

Esta unidad supone el **5%** del programa teórico de la asignatura Análisis de sistemas, estando compuesta por un único tema: **Evolución y mantenimiento del software** que se detalla a continuación.

Tema 10: ***Evolución y mantenimiento del software***

Descriptores

Mantenimiento del software; barrera de mantenimiento; evolución del software; mantenimiento correctivo; mantenimiento adaptativo; mantenimiento perfectivo; mantenimiento preventivo; reingeniería; ingeniería inversa.

Objetivos

Este tema está orientado a satisfacer el objetivo **T12** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Identificar los principales problemas asociados con la evolución del software, explicando su impacto en el ciclo de vida del software.
- Presentar los diferentes tipos de mantenimiento y su aplicación en proyectos reales.
- Discutir los retos del mantenimiento de sistemas legados y la necesidad de la ingeniería inversa.
- Establecer cuándo es conveniente y cómo llevar a cabo un plan de reingeniería.
- Discutir las relaciones existentes entre la evolución y la reutilización del software.

Contenidos

10.1 Introducción
10.2 Actividades de mantenimiento
10.3 El proceso de mantenimiento
10.4 Problemas del mantenimiento
10.5 Costes de mantenimiento
10.6 Herramientas y técnicas
10.7 Ingeniería inversa y reingeniería

Tabla 5.34. Contenidos del tema diez de teoría de Análisis de Sistemas

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE7 Software evolution** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Maintenance** [Pigoski, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El interés de este tema se centra en la descripción del proceso de mantenimiento de software, en discutir los factores que afectan a su coste, en el papel de las medidas en la predicción de la facilidad de mantenimiento y en estudiar los procesos de ingeniería inversa y reingeniería.

En la introducción del tema se examina la forma en que ha progresado este proceso desde que M. M. Lehman y L. A. Belady comenzaran su investigación sobre la evolución de los sistemas de software que les ha conducido a la formulación de un conjunto de leyes de evolución [Lehman y Belady, 1985; Lehman, 1997]. Se introduce también en este apartado la definición de esta última fase del ciclo de vida, sus aspectos más importantes y la clasificación realizada por E. B. Swanson [Swanson, 1976] que propone tres categorías (correctivo, adaptativo y perfectivo) reflejadas en la norma ISO/IEC 14764 [ISO/IEC, 1997] y ampliadas con una cuarta categoría (mantenimiento preventivo) presentada en el estándar IEEE 1219 [IEEE, 1998]. Puede observarse que, mientras que el cambio tecnológico afecta indirectamente a los sistemas software, el entorno de trabajo y los usuarios lo hacen directamente [Hybertson et al., 1997], produciendo demandas de mantenimiento adaptativo y perfectivo respectivamente.

En el segundo apartado se consideran las actividades específicas del mantenimiento que difieren de las del proceso de desarrollo como el estudio del código fuente, el análisis de impacto para determinar el coste del cambio, análisis de las peticiones de cambio, identificación de componentes afectados y actividades de soporte, entre las que se encuentran la gestión de configuraciones, aseguramiento de la calidad, verificación y validación, migración... El apartado termina con la descripción de la actividad de planificación del mantenimiento.

Posteriormente, se estudia el proceso de mantenimiento tomando como base el papel que éste juega en los niveles de madurez del modelo CMM (*Capability Maturity Model*) [Paulk et al., 1993a; Paulk et al., 1993b], así como en otros modelos de proceso [Takang y Grubb, 1997] y los diferentes estándares que contemplan la evolución y mantenimiento del software.

En este tema, además, se discuten los problemas que conlleva la labor de mantenimiento [Schneidewind, 1985] derivados en parte de la falta de sistematización de este proceso y de la escasez de herramientas de apoyo, se analizan los factores técnicos y no técnicos que afectan a los costes y la estimación de los mismos.

En el apartado seis se hace referencia a las herramientas automatizadas que pueden contribuir a la mejora del mantenimiento como las herramientas de prueba, gestión de configuraciones, medición y documentación, trazabilidad de requisitos... Se tratan igualmente en este apartado técnicas de comprensión de programas, análisis de impacto, reingeniería e ingeniería inversa. Estas dos últimas se estudian con más detalle en el último apartado [Jacobson et al., 1994; Jacobson et al., 1997].

Bibliografía básica

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 16).

Piattini, M., Villalba, J., Ruiz, F., Bastanchury, T., Polo, M., Martínez, M. Á., Nistal, C. "*Mantenimiento del Software. Modelos, Técnicas y Métodos para la Gestión del Cambio*". Ra-ma, 2000. (Capítulos 1, 2, 3, 4 y 6).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 2 y 30).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 26, 27, 28 y 29).

Bibliografía complementaria

Bardou, L. "*Mantenimiento y Soporte Logístico de los Sistemas Informáticos*". Ra-ma, 1997.

Bennett, K. H., Rajlich, V. T. "*Software Maintenance and Evolution: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 73-87. ACM Press, 2000.

Carrera, L. U., Dolado, J. "*Mantenimiento del Software desde una Perspectiva de Proceso*". En *Medición para la Gestión en la Ingeniería del Software*. Dolado, J., Fernández, L. (coordinadores). Páginas 241-250. Ra-ma, 2000.

Carsí, J. A. "*OASIS como Marco Conceptual para la Evolución de Software*". Tesis Doctoral. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. 1999.

Cifuentes, C., Fitzgerald, A. "*The Legal Status of Reverse Engineering of Computer Software*". Annals of Software Engineering, 9:337-351. 2000.

IEEE-CS. "*IEEE Software Special Volume on Legacy Systems*". IEEE Software, 12(1). January 1995.

Jacobson, I. Ericsson, M., Jacobson, A. "*The Object Advantage-Business Process Reenginering with Object Technology*". Addison Wesley, 1994.

Jacobson, I., Griss, M., Patrik, J. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press, 1997.

Lehman, M. M. "*Laws of Software Evolution Revisited*". Academic Press Inc., 1997.

Martínez Hernández, A. "*Medidas para Asegurar la Mantenibilidad de Entornos de Cuarta Generación*". Tesis Doctoral. Universidad de Castilla-La Mancha. Escuela Superior de Informática. Ciudad Real. 2001.

Osborne, W. M., Chifofsky, E. J. (Guest Editors). "*IEEE Software Special Volume on Maintenance, Reverse Engineering, and Design Recovery*". IEEE Software, 7(1). January 1990.

Pigoski, T. M. "*Practical Software Maintenance. Best Practices for Managing Your Investment*". John Wiley & Sons, 1996.

Pigoski, T. M. "*Software Maintenance*". Chapter 6 in [Abran et al., 2001].

Polo, M. "*MANTEMA: Una Propuesta Metodológica y Metrológica para el Mantenimiento del Software*". Tesis Doctoral. Universidad de Castilla-La Mancha. Escuela Superior de Informática. Ciudad Real. 2000.

Polo, M., Piattini, M. "*Elaboración de una Metodología para el Mantenimiento de Software*". IV Jornadas de Ingeniería del Software y Bases de Datos - JISDB'99. P. Botella, J. Hernández, F. Saltor (editores). (Cáceres, 24-26 de noviembre de 1999). Páginas 219-230. 1999.

Ruiz, F., Piattini, M., Polo, M., Calero, C. "*Maintenance Types in the MANTEMA Methodology*". International Conference on Enterprise Information Systems. Portugal, 1999.

Schach, S. R. "*Object-Oriented and Classical Software Engineering*". 5th edition. McGraw-Hill, 2002.

Schneidewind, N. F., Ebert, C. (Guest Editors). "*IEEE Software Special Volume on Managing Legacy Systems*". IEEE Software, 15(4). July/August 1998.

Smith, D. S. "*Designing Maintainable Software*". Springer-Verlag, 1999.

Sneed, H. M. "*Encapsulation of Legacy Software: A Technique for Reusing Legacy Software Components*". Annals of Software Engineering, 9:293-313. 2000.

Waters, R. G., Chikofsky, E. (Guest Editors). "*Communications of the ACM Special Volume on Reverse Engineering*". Communications of the ACM, 37(5). May 1994.

5.3.2 Programa de la parte práctica

La parte práctica de la asignatura *Análisis de Sistemas* está dirigida a satisfacer aquellos objetivos prácticos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**P1, P2** y **P4**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a] (incluido en el Apéndice A de este Proyecto Docente).

P1	Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.
P2	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
P4	Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Tabla 5.35. Objetivos del programa de prácticas de la asignatura Análisis de Sistemas

5.3.2.1 Consideraciones iniciales

El objetivo fundamental de las prácticas de la asignatura *Análisis de Sistemas* es el análisis y diseño de sistemas software, aplicando los conceptos presentados en la teoría, apoyados éstos en herramientas CASE.

Se considera que las 30 horas, que equivalen a los tres créditos de la parte práctica, deben repartirse en transmisión de contenidos en laboratorio y práctica no guiadas.

Las prácticas de transmisión de contenidos en laboratorio se reducen al mínimo de horas necesarias para presentar los entornos utilizados en las prácticas, así como los enunciados de las prácticas obligatorias.

En las prácticas no guiadas los alumnos deberán desarrollar las prácticas obligatorias, teniendo el respaldo del profesor para las dudas que puedan surgir en la realización de los mismos.

El número de prácticas obligatorias a realizar será dos, una centrada en el paradigma estructurado y otra en el paradigma objetual. Cada práctica se puede realizar en grupos de dos alumnos. Al finalizar las mismas, los alumnos deberán entregar la documentación correspondiente, que servirá para su evaluación. El documento podrá ser elaborado

conjuntamente por ambos alumnos, pero la defensa de las prácticas se realizará de forma individualizada.

En la página web de la asignatura se encuentra la descripción de las prácticas que se van a realizar, así como el material necesario para llevarlas a cabo y enlaces a partir de los cuales se puede encontrar información adicional.

Para el mejor aprovechamiento de las prácticas, éstas se llevan a cabo una vez cada semana durante dos horas consecutivas, durante el segundo cuatrimestre (la parte teórica se desarrolla durante tres horas a la semana durante el primer cuatrimestre, y durante una hora en el segundo cuatrimestre).

5.3.2.2 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte práctica de esta asignatura se ha organizado en dos prácticas, tal y como se muestra en la Tabla 5.36.

En la Tabla 5.37 se presenta la correspondencia existente entre el programa de la parte práctica y los objetivos prácticos perseguidos en esta asignatura.

Laboratorio (30 Horas)
Práctica 1. Análisis y diseño estructurado (15 Horas)
Práctica 2. Análisis y diseño orientado a objetos (15 Horas)

Tabla 5.36. Estructura del programa de prácticas de la asignatura Análisis de Sistemas (3 créditos)

Elemento Docente	Objetivos
Práctica 1	P1, P4, H1, H2, H3, H4
Práctica 2	P2, P4, H1, H2, H3, H4

Tabla 5.37. Correspondencia entre el temario de prácticas y los objetivos prácticos de la asignatura

Evaluación de la parte práctica

La forma principal de evaluar la parte práctica de esta asignatura es mediante la realización de dos prácticas obligatorias, cuyos requisitos deben obtenerse de *usuarios* o *clientes* reales.

La nota será la media de las calificaciones obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individual, aunque el trabajo hay sido realizado en pareja.

En la Figura 5.13 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota de la práctica obligatoria y los informes de los talleres, realizados voluntariamente.

Si (Teoría ≥ 4,75) **y** (***Práctica*** ≥ 5.0)
    Nota Final = ((Teoría*0,6) + (***Práctica****0,3)) * 10/9 + Nota trabajos
Sino
    Ø
Fin si

Figura 5.13. Influencia de la nota en la parte práctica en la nota final de Análisis de Sistemas

Bibliografía básica de referencia

La siguiente lista refleja los títulos recomendados para afrontar la parte práctica de la asignatura. Algunos de ellos ya se recomendaron como bibliografía básica general de la asignatura, mientras que otros van más encaminados a ser referencia para uno de los entornos utilizados, aunque los alumnos utilizarán más habitualmente la ayuda en línea disponible.

- **Abbey, M., Corey, M. J.** "*Oracle8. Guía de Aprendizaje*". Oracle Press – Osborne/McGraw-Hill, 1998.
- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **Corey, M. J., Abbey, M., Dechichio, D. J., Abramson, I.** "*Puesta a Punto de Oracle 8*". Oracle Press – Osborne/McGraw-Hill, 1998.
- **Date, C. J.** "*Introducción a los Sistemas de Bases de Datos*". 7ª Edición, Addison-Wesley, 2001.
- **Dorsey, P., Hudicka, J. R.** "*Oracle8. Design Using UML Object Modeling*". Oracle Press – Osborne/McGraw-Hill, 1999.
- **Dorsey, P., Koletzke, P.** "*Manual de Oracle Designer/2000*". Oracle Press – Osborne/McGraw-Hill, 1997.
- **Durán, A., Bernárdez, B.** "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001.
- **Ministerio de Administraciones Públicas.** "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.
- **Morriseau-Leroy, N., Solomon, M. K., Basu, J.** "*Oracle8i Java Component Programming with EJB, CORBA, and JSP*". Oracle Press – Osborne/McGraw-Hill, 2000.
- **Muller, R. J.** "*Manual de Oracle Developer/2000*". Oracle Press – Osborne/McGraw-Hill, 1998.

- **Object Management Group.** "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.
- **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.
- **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana. 1993.

5.3.2.3 Desarrollo comentado del programa

En el programa de práctica se distinguen dos prácticas destinadas al análisis y diseño de sistemas software. La primera está centrada en el paradigma estructurado y la segunda en la orientación a objetos.

Práctica 1. Análisis y diseño estructurado

En la primera práctica de la asignatura el estudiante aplica un método estructurado para realizar el análisis y diseño de un sistema elegido por él mismo. Los conceptos teóricos que requiere los ha adquirido previamente en la asignatura *Ingeniería del Software* de primer ciclo y en la asignatura que se presenta, *Análisis de Sistemas*, durante el primer cuatrimestre.

Las cuatro primeras horas de laboratorio se dedicarán a explicar el funcionamiento de las herramientas CASE a utilizar. Las horas restantes se destinan al desarrollo tutelado del proyecto elegido. Las herramientas son **REM** (*REquirements Management*) [Durán, 2002], **DESIGNER 2000 v. 2.1** y **DEVELOPER 2000 v2.1** de **Oracle Corporation** (http://www.oracle.com/) de las cuales se dispone licencia para 25 usuarios. Ambas constan de una parte cliente que funciona en entorno Windows y se instala en cada puesto de trabajo del laboratorio y un repositorio central instalado en una plataforma UNIX (Origin 200 de Silicon).

Con la ayuda de tales herramientas los estudiantes deben elaborar la siguiente documentación:

1. Definición del proyecto
   1.1. Descripción general del proyecto
   1.2. Identificación de unidades
2. Catálogo de requisitos
   2.1. Objetivos
   2.2. Requisitos de información
   2.3. Requisitos funcionales
   2.4. Requisitos no funcionales

3. Construcción del modelo de procesos del sistema
    3.1. Representación jerárquica de subsistemas y funciones
    3.2. Representación del modelo de procesos mediante DFD
    3.3. Descripción de cada función y las subfunciones de que consta
        3.3.1. Modo de acceso a entidades
        3.3.2. Características de la subfunción
        3.3.3. Frecuencia de ejecución
4. Construcción del modelo de datos
    4.1. Identificación de entidades atributos y asociaciones
    4.2. Construcción del diagrama entidad-relación inicial
    4.3. Normalización y representación del modelo lógico final
5. Documentación de las relaciones entre el modelo de proceso y el modelo de datos mediante técnicas matriciales
6. Diseño de la aplicación
    6.1. Diagramas de módulos
    6.2. Diseño de la base de datos

Esta práctica representa el **50%** del tiempo dedicado a las prácticas de la asignatura.

Práctica 2. Análisis y diseño orientado a objetos

La segunda práctica consiste en el modelado de un sistema orientado a objetos utilizando el lenguaje UML.

En este caso, los alumnos, además de elegir el sistema a desarrollar, eligen la herramienta con la que van a trabajar. En este sentido, se le proponen varias herramientas de libre disposición entre las que se encuentra **Rational Rose** (http://www.rational.com), herramienta creada por la empresa que ha desarrollado UML. Pueden utilizar la versión 4.0, que es una versión limitada en cuanto al número de elementos que permite modelar, pero sin límite de tiempo para su uso, o la versión actual que se distribuye en la web de Rational Software Corporation, que no tiene limitaciones funcionales ni de tamaño de los modelos, pero únicamente tiene validez durante 30 días.

Otras herramientas o entornos que se les proponen son los siguientes: **ArgoUML** (http://argouml.tigris.org/), el entorno de desarrollo de programas Java **Bluette** (http://www.bluette.com/), que integra una herramienta de modelado, **Plastic**, capaz de realizar diagramas de clases y de objetos en UML, **ADAM CASE v1.1** [García et al., 2000b; García et al., 2001b] o **Left CASE v1.0** [García et al., 2001a].

A estas herramientas se le puede añadir **REM** (*REquirements Management*) [Durán, 2002] para la documentación de requisitos.

El documento que deben entregar al finalizar la práctica debe presentar la siguiente estructura:

1. Definición del proyecto
    1.1. Descripción general del proyecto
    1.2. Identificación de unidades
2. Catálogo de requisitos
    2.1. Objetivos
    2.2. Requisitos de información
    2.3. Requisitos funcionales
    2.4. Requisitos no funcionales
3. Vista de casos de uso
4. Vista de interacción
5. Vista estática
6. Diseño de la aplicación
    6.1. Diseño de la base de datos

Esta práctica representa el **50%** del tiempo dedicado a las prácticas de la asignatura.

5.4 Programa de la asignatura Administración de Proyectos Informáticos

La asignatura *Administración de Proyectos Informáticos* es la segunda de las asignaturas que recogen los créditos troncales de la materia Ingeniería del Software en la titulación de Ingeniero Informático (2º ciclo), junto a la asignatura *Análisis de Sistemas*. Su propósito fundamental es que los alumnos de esta titulación adquieran los conocimientos necesarios para desarrollar actividades relacionadas con la gestión y el control de los proyectos informáticos. Dichas tareas comienzan antes del inicio de cualquier actividad técnica y continúan a lo largo de todo el ciclo de vida del software. El objetivo final de dichas actividades es obtener un producto software final de calidad y fiable desarrollado mediante un proceso eficiente y productivo.

Asignatura	Administración de Proyectos Informáticos (troncal)
Créditos	6T + 3P
Estudios	Ingeniería Informática (2º ciclo)
Plan	B.O.E de 1-7-1999
Curso	2º
Cuatrimestre	1º y 2º (anual)
Responsable	María N. Moreno García (mmg@usal.es)
Página web de la asignatura	http://lisisu02.usal.es/~mmoreno/api.html

Tabla 5.38. Ficha de la asignatura Administración de Proyectos Informáticos

Los objetivos específicos que se pretenden alcanzar son los siguientes:

- Adquisición de conocimientos sobre los fundamentos teóricos de las técnicas de gestión de proyectos.
- Capacidad de uso de herramientas automatizadas.
- Valoración de la importancia del trabajo en equipo.
- Desarrollo de capacidades de comunicación.

Esta asignatura se imparte durante los cuatrimestres tercero y cuarto del segundo curso, dentro del Plan de Estudios del B.O.E de 1-7-1999 [BOE, 1999].

Esta asignatura está dotada de ***9 créditos, 6 teóricos*** y ***3 prácticos***. Se orienta hacia los conocimientos y capacidades prácticas necesarias para llevar a cabo las actividades de Ingeniería del Software relacionadas con la gestión de proyectos informáticos.

Para la elaboración del programa se ha optado por una estrategia sustentada en los siguientes puntos:

1. Se tiene como prerrequisito que el alumno tenga unos conocimientos teóricos y prácticos de los fundamentos de la Ingeniería del Software.
2. En el caso de la Ingeniería Informática (2º ciclo) los conocimientos requeridos se adquieren en la asignatura *Análisis de Sistemas*, del primer curso, así como en la asignatura *Ingeniería del Software*, cursada en la titulación de primer ciclo Ingeniería Técnica en Informática de Sistemas.
3. La gestión inadecuada es una de las causas más importantes del fracaso de los proyectos informáticos. En pro de conseguir una gestión efectiva se van a estudiar técnicas de medición, estimación, planificación, aseguramiento de la calidad y gestión de configuraciones.
4. Aunque la asignatura se fundamenta en la transmisión de conocimiento técnico, no se puede (ni se quiere) perder la oportunidad de hacer que los alumnos desarrollen y potencien otras habilidades más generales, pero de importancia capital en su futuro como profesionales: *acostumbrarse a consultar bibliografía (especialmente en inglés), haciendo hincapié en la importancia que tiene leer con cuidado para sintetizar, escribir y modelar de forma adecuada* [Jackson, 1995]; *escribir documentos técnicos que describan los diferentes elementos software que se crean a lo largo de un proyecto* [Deveaux et al., 1999] *cuidando los estándares de documentación* [Gersting, 1994; McCauley et al., 1996], *sin que ello signifique dar la espalda a las reglas gramaticales y de estilo que ofrece un lenguaje tan rico como el castellano* [Vaquero, 1999]; *desarrollar una capacidad de comunicación oral adecuada* [McDonald y McDonald, 1993; Fell et al., 1996].
5. Llegar a un equilibrio entre la teoría y la práctica [Glass, 1996], de forma que una base teórica bien establecida sea el fundamento adecuado para la aplicación práctica de la Ingeniería del Software.

5.4.1 Programa de la parte teórica

La parte teórica de la asignatura *Administración de Proyectos Informáticos* está orientada a satisfacer aquellos objetivos teóricos que, habiendo sido identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y al contexto docente de la asignatura (**T10, T11** y **T12**), además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a].

T10	Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.
T11	Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.
T12	Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

Tabla 5.39. Objetivos del programa de teoría de la asignatura Administración de Proyectos Informáticos

5.4.1.1 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de seis temas, organizados en cuatro unidades docentes, tal y como se puede apreciar en la Tabla 5.40

La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve presentación del temario de teoría y práctica de la misma. Los alumnos ya cuentan con el programa de la asignatura, resumido en la guía académica que se les entrega con la matrícula [GAFC-USAL, 2001], y totalmente actualizado en la página web de la asignatura (http://lisisu02.usal.es/~mmoreno/api.html).

Presentación de la asignatura (1 Hora)
Tema 1. Visión general de la administración de proyectos (6 horas)
Tema 2. Medición del software (11 horas)
Tema 3. Métodos de estimación y gestión del riesgo (10 horas)
Tema 4. Planificación temporal de proyectos (12 horas)
Tema 5. Gestión y aseguramiento de la calidad (15 horas)
Tema 6. Gestión de configuraciones (5 horas)

Tabla 5.40. Estructura del programa de teoría de la asignatura Administración de Proyectos Informáticos

La Figura 5.14 refleja el reparto porcentual de los diferentes temas. En la Tabla 5.41 se presenta la correspondencia existente entre el programa de teoría y los objetivos teóricos perseguidos en esta asignatura.

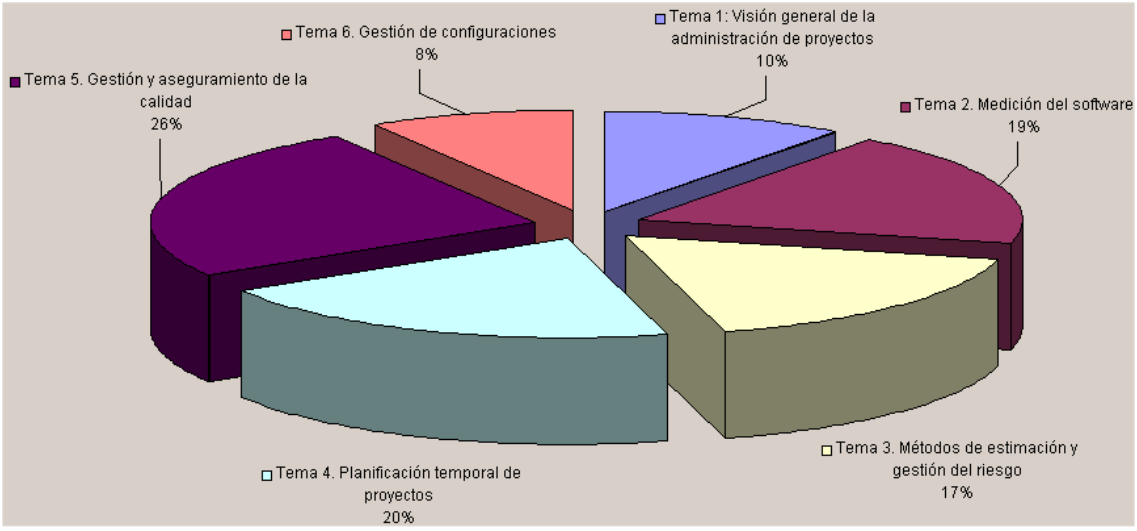


Figura 5.14. Reparto de las horas de teoría de Administración de Proyectos Informáticos entre los temas

Elemento Docente	Objetivos
Tema 1	T10, T11, T12, H3
Tema 2	T11, H3
Tema 3	T10, H3
Tema 4	T10, H3
Tema 5	T10, T11, H3
Tema 6	T12, H3

Tabla 5.41. Correspondencia entre el temario teórico y los objetivos teóricos de la asignatura

Estos temas pueden verse completados por otras actividades complementarias, que se llevarán a cabo dependiendo de diversos factores, entre los que cabe citar *la disponibilidad de tiempo para hacerlas efectivas* y *el interés y colaboración de los propios alumnos*. Las actividades a realizar pueden caer dentro de alguno de los siguientes grupos:

- Seminarios impartidos sobre temas específicos.
- Trabajos voluntarios realizados por los alumnos.
- Conferencias invitadas.
- *Workshop* de trabajos realizados por los alumnos sobre temas de la asignatura.

Desarrollo de las clases de teoría

Las clases de teoría se desarrollan de forma similar a las de las asignaturas ya presentadas, esto es, utilizando una variante de la clase magistral, donde el profesor se apoya en un retroproyector y en la pizarra para el desarrollo de los siete temas de los que se compone el temario.

Los alumnos cuentan de antemano con las transparencias de los temas para que no tengan que tomar apuntes en el sentido clásico del término, y puedan prestar atención a las explicaciones, completando las transparencias con las notas que cada uno crea oportuno. Además, permite que el alumno que lo desee intervenga en cualquier momento para hacer una pregunta o solventar una duda y no, como en el dictado de apuntes, para pedir que se repita una frase.

Evaluación de la parte teórica

La forma principal de evaluar la parte teórica de esta asignatura es mediante la realización de una prueba escrita. En la Figura 5.15 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura.

Si (***Teoría*** ≥ 4,75) **y** (Práctica ≥ 5.0)
    Nota Final = ((***Teoría****0,6) + (Práctica*0,3)) * 10/9 + Nota trabajos
Sino
    Ø
Fin si

Figura 5.15. Influencia de la nota de la parte teórica en la nota final de Administración de Proyectos Informáticos

La parte de teoría se evalúa mediante un examen escrito, formado por cuestiones de respuesta corta y pequeños supuestos prácticos, hadiendo un solo examen por convocatoria, esto es, un examen final en el mes de junio y un examen final extraordinario en el mes de septiembre.

Los trabajos voluntarios presentados obtendrán una puntuación entre 0,5 y 1,5 puntos, que se sumarán a la nota conseguida en los apartados de teoría y de prácticas, siempre y cuando en éstos se haya obtenido la calificación mínima exigida.

Bibliografía básica de referencia

La lista de títulos que se les propone como bibliografía básica de consulta es:

- **Burnett, K.** "*The Project Management Paradigm*". Springer-Verlag, 1998.
- **Cos, M.** "*Teoría General del Proyecto*". Síntesis, 1997.
- **Fenton, N. E., Pfleeger, S. L.** "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.
- **Humphrey, W. S.** "*Introducción al Proceso Software Personal*". Addison-Wesley, 2001.

- **Lewin, M. D.** "*Better Software Project Management: A Primer for Success*". John Wiley & Sons, 2002.
- **McConnell, S.** "*Desarrollo y Gestión de Proyectos Informáticos*", McGraw-Hill 1997.
- **McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., Hall, F.** "*Practical Software Measurement. Objective Information for Decision Makers*". Addison-Wesley, 2002.
- **Ministerio de Administraciones Públicas.** "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.
- **Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.
- **Pfleeger, S. L.** "*Software Engineering. Theory and Practice*". Prentice-Hall, 1998.
- **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.
- **Puig, J.** "*Proyectos Informáticos. Planificación, Desarrollo y Control*". Paraninfo, 1994.
- **Quang, P., Gonin J.** "*Dirección de proyectos informáticos*". Eyrolles, 1994.
- **Romero, C.** "*Técnicas de Programación y Control de Proyectos*". Pirámide, 1997.
- **Sommerville, I.** "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002.

5.4.1.2 Desarrollo comentado del programa de teoría

El Plan de Estudios de Ingeniero en Informática (segundo ciclo) de la Universidad de Salamanca, publicado en el BOE número 156 de 1 de julio de 1999 [BOE, 1999] contiene la siguiente descripción de los contenidos de la asignatura *Administración de Proyectos Informáticos*:

> *Gestión de configuraciones. Planificación y gestión de proyectos informáticos.*

Atendiendo a la descripción anterior, el programa tiene que cubrir el estudio de las técnicas y actividades relacionadas con los aspectos de planificación y gestión de proyectos que deben llevarse a cabo durante el desarrollo de un proyecto de software y durante el mantenimiento posterior del sistema.

La gestión del proyecto comprende actividades muy diversas como pueden ser la programación y seguimiento del proceso, gestión de recursos y gestión de la calidad. Para llevarlas a cabo es necesario hacer uso de técnicas y herramientas de medición, de estimación y análisis de riesgos, entre otras. Todos estos aspectos se recogen en los que conforman el programa docente de la asignatura.

En este epígrafe se va a desarrollar con mayor grado de detalle el programa de la parte teórica de la asignatura *Administración de Proyectos Informáticos.* Este programa se ha dividido en seis temas. Dicho programa se desarrolla a lo largo de dos cuatrimestres, en el primero sólo se imparte teoría, durante dos horas a la semana, mientras que en el segundo se imparten dos horas de teoría y dos de prácticas.

Es obvio que, debido a los créditos asignados a la asignatura, no es posible extenderse en todas los temas por igual. En algunos casos la explicación se verá reducida a una breve referencia o comentario, mientras que en otros, que se consideran más representativos y de mayor actualidad, se profundiza más en las explicaciones.

Igual que en la descripción del temario de las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*, cada tema se divide en subtemas, y éstos en epígrafes. Todos los temas van a ser descritos siguiendo el patrón de documentación compuesto por las entradas: *título, descriptores, objetivos, contenido, resumen* y *bibliografía*.

Tema 1. Visión general de la administración de proyectos (6 horas)

1.1 Bases del desarrollo de software (2 Horas)

1.1.1 Bases de gestión

1.1.2 Bases técnicas

1.1.3 Bases de control de calidad

1.2 Madurez del proceso de software (1 Hora)

1.2.1 El proceso software

1.2.2 Niveles de madurez

1.3 Organización de un proyecto (1,5 Horas)

1.3.1 Decisión estratégica

1.3.2 El equipo de trabajo

1.4 Herramientas de ayuda (1 Hora)

1.4.1 Tipos de herramientas

1.4.2 Criterios de selección

1.5 Teoría W (0,5 Horas)

Tema 2. Medición del software (11 horas)

2.1 Conceptos básicos (0,5 Horas)

2.2 Medidas y modelos (0,5 Horas)

2.3 Alcance de las métricas del software (0,5 Horas)

2.4 Clasificación de las métricas de software (1 Hora)

2.4.1 Procesos

2.4.2 Productos

2.4.3 Recursos

2.5 Recogida de datos métricos (0,5 Horas)

2.6 Medición de atributos internos del producto (4 Horas)

2.6.1 Longitud

2.6.2 Funcionalidad

2.6.3 Complejidad

2.6.4 Medidas estructurales

2.7 Medición de atributos externos del producto (2 Horas)

2.8 Medición de recursos (1 Hora)

2.9 Métricas para sistemas orientados a objetos (1 Hora)

Tema 3. Métodos de estimación y gestión del riesgo (10 horas)

3.1 Introducción (0,5 Horas)

3.2 Precisión y exactitud de las estimaciones (0,5 Horas)

3.3 Principios de estimación (0,5 Horas)

3.4 Estimación de costes (1 Hora)

3.4.1 Técnicas de estimación

3.4.2 Curva de aprendizaje

3.5 Modelos de coste y esfuerzo (5 Horas)

3.5.1 Modelos de regresión

3.5.2 Modelo de Bailey-Basili

3.5.3 Modelo COCOMO

3.5.4 Modelo SLIM de Putnam

3.6 Las estimaciones en el CMM (0,5 Horas)

3.7 Gestión de riesgos (2 Horas)

3.7.1 Definición y clasificación

3.7.2 Actividades

3.7.3 Estimación de riesgos

3.7.4 Control de riesgos

3.7.5 Los riesgos en el modelo COCOMO

Tema 4. Planificación temporal de proyectos (12 horas)

4.1 Introducción (1 Hora)

4.2 Notaciones gráficas (0,5 Horas)

4.3 Grafos: conceptos básicos (1 Hora)

4.4 Métodos de planificación temporal (0,5 Horas)

4.5 Método PERT (6 Horas)

4.5.1 Principios básicos

4.5.2 Construcción del grafo PERT

4.5.3 Asignación de tiempos a las actividades

4.5.4 Cálculo de los tiempos EET y LET

4.5.5 Conceptos de holgura y camino crítico

4.5.6 Calendario de ejecución del proyecto

4.5.7 El método PERT en un contexto aleatorio

4.6 Método CPM (1 Hora)

4.6.1 Relación entre duración y coste de una actividad
4.6.2 Diferencias entre PERT y CPM
4.7 Método ROY (2 Horas)
4.7.1 Principios básicos
4.7.2 Construcción del grafo ROY
4.7.3 Cálculo de los tiempos mínimo y máximo
4.7.4 Holguras y calendario de ejecución
Tema 5. Gestión y aseguramiento de la calidad (15 horas)
5.1 Concepto de calidad (15 Minutos)
5.2 Definición de calidad del software (15 Minutos)
5.3 Aspectos de la gestión de calidad (0,5 Horas)
5.4 Ámbitos de la gestión de calidad (2 Horas)
5.4.1 Calidad en el ámbito de la organización
5.4.2 Calidad en el ámbito del proyecto
5.5 Marco normativo (3 Horas)
5.5.1 Estándares ISO 9000
5.5.2 Estándares IEEE
5.6 Actividades de aseguramiento de la calidad (2 Horas)
5.7 Evaluación de la calidad (5 Horas)
5.7.1 Modelos de calidad del software
5.7.2 Fiabilidad del software
5.7.3 Revisiones
5.8 Métricas de calidad (2 Horas)
Tema 6. Gestión de configuraciones (5 horas)
6.1 Introducción (0,5 Horas)
6.2 Configuración de referencia (0,5 Horas)
6.3 Funciones de la gestión de la configuración (0,5 Horas)
6.4 Identificación de la configuración (1 Hora)
6.5 Control de la configuración (1,5 Horas)
6.6 Auditoría de la configuración (0,5 Horas)
6.7 Contabilidad del estado de la configuración (0,5 Horas)

Tabla 5.42. Estructura detallada del programa de teoría de Administración de Proyectos Informáticos

Tema 1: ***Visión general de la administración de proyectos***

Descriptores

Gestión de proyectos; producto; coste; planificación; seguimiento; medida; estimación; calendario; auditoría; gestión de configuraciones; gestión de requisitos; niveles de madurez; CMM; plan de sistemas; plan estratégico; equipo de trabajo; tipo de equipo de trabajo; modelo de equipo; CASE; teoría W.

Objetivos

Este primer tema está orientado a satisfacer los objetivos **T10** y **T11** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.
- Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Ofrecer una visión global de la gestión de proyectos.
- Justificar la importancia de una correcta y efectiva gestión de los proyectos informáticos.

Contenidos

1.1 Bases del desarrollo de software
1.2 Madurez del proceso de software
1.3 Organización de un proyecto
1.4 Herramientas de ayuda
1.5 Teoría W

Tabla 5.43. Contenido del primer tema del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE8 Software management** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Management** [MacDonell y Gray, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

En este tema se pretende ofrecer una visión global de las actividades de la gestión de proyectos, para lo cual se comienza estableciendo la clasificación de las bases de desarrollo de cualquier sistema de software en tres categorías, técnicas, de gestión y control de calidad. Gran parte de las actividades y métodos encuadrados en alguno de los grupos anteriores son las que se contemplan en esta asignatura y serán desarrolladas en temas posteriores.

El nivel de aplicación de las prácticas y técnicas mencionadas va a servir para establecer el grado de madurez del proceso de una organización. En el segundo apartado del tema se describen las actividades del proceso y se describen los niveles de madurez propuestos en el modelo CMM (*Capability Maturity Model*) del SEI (*Software Engineering Institute*) [Paulk et al., 1993a; Paulk et al., 1993b].

En este tema introductorio es importante apuntar la forma cómo se inician los proyectos de software y en qué forma se organizan. La estructura y funciones del equipo del proyecto, así como los distintos modelos de equipo, las reglas básicas de formación y la estructuración de caminos de comunicación en grandes equipos se tratan en el tercer apartado del tema.

Actualmente, existe en el mercado gran número de herramientas que permiten automatizar total o parcialmente la mayoría de las actividades descritas. La clasificación de tales herramientas y el establecimiento de una serie de criterios de selección que se proporciona en este tema será de utilidad al alumno para saber qué tipo de herramienta utilizar en cada momento.

El tema finaliza con la descripción de la teoría W propuesta por Barry Boehm y Rony Ross [Boehm y Ross, 1989] en la que se sugieren un conjunto de condiciones y reglas de trabajo para la reconciliación de intereses opuestos. Su objetivo es hacer ganadores a todos los participantes en el proyecto incluyendo desarrolladores y clientes.

Bibliografía básica

Humphrey, W. S. "*Introducción al Proceso Software Personal*". Addison-Wesley, 2001. (Capítulos 1, 5, 10, 11, 18 y 19).

Lewin, M. D. "*Better Software Project Management: A Primer for Success*". John Wiley & Sons, 2002. (Capítulos 1 y 2).

McConnell, S. "*Desarrollo y Gestión de Proyectos Informáticos*", McGraw-Hill 1997. (Capítulos 4, 11, 12, 13, 15).

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 2, 3, y 31).

Puig, J. "*Proyectos Informáticos. Planificación, Desarrollo y Control*". Paraninfo, 1994.

Quang, P., Gonin J. "*Dirección de proyectos informáticos*". Eyrolles, 1994.

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 3, 4, 22 y 25).

Bibliografía complementaria

Boehm, B., Ross, R. "*Theory-W Software Project Management: Principles and Examples*". IEEE Transaction on Software Engineering, 1989.

Brooks, F. "*The Mythical Man-Month*". Anniversary Edition, Addison-Wesley, 1995.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

Chang, C. K. (Editor). "*Annals of Software Engineering Special Volume on Software Management*". Annals of Software Engineering, 11(1). November 2001.

Humphrey, W. S. "*Introduction to the Team Software Process*". Addison-Wesley, 2000.

IEEE-CS. "*IEEE Software Special Volume on Managing Megaprojects*". IEEE Software, 13(4). July 1996.

Juan, A., Pérez, P. "*La Auditoría en el Desarrollo de Proyectos Informáticos*". Díaz de Santos, 1988.

Larson, C., LaFasto, F. "*Teamwork: What Must Go Right, What Can Go Wrong*". Sage, 1989.

MacDonell, S. G., Gray, A. R. "*Software Engineering Management*". Chapter 8 in [Abran et al., 2001].

McCarthy, J. "*Dynamics of Software Development*". Microsoft Press, 1995.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

Project Management Institute. "*A Guide to the Project Management Body of Knowledge*". Project Management Institute. 2000 Edition. 2000.

Tema 2: ***Medición del software***

Descriptores

Medida; métrica; indicador; medida directa; medida indirecta; estimación de coste; estimación de esfuerzo; modelos y medidas de productividad; recolección de datos; modelos y medidas de calidad; métricas del proceso; métricas del proyecto; métricas de calidad; métricas orientadas al tamaño; puntos de función; atributos internos; atributos externos; COCOMO; medidas estructurales; métricas de recursos; métricas OO.

Objetivos

Este tema está orientado a satisfacer el objetivo **T11** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Recalcar la importancia que tiene la medición, siendo elemental para cualquier disciplina de Ingeniería del Software.
- Diferenciar diferentes tipos de métricas internas al producto.
- Presentar diversas métricas externas al producto.
- Introducir las métricas de recursos.
- Introducir algunas métricas orientadas a objetos.

Contenidos

2.1 Conceptos básicos
2.2 Medidas y modelos
2.3 Alcance de las métricas del software
2.4 Clasificación de las métricas de software
2.5 Recogida de datos métricos
2.6 Medición de atributos internos del producto
2.7 Medición de atributos externos del producto
2.8 Medición de recursos
2.9 Métricas para sistemas orientados a objetos

Tabla 5.44. Contenidos del tema dos del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE8 Software management** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Management** [MacDonell y Gray, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

El propósito del segundo tema es el estudio de las diferentes técnicas y métricas que permiten evaluar la calidad de un producto y la productividad del proceso. Dichas técnicas, constituyen la base de los métodos de estimación que se tratarán en el tema siguiente.

Se comienza introduciendo los conceptos de medida, métrica e indicador, así como la diferencia entre medidas directas e indirectas. El análisis de la doble vertiente de los métodos de medición, evaluación y predicción, sirve para introducir diferentes modelos de calidad y productividad que dan una idea de la utilidad y el alcance de las métricas del software.

El proceso de recogida, refinamiento y análisis de datos se describe en el apartado cinco. En él se recoge también el enfoque GQM (*Goal-Question-Metric*) [Basili y Weiss, 1984; Basili y Rombach, 1988] para seleccionar e implementar métricas de una manera efectiva.

La clasificación de las métricas se realiza en primer lugar en función de que los atributos a medir sean del proceso, del producto o de recursos. Para cada uno de los grupos se establece una nueva clasificación dependiendo de que los atributos sean internos o externos.

La clasificación anterior se utiliza en los apartados seis, siete y ocho del tema para profundizar en las métricas propuestas por diferentes autores para evaluar y cuantificar aspectos internos del software como tamaño, funcionalidad, complejidad, modularidad... o las características de calidad establecidas en el modelo estándar ISO 9126 [ISO/IEC, 1991], sin olvidar las métricas específicas de recursos. Se dedica también un apartado a las métricas para sistemas orientados a objetos que, aunque están menos desarrolladas que las tradicionales, cada vez van adquiriendo mayor importancia.

Bibliografía básica

Fenton, N. E., Pfleeger, S. L. "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.

McConnell, S. "*Desarrollo y Gestión de Proyectos Informáticos*", McGraw-Hill 1997. (Capítulo 26).

McGarry, J., Card, D., Jones, C., Layman, B., Clark, E., Dean, J., Hall, F. "*Practical Software Measurement. Objective Information for Decision Makers*". Addison-Wesley, 2002. (Capítulos 1, 2, 3, 4, 5, 6, 7 y 8).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 4, 19 y 24).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 24).

Bibliografía complementaria

Basili, V. R., Rombach, H. D. "*The TAME Project: Towards Improvement-Oriented Software Environments*". IEEE Transaction on Software Engineering, 14(6):758-773, 1988.

Basili, V. R., Weiss, D. "*A Methodology for Collecting Valid Software Engineering Data*". IEEE Transaction on Software Engineering, 10(6):728-738, 1984.

Chidamber, S. R., Kemerer, C. F. "*A Metrics Suite for Object-Oriented Design*". IEEE Transactions of Software Engineering, 20(6):476-493, 1994.

Churcher, N. I., Shepperd, M. J. "*Towards Conceptual Framework for Object-Oriented Metrics*". ACM Software Engineering Notes, 20(2):67-76, 1995.

Dolado, J. J., Fernández, L. (Coordinadores). "*Medición para la Gestión en la Ingeniería del Software*". Ra-ma, 2000.

Heller, R. "*An Introduction to Function Point Analysis*". Crosstalk. November-December 1995.

Lorenz, M., Kidd, J. "*Object-Oriented Software Metrics*". Prentice Hall, 1994.

MacDonell, S. G., Gray, A. R. "*Software Engineering Management*". Chapter 8 in [Abran et al., 2001].

Mills, H. D., Dyson, P. B. (Guest Editors). "*IEEE Software Special Volume on Metrics*". IEEE Software, 7(2). March-April 1990.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

Pfleeger, S. L. (Guest Editor). "*IEEE Software Special Volume on Measurement*". IEEE Software, 14(2). March-April 1997.

Ragland, B. "*Measure, Metric or Indicator: What´s the Difference?*". Crosstalk, 8(3):29-30, 1995.

Tema 3: ***Métodos de estimación y gestión del riesgo***

Descriptores

Modelo de estimación; estimación de costes; técnicas de estimación; modelo de coste y esfuerzo; COCOMO; SLIM; CMM; riesgo; gestión de riesgos.

Objetivos

Este tema está orientado a satisfacer el objetivo **T10** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Profundizar en el aspecto predictivo de las métricas.
- Establecer la importancia de la recolección de métricas de proyectos, para la estimación adecuada en nuevos proyectos.

Contenidos

3.1 Introducción
3.2 Precisión y exactitud de las estimaciones
3.3 Principios de estimación
3.4 Estimación de costes
3.5 Modelos de coste y esfuerzo
3.6 Las estimaciones en el CMM
3.7 Gestión de riesgos

Tabla 5.45. Contenidos del tema tres del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE8 Software management** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Management** [MacDonell y Gray, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

En este tema se profundiza, en el aspecto predictivo de las métricas del software. Las técnicas estudiadas en el tema anterior se utilizan en los métodos de estimación que se describen en este tema. La aplicación efectiva de los métodos requiere el conocimiento previo de una serie de

conceptos y principios como son la exactitud y precisión apropiadas o la necesidad de volver a realizar estimaciones a medida que avanza el proyecto. Estas nociones se exponen en los tres primeros apartados.

Los métodos que se reseñan a continuación son específicos de estimación de esfuerzo, coste y tiempo. En primer lugar se introduce la clasificación de los mismos, desarrollando detalladamente después las técnicas conocidas como modelos (paramétricos, empíricos o algorítmicos), ya que son las más utilizadas y las que producen estimaciones más fiables.

Entre los métodos de estimación de esfuerzo estudiados destaca en especial COCOMO (*COnstructive COst MOdel*), del que existe una versión clásica [Boehm, 1981] y la versión actual, denominada COCOMO II [Boehm et al, 1995; Boehm et al., 2000].

El hecho de que la maduración del proceso de Software dentro de una organización mejore la predictibilidad y controlabilidad justifica nuevamente la aparición del modelo CMM (*Capability Maturity Model*) [Paulk et al., 1993a; Paulk et al., 1993b] y la relación de las prácticas de estimación con sus diferentes niveles.

Tampoco podía faltar en este tema la actividad de gestión de riesgos, indispensable para identificar, cuantificar y controlar las incertidumbres sobre ciertas características del proyecto (derivadas de la incertidumbre inherente a las estimaciones) y las pérdidas asociadas a ellas.

Bibliografía básica

Burnett, K. "*The Project Management Paradigm*". Springer-Verlag, 1998.

Fenton, N. E., Pfleeger, S. L. "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.

Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International Editions, 1991. (Capítulo 8).

McConnell, S. "*Desarrollo y Gestión de Proyectos Informáticos*". Mc Graw Hill, 1997. (Capítulo 8).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 5).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulos 5 y 6).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 4 y 23).

USC. "*COCOMO II Model Manual*". University of Southern California. USC COCOMO II.1998.0. http://sunset.usc.edu/research/COCOMOII/index.html. 1998.

USC. "*USC COCOMO II User's Manual*". University of Southern California. USC COCOMO II.1999.0. http://sunset.usc.edu/research/COCOMOII/index.html. 1999.

Bibliografía complementaria

Arthur, J. D., Henry, S. M. (Editors). "*Annals of Software Engineering Special Volume on Software Process and Product Measurement*". Annals of Software Engineering, Vol. 1, 1995.

Bailey, J. W., Basili, V. R. "*A Meta-Model for Software Development Resource Expenditure*". In proceedings of the 5th International Conference on Software Engineering, IEEE Computer Society Press. Pages 189-197, 1981.

Boehm, B. W. "*Software Engineering Economics*". Prentice Hall, Englewood Cliffs, NJ, 1981.

Boehm, B. W. "*Software Risk Management: Principles and Practices*". IEEE Software, 8(1):32-41. January 1991.

Boehm, B. W., Abts, C., Chulani, S. "*Software Development Cost Estimation Approaches – A Survey*". Annals of Software Engineering, 10:177-205. 2000.

Boehm, B. W., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R. "*Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*". Annals of Software Engineering, Vol. 1:57-94, 1995.

Boehm, B. W., DeMarco, T. (Guest Editors). "*IEEE Software Special Volume on Risk Management*". IEEE Software, 14(3). May-June 1997.

Boehm, B. W., DeMarco, T. "*Software Risk Management*". IEEE Software, 14(3):17-19. May-June 1997.

Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., Abts, C. "*Software Cost Estimation with Cocomo II*". Prentice Hall, 2000.

Charette, R. "*Software Engineering Risk Analysis and Management*". McGraw-Hill, 1989.

Dolado, J. J., Fernández, L. (Coordinadores). "*Medición para la Gestión en la Ingeniería del Software*". Ra-ma, 2000.

Humphrey, W. S. "*Introducción al Proceso Software Personal*". Addison-Wesley, 2001.

Känsälä, K. "*Integrating Risk Assessment with Cost Estimation*". IEEE Software, 14(3):61-67. May-June 1997.

MacDonell, S. G., Gray, A. R. "*Software Engineering Management*". Chapter 8 in [Abran et al., 2001].

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

Project Management Institute. "*A Guide to the Project Management Body of Knowledge*". Project Management Institute. 2000 Edition. 2000.

Putnam, L. H. "*A General Empirical Solution to the Macrosoftware Sizing and Estimating Problem*". IEEE Transaction on Software Engineering, 4(4):345-361, 1978.

Rosenberg, L. H., Gallo, A., Hammer, T., Parolek, F. "*Continuing Risk Management at NASA*". CrossTalk, 13(2):7-11, 2000.

Tema 4: ***Planificación temporal de proyectos***

Descriptores

Planificación temporal; calendario; red de tareas; diagrama de Gantt; método de planificación temporal; PERT; CPM; ROY; camino crítico; retraso; holgura; estructura de descomposición de trabajo; plan de proyecto.

Objetivos

Este tema está orientado a satisfacer el objetivo **T10** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar los objetivos, principios y métodos de planificación temporal de las diferentes actividades del proyecto.
- Realizar un calendario del proyecto.

Contenidos

4.1 Introducción
4.2 Notaciones gráficas
4.3 Grafos: conceptos básicos
4.4 Métodos de planificación temporal
4.5 Método PERT
4.6 Método CPM
4.7 Método ROY

Tabla 5.46. Contenidos del tema cuatro del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos del área **SE8 Software management** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Engineering Management** [MacDonell y Gray, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

A **Fred Brooks**, el conocido autor del libro *The Mythical Man-Month* [Brooks, 1995], se le preguntó una vez cómo se retrasan los proyectos. Su respuesta fue simple y rotunda: "Diariamente".

La finalidad de este tema es dar a conocer los objetivos, principios y métodos de planificación temporal de las diferentes actividades del proyecto. Dado que la mayoría de las técnicas actuales se apoyan en representaciones gráficas, se explican en primer lugar dos de las notaciones más utilizadas: *los diagramas de Gantt* y *las redes de tareas o grafos del proyecto*. Respecto a estos últimos, en el tema se incide especialmente en aquellos aspectos de la teoría de grafos que son esenciales para la comprensión de los métodos.

Después de realizar una revisión somera de las principales técnicas de planificación se aborda el estudio en profundidad de los métodos que más difusión han tenido hasta el momento: PERT (*Program Evaluation and Review Technique*) [Malcom et al., 1959], CPM (*Critical Path Method*) [Kelley, 1961; Moder et al., 1983] y ROY [Roy, 1960]. Se contemplan, además, en este estudio las peculiaridades del método PERT en situaciones de riesgo, en las cuales se supone que las duraciones de las actividades son variables aleatorias de las que sólo se conocen sus distribuciones de probabilidad. También se incluye la variante del método CPM denominada programación de proyectos a coste mínimo en la que se establece la relación entre el coste y la duración de una actividad.

Bibliografía básica

Cos, M. "*Teoría General del Proyecto*". Síntesis, 1997.

Humphrey, W. S. "*Introducción al Proceso Software Personal*". Addison-Wesley, 2001. (Capítulos 2, 3, 4, 5, 7 y 10).

Lewin, M. D. "*Better Software Project Management: A Primer for Success*". John Wiley & Sons, 2002. (Capítulo 3).

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulos 4 y 7).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 7).

Romero, C. "*Técnicas de Programación y Control de Proyectos*". Pirámide, 1997.

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 4).

Bibliografía complementaria

Burnett, K. "*The Project Management Paradigm*". Springer-Verlag, 1998.

Davis, W. S. "*Herramientas CASE. Metodología Estructurada para el Desarrollo de Sistemas*". Paraninfo, 1983.

Humphrey, W. S. "*Introduction to the Team Software Process*". Addison-Wesley, 2000.

Kezner, H. "*Project Management: A Systems Approach to Planning, Scheduling, and Controlling*". Wiley, 1998.

Lewis, J. P. "*Mastering Project Management: Applying Advanced Concepts of Systems Thinking, Control and Evaluation*". McGraw-Hill, 1988.

MacDonell, S. G., Gray, A. R. "*Software Engineering Management*". Chapter 8 in [Abran et al., 2001].

McConnell, S. C. "*Software Project Survival Guide*". Microsoft Press, 1998.

O'Connell, F. "*How to Run Successful Projects III: The Silver Bullet*". 3rd Edition. Addison-Wesley, 2001.

Project Management Institute. "*A Guide to the Project Management Body of Knowledge*". Project Management Institute. 2000 Edition. 2000.

Yu, L. "*Aplicaciones prácticas de PERT y CPM*". Ediciones Deusto, 1974.

Zaderenco, S. G. "*Sistemas de Programación por Camino Crítico*". Editorial Librería Mitre, 1968.

Tema 5: ***Gestión y aseguramiento de la calidad***

Descriptores

Calidad; SQA; ISO 9001; modelos de calidad del software; FCM; GQM; CMM; SPICE; ISO 9126; modelo de Boehm; modelo de Gilb; métricas de calidad; fiabilidad del software; revisión del software; coste de la calidad

Objetivos

Este tema está orientado a satisfacer los objetivos **T10** y **T11** identificados en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.
- Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Presentar los aspectos esenciales de las actividades de gestión destinadas a mejorar la calidad de los procesos y de los productos software.
- Comparar y contrastar los diferentes métodos y técnicas utilizadas para asegurar la calidad de un producto software.

Contenidos

5.1 Concepto de calidad
5.2 Definición de calidad del software
5.3 Aspectos de la gestión de calidad
5.4 Ámbitos de la gestión de calidad
5.5 Marco normativo
5.6 Actividades de aseguramiento de la calidad
5.7 Evaluación de la calidad
5.8 Métricas de calidad

Tabla 5.47. Contenidos del tema cinco del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos de las áreas **SE8 Software management** y **SE11 Software reliability** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Quality** [Wallace y Reeker, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

La Ingeniería del Software está dirigida hacia el objetivo del producir software de calidad. Pero la gran pregunta es ¿qué es la calidad?. Philip Crosby expone esta situación [Crosby, 1979]:

> "*El problema de la gestión de la calidad no es lo que la gente no sabe sobre ella. El problema es lo que creen que saben...*
>
> *A este respecto, la calidad tiene mucho en común con el sexo. Todo el mundo lo quiere (bajo ciertas condiciones, por supuesto). Todo el mundo cree que lo conoce (incluso aunque no quiera explicarlo). Todo el mundo piensa que su ejecución sólo es cuestión de seguir las inclinaciones naturales (después de todo nos las arreglamos de alguna forma). Y, por supuesto, la mayoría de la gente piensa que los problemas en estas áreas están producidos por otra gente (como si sólo ellos se tomaran el tiempo para hacer las cosas bien)*".

Algunos desarrolladores de software continúan creyendo que la calidad del software es algo en lo que hay que preocuparse una vez que se ha generado el código, lo que es totalmente falso. La garantía de calidad del software, SQA (*Software Quality Assurance*), es una actividad de protección que se aplica a lo largo de todo el proceso del software.

La SQA engloba: 1) un enfoque de gestión de calidad; 2) tecnología de Ingeniería del Software efectiva (métodos y herramientas); 3) revisiones técnicas formales que se aplican durante el proceso del software; 4) una estrategia de prueba multiescalada; 5) el control de la documentación del software y de los cambios realizados; 6) un procedimiento que asegure un ajuste a los estándares de desarrollo del software (cuando sea posible); y 7) mecanismos de medición y de generación de informes [Pressman, 2000].

Así, el presente tema tiene por objeto presentar los aspectos esenciales de las actividades de gestión encaminadas a mejorar la calidad de los procesos y productos de software.

Los primeros apartados del tema están destinados a la definición y aclaración de conceptos relacionados con la calidad, así como a la introducción de las principales actividades y técnicas asociadas con la calidad del software como son: el aseguramiento, el control de calidad y la verificación y validación, algunas de las cuales se desarrollarán a lo largo del tema. Previamente se realiza una clasificación en función su ámbito de aplicación, que se realiza a dos niveles, uno más general que se enmarca en la política de la empresa u organización y que dicta las

directrices comunes a todos los proyectos, y otro más específico en el que se adaptan dichas disposiciones a cada proyecto particular.

No podía faltar en este tema la normativa estándar de las organizaciones ISO e IEEE. Ambas normativas se presentan de forma esquemática, describiéndose sus características fundamentales. En el caso de las normas ISO 9000 [ISO, 1994a] se hace una clasificación en función de su aplicación para la gestión interna o para el aseguramiento externo de la calidad y se explica con mayor detenimiento la guía ISO 9000-3 [ISO, 1994a] que adapta la norma ISO 9001 [ISO, 1994b] al desarrollo suministro y mantenimiento de software. Asimismo, se presta especial atención al estándar ISO 9126 [ISO/IEC, 1991] en el que se describen las características y atributos de la calidad del software. Los estándares IEEE [IEEE, 1999] ya han sido comentados en otros temas de las asignaturas de Ingeniería del Software debido a que están orientados a normalizar las actividades del ciclo de vida. Sin embargo aquí se enfatiza la incidencia que tienen en la calidad del proceso y del producto.

El tema de la evaluación de la calidad se aborda desde la perspectiva de diferentes modelos propuestos para descomponer el concepto global de calidad en propiedades más fáciles de medir y evaluar. Se estudian los modelos de B. W. Boehm [Boehm et al., 1980], FCM (*Factor-Criteria-Metric*) [McCall et al., 1977], GQM (*Goal-Question-Metric*) [Basili y Weiss, 1984; Basili y Rombach, 1988], de T. Gilb [Gilb, 1988], CMM (*Capability Maturity Model*) [Paulk, et al., 1993a; Paulk, et al., 1993b] y SPICE (*Software Process Improvement and Capability dEtermination*) [Rout, 1995; El Emam et al., 1998]. En el mismo apartado se discute la técnica de las revisiones y sus diferentes tipos: revisiones de gestión, revisiones técnicas, auditorías, inspecciones de software y *walkthrough* [Wallace y Reeker, 2001].

El tema concluye con el estudio de las métricas más apropiadas para evaluar la calidad. La mayoría de estas métricas están centradas en la evaluación de atributos del producto, por lo que ya se han contemplado en el segundo tema de la asignatura, aunque aquí se incidirá especialmente en la medida de fiabilidad, ya que ésta es una de las características de la calidad más importantes y más ampliamente estudiadas [Fenton y Pfleeger, 1997].

Bibliografía básica

Fenton, N. E., Pfleeger, S. L. "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997. (Capítulo 9).

Humphrey, W. S. "*Introducción al Proceso Software Personal*". Addison-Wesley, 2001. (Capítulos 18, 19 y 20).

Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 13).

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice-Hall, 1998. (Capítulo 11).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 8).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulos 17, 24 y 25).

Bibliografía complementaria

AENOR. "*Normas para la Gestión y el Aseguramiento de la Calidad*". Madrid, AENOR, 1992.

Basili, V. R., Rombach, H. D. "*The TAME Project: Towards Improvement-Oriented Software Environments*". IEEE Transaction on Software Engineering, 14(6):758-773, 1988.

Basili, V. R., Weiss, D. "*A Methodology for Collecting Valid Software Engineering Data*". IEEE Transaction on Software Engineering, 10(6):728-738, 1984.

Burnett, K. "*The Project Management Paradigm*". Springer-Verlag, 1998.

Dolado, J. J., Fernández, L. (Coordinadores). "*Medición para la Gestión en la Ingeniería del Software*". Ra-ma, 2000.

Fernández, L. "*Una Revisión Breve de la Medición del Software*". Novática, 137:20-24, 1999.

Gilb, T. "*Principles of Software Engineering Management*". Addison-Wesley, 1988.

Hoffnagle, G. F. (Editor). "*Special Issue on Software Quality*". IBM Systems Journal, 33(1). 1994.

Humphrey, W. S. "*Introduction to the Team Software Process*". Addison-Wesley, 2000.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. 4–Volume Set*". IEEE Computer Society Press, 1999.

IEEE Std. 610.12. "*IEEE Standard Glossary of Software Engineering Terminology*". 1990.

IEEE Std. 12207. "*IEEE Standard for developing Software Life Cycle Processes*". 1998.

IEEE Std. 730. "*Software Quality Assurance Plans*". 1998.

IEEE Std. 829. "*Software Test Documentation*". 1998.

IEEE Std. 1008. "*Software Unit Test*". 1998.

IEEE Std. 1012. "*Software Verification and Validation*". 1998.

IEEE Std. 1028. "*Software Reviews*". 1997.

IEEE Std. 1044. "*Standard Classification for Software Anomalies*". 1993.

IEEE Std. 1061. "*Standard for a Software Quality Metrics Methodology*". 1992.

IEEE Std. 1228. "*Software Safety Plans*". 1994.

In, H., Boehm, B. W. "*Using WinWin Quality Requirements Management Tools: A Case Study*". Annals of Software Engineering, 11:141-174. 2001.

ISO/IEC 12207. "*Information Technology-Software Life Cycle Processes*". 1995.

ISO/IEC TR 15504. "*Software Process Assessment*". 1998.

ISO/IEC 8402. "*Quality - Vocabulary*". 1986.

ISO 9000. "*Quality Management and Quality Assurance Standards*". 1994.

ISO 9001. "*Quality Systems*". 1994.

ISO/IEC 9126. "*Software Product Evaluation – Quality Characteristics and Guidelines for their Use*". 1991.

Ministerio para las Administraciones Públicas. "*PlanGeneral de Garantía de Calidad Aplicable al Desarrollo de Equipos Lógicos*". 1991.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

Project Management Institute. "*A Guide to the Project Management Body of Knowledge*". Project Management Institute. 2000 Edition. 2000.

Rout, T. P. "*SPICE: A Framework for Software Process Assessment*". Software Process Improvement and Practice, 1(1):57-66, 1995.

Schneidewind, N. F. "*Body of Knowledge for Software Quality Measurement*". IEEE Computer, 35(2):77-83. February 2002.

SPICE. "*SPICE Document Suite, Software Process Improvement and Capability dEtermination*". http://www.sqi.gu.edu.au/spice/. 1999.

Wallace, D., Reeker, L. "*Software Quality*". Chapter 11 in [Abran et al., 2001].

Tema 6: ***Gestión de configuraciones***

Descriptores

Configuración del software; gestión de la configuración del software; cambio; control del cambio; configuración de referencia (*baseline*); elementos de configuración del software; control de la configuración; auditoría de la configuración.

Objetivos

Este primer tema está orientado a satisfacer el objetivo **T12** identificado en la *Unidad Docente de Ingeniería del Software y Orientación a Objetos*, a saber:

- Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

De manera más concreta se pueden enunciar los siguientes objetivos:

- Introducir los conceptos fundamentales de la gestión de la configuración del software.
- Describir las actividades necesarias para gestionar y controlar los cambios que se producen en un producto software a lo largo de su evolución.
- Distinguir entre mantenimiento del software y la gestión de la configuración del software.

Contenidos

6.1 Introducción
6.2 Configuración de referencia
6.3 Funciones de la gestión de la configuración
6.4 Identificación de la configuración
6.5 Control de la configuración
6.6 Auditoría de la configuración
6.7 Contabilidad del estado de la configuración

Tabla 5.48. Contenidos del tema seis del programa teórico de Administración de Proyectos Informáticos

Los epígrafes de este tema se corresponden con algunos tópicos de las áreas **SE3 Software tools and environments** y **SE8 Software management** del *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

Los contenidos de este tema se corresponden parcialmente con el área de conocimiento **Software Configuration Management** [Scott y Nisse, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

Resumen

Los cambios son inevitables cuando se construye software. Los cambios aumentan el grado de confusión entre los ingenieros del software que están trabajando en el proyecto. La confusión surge cuando no se han analizado los cambios antes de realizarlos, no se han registrado antes de implementarlos, no se les han comunicado a aquellas personas que necesitan saberlo o no se han controlado de forma que mejoren la calidad y reduzcan los errores. W. A. Babich se refiere a este asunto cuando dice [Babich, 1986]: "*El arte de coordinar el desarrollo de software para minimizar la confusión, se denomina gestión de configuración. La gestión de configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores*".

La gestión de configuración del software es una actividad de autoprotección que se aplica durante el proceso del software. Como el cambio se puede producir en cualquier momento, las actividades de gestión de configuración del software sirve para: 1) identificar el cambio; 2) controlar el cambio; 3) garantizar que el cambio se implementa adecuadamente; y 4) informar del cambio a todos aquéllos que puedan estar interesados.

De esta forma, en el último tema de la asignatura se proporcionan los conceptos clave de la gestión de la configuración del software y se describen todas las actividades necesarias para gestionar y controlar de manera adecuada los cambios que se producen en un producto software a lo largo de su evolución.

Se comienza introduciendo algunos conceptos básicos y justificando la necesidad de realizar las actividades de la gestión de configuración de software tomando como base los efectos negativos que produce la falta de visibilidad y de trazabilidad en el proceso de desarrollo y en los productos que se van produciendo.

En el segundo apartado se estudia el proceso de la gestión de configuración de software, considerado como un proceso de soporte del ciclo de vida. Desde la perspectiva de gestión controla la evolución de un producto identificando sus elementos, gestionando y controlando el cambio y, finalmente, verificando, registrando y generando informes sobre la información de configuración. Desde la perspectiva de los desarrolladores facilita el desarrollo y las actividades de implementación de cambios [Scott y Nisse, 2001]. La efectividad de la gestión de configuración de software requiere una cuidadosa planificación, y ello requiere a su vez el

conocimiento de la estructura y relaciones de la organización. Todos estos aspectos se tratan en este apartado del tema, además de las guías proporcionadas por organismos internacionales, las restricciones originadas por las políticas de la empresa y la necesidad de supervisión y aplicación de métricas para controlar la evolución del proceso y de los productos.

En los apartados siguientes se describen todas las actividades implicadas en el proceso de gestión de configuración de software, comenzando por la identificación de los elementos susceptibles de control (configuración, elemento de configuración, versiones, línea base...) y el tipo de biblioteca de software que se va a utilizar. La explicación de la siguiente actividad, control de la configuración, se centra en la determinación de los cambios que hay que realizar, la auditoría para aprobar ciertos cambios y la autorización de desviaciones o incumplimiento de ciertos requisitos establecidos. Se analiza también la información de estado que es necesario identificar, recoger y mantener y los informes que hay que generar. Asimismo se subraya la importancia de la realización de auditorías en la gestión de configuración del software para determinar el grado en que un elemento satisface las características físicas y funcionales requeridas y para establecer la línea base del producto y se explican los diferentes tipos de auditorías que deben llevarse a cabo. El tema concluye con el examen de las actividades de distribución de elementos de configuración fuera de las actividades de desarrollo, así como la preparación y entrega de diferentes versiones si fuera necesario.

Bibliografía básica

Ayer, S. J., Patrinostro, F. S. "*Software Configuration Management: Identification, Accounting, Control and Management*". McGraw Hill, 1992.

Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000. (Capítulo 10).

Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996. (Capítulo 15).

Pressman, R. S. "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002. (Capítulo 9).

Sommerville, I. "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002. (Capítulo 29).

Bibliografía complementaria

AENOR. "*Normas para la Gestión y el Aseguramiento de la Calidad*". Madrid, AENOR, 1992.

Babich, W. A. "*Software Configuration Management Coordination for Team Productivity*". Addison Wesley, 1986.

Berlack, H. R. "*Software Configuration Management*". Wiley, 1992.

Bersoff, E. H. "*Elements of Software Configuration Management*". In Dorfman, M., Thayer, R. H. Editors. *Software Engineering*, IEEE C-S Press, 1997.

Buckley, F. J. "*Implementation Configuration Management: Hardware, Software, and Firmware*". 2nd Edition. IEEE CS Press, 1996.

Conradi, R., Westfechtel, B. "*Version Models for Software Configuration Management*". ACM Computing Surveys, 30(2)232-282. June 1998.

Hoek, A. "*Configuration Management Yellow Pages*". http://www.cs.colorado.edu/users/andre/configuration_management.html.

IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. 4–Volume Set*". IEEE Computer Society Press, 1999.

IEEE Std. 828. "*IEEE Standard for Software Configuration Management Plans*". 1998.

IEEE Std. 1042. "*IEEE Guide to Software Configuration Management*". 1987.

IEEE Std. 12207. "*IEEE Standard for developing Software Life Cycle Processes*". 1998.

ISO/IEC TR 15846. "*Information Technology-Software Life Cycle Processes-Configuration Management*". ISO/IEC, 1998.

ISO/DIS 9004-7 (ISO 10007). "*Quality Management and Quality System Elements, Guidelines for Configuration Management*". ISO, 1993.

Meiser, K. "*Software Configuration Management Terminology*". Crosstalk. January 1995.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

Pfleeger, S. L. "*Software Engineering. Theory and Practice*". Prentice-Hall, 1998.

Royce, W. "*Software Project Management. A Unified Framework*". Addison Wesley, 1998.

Scott, J. A., Nisse, D. "*Software Configuration Management*". Chapter 7 in [Abran et al., 2001].

Whitgift, D. "*Methods and Tools for Software Configuration Management*". John Wiley & Sons, 1991.

5.4.2 Programa de la parte práctica

La parte práctica de la asignatura *Administración de Proyectos Informáticos* está dirigida a satisfacer aquellos objetivos prácticos que, identificados en la Unidad Docente de Ingeniería del Software y Orientación a Objetos, se ajustan a las características y el contexto docente de esta asignatura (**P3** y **P8**); además de promover las habilidades personales en los alumnos (**H1, H2, H3** y **H4**).

Las líneas de acción específicas para la consecución de estos objetivos se detallan en el Plan de Calidad para la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000a] (incluido en el Apéndice A de este Proyecto Docente).

P3	Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.
P8	Recolección de diferentes métricas en el desarrollo de sistemas software reales.

Tabla 5.49. Objetivos del programa de prácticas de la asignatura Administración de Proyectos Informáticos

5.4.2.1 Consideraciones iniciales

El objetivo fundamental de las prácticas de la asignatura *Administración de Proyectos Informáticos* es la aplicación de las técnicas de gestión de proyectos informáticos, fundamentalmente medir, estimar y planificar temporalmente, utilizando los conceptos presentados en la teoría, apoyados éstos en herramientas CASE.

Se considera que las 30 horas, que equivalen a los tres créditos de la parte práctica, deben repartirse en transmisión de contenidos en laboratorio y prácticas no guiadas.

Las prácticas de transmisión de contenidos en laboratorio se reducen al mínimo de horas necesarias para presentar los entornos utilizados en las prácticas, así como los enunciados de las prácticas obligatorias.

En las prácticas no guiadas los alumnos deberán desarrollar las prácticas obligatorias, teniendo el respaldo del profesor para las dudas que puedan surgir en la realización de los mismos.

El número de prácticas obligatorias a realizar será tres, una centrada en la familiarización con las técnicas de medición, otra en el uso de técnicas de estimación y una tercera centrada en la planificación temporal. Para llevar a cano cada práctica se tomará como base un proyecto realizado por los alumnos en la asignatura *Análisis de Sistemas*. Cada práctica se puede realizar

en grupos de dos alumnos. Al finalizar las mismas, los alumnos deberán entregar la documentación correspondiente, que servirá para su evaluación. El documento podrá ser elaborado conjuntamente por ambos alumnos, pero la defensa de las prácticas se realizará de forma individualizada.

En la página web de la asignatura se encuentra la descripción de las prácticas que se van a realizar, así como el material necesario para llevarlas a cabo y enlaces a partir de los cuales se puede encontrar información adicional.

Para el mejor aprovechamiento de las prácticas, éstas se llevan a cabo una vez cada semana durante dos horas consecutivas, durante el segundo cuatrimestre (la parte teórica se desarrolla durante dos horas a la semana durante los dos cuatrimestres).

5.4.2.2 Estructura y distribución temporal

Para lograr los objetivos marcados, el programa de la parte práctica de esta asignatura se ha organizado en tres prácticas, tal y como se muestra en la Tabla 5.50.

En la Tabla 5.51 se presenta la correspondencia existente entre el programa de la parte práctica y los objetivos prácticos perseguidos en esta asignatura.

Laboratorio (30 Horas)
Práctica 1. Aplicación de métricas (4 horas)
Práctica 2. Estimación de coste y esfuerzo de un proyecto (8 horas).
Práctica 3. Planificación temporal del proyecto (18 horas).

Tabla 5.50. Estructura del programa de prácticas de la asignatura Administración de Proyectos Informáticos (3 créditos)

Elemento Docente	Objetivos
Práctica 1	P8, H1, H2, H4
Práctica 2	P3, H1, H2, H4
Práctica 3	P3, H1, H2, H4

Tabla 5.51. Correspondencia entre el temario de prácticas y los objetivos prácticos de la asignatura

En las figuras 5.16 y 5.17 se muestra el reparto de horas, absoluto y porcentual respectivamente, entre las prácticas de la asignatura *Administración de Proyectos Informáticos*.

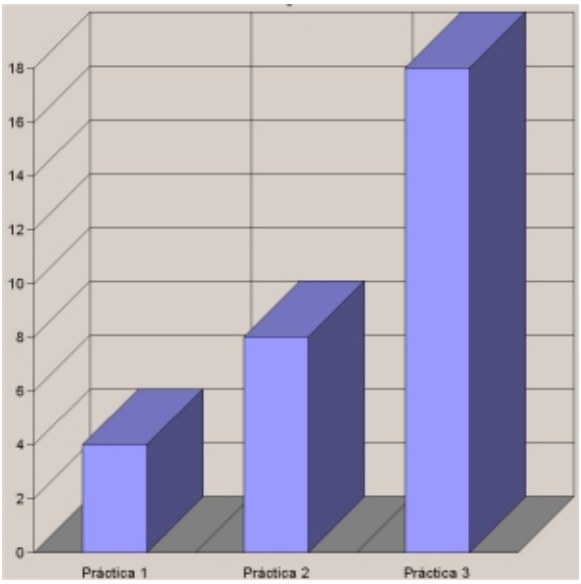


Figura 5.16. Reparto de las horas entre las prácticas de Administración de Proyectos Informáticos

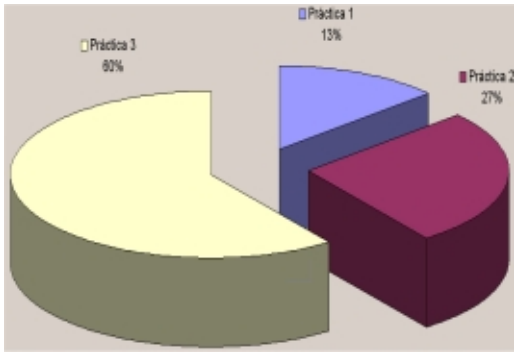


Figura 5.17. Reparto porcentual de las horas prácticas de Administración de Proyectos Informáticos

Evaluación de la parte práctica

La forma principal de evaluar la parte práctica de esta asignatura es mediante la realización de tres prácticas obligatorias.

La nota será la media de las calificaciones obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individual, aunque el trabajo haya sido realizado en pareja.

En la Figura 5.18 se muestra la fórmula que se utiliza para calcular la nota final de la asignatura, donde se puede apreciar el peso que tiene la nota de la práctica obligatoria y los informes de los talleres, realizados voluntariamente.

Si (Teoría ≥ 4,75) **y** (***Práctica*** ≥ 5.0)
    Nota Final = ((Teoría*0,6) + (***Práctica****0,3)) * 10/9 + Nota trabajos
Sino
    Ø
Fin si

Figura 5.18. Influencia de la nota en la parte práctica en la nota final de Administración de Proyectos Informáticos

Bibliografía básica de referencia

La siguiente lista refleja los títulos recomendados para afrontar la parte práctica de la asignatura. Algunos de ellos ya se recomendaron como bibliografía básica general de la asignatura,

mientras que otros van más encaminados a ser referencia para uno de los entornos utilizados, aunque los alumnos utilizarán más habitualmente la ayuda en línea disponible.

- **Fenton, N. E., Pfleeger, S. L.** "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.
- **Ministerio de Administraciones Públicas.** "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.
- **Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.
- **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.
- **USC.** "*COCOMO II Model Manual*". University of Southern California. USC COCOMO II.1998.0. http://sunset.usc.edu/research/COCOMOII/index.html. 1998.
- **USC.** "*USC COCOMO II User's Manual*". University of Southern California. USC COCOMO II.1999.0. http://sunset.usc.edu/research/COCOMOII/index.html. 1999.

5.4.2.3 Desarrollo comentado del programa

Durante el desarrollo de las prácticas se aplicarán diferentes actividades de la gestión de proyectos explicadas en las clases teóricas. Para llevarlas a cabo se basarán en uno de los proyectos realizados por ellos mismos el curso anterior en la asignatura *Análisis de Sistemas*. Deben realizar la estimación de costes y esfuerzo y la planificación temporal de dicho proyecto. La totalidad de los créditos prácticos se han distribuido en tres bloques, uno inicial de sólo cuatro horas de duración que servirá para familiarizarse con las técnicas de medición, imprescindibles para la realización de las estimaciones que se requieren en los bloques posteriores, en el primero de ellos se estimarán los recursos necesarios para desarrollar el proyecto, y en el segundo se realizará la planificación temporal basándose en dichos recursos. Aunque las tres partes tienen cierta continuidad, se han propuestos tres prácticas distintas debido a que en cada una de ellas se utilizan herramientas diferentes.

Práctica 1. Aplicación de métricas

En esta práctica, de tan sólo cuatro horas de duración, se utilizan herramientas automatizadas para contabilizar el número de líneas de código a partir de ficheros de código fuente de programas realizados en diferentes lenguajes de programación y para calcular el número de puntos de función de algunos módulos del sistema con el que van a trabajar posteriormente.

La contabilización del número de líneas de código la llevarán a cabo con la herramienta multiplataforma de libre disposición **CodeCount** (*copyright* 1998 *University of Southern California Center for Software Engineering*). Aunque admite código escrito en muchos

lenguajes de programación en esta práctica sólo se usará con programas C/C++ y Java. Los ficheros fuente pueden contener líneas en blanco, comentarios, directivas de compilación, líneas de datos y líneas ejecutables. El resultado se expresa en SLOC (*Source Line Of Code*), distinguiendo entre SLOC físicas cuya definición coincide con el concepto de DSI (*Deliverable Source Instructions*) y SLOC lógicas que corresponden a la suma de líneas del tipo: directivas de compilación, líneas de datos y líneas ejecutables.

El cálculo de puntos de función se realiza con la herramienta **COCOMO II** (*copyright* 1990-1999 *University of Southern California*) que se utiliza también en la segunda práctica, siendo, al igual que la anterior, de libre disposición. El método utilizado es el propuesto por Albretch, que ya ha sido explicado en las clases teóricas, al que se le puede introducir una modificación para considerar la reutilización (enfoque COCOMO).

Ambas herramientas (**CodeCount** y **COCOMO II**) se pueden obtener en la URL http://sunset.usc.edu/available_tools/index.html.

Esta práctica representa el **13%** del tiempo dedicado a las prácticas de la asignatura.

Práctica 2. Estimación de coste y esfuerzo de un proyecto

En la segunda práctica, de ocho horas de duración, se completa la estimación del tamaño de todos los módulos del sistema, que fue iniciada en el punto anterior. Dicha estimación será la base de la aplicación del modelo **COCOMO** [Boehm, 1981; Boehm et al, 1995; Boehm et al., 2000] con la herramienta **COCOMO II.**

El modelo se aplica en dos niveles del desarrollo del proyecto: modelo de diseño inicial (*early design*) y el modelo post arquitectura. En el primero se parte de estimaciones de tamaño del software en forma de puntos de función, mientras que en el segundo se recomienda estimar el número de líneas de código. En ambos modelos debe realizarse la descomposición del sistema en módulos que se estiman de forma individual. Los alumnos deben seleccionar los valores de los factores de escala y de las guías de coste justificando en todo momento su elección. Para ello deben hacer uso de las tablas contenidas en el manual de definición del modelo [USC, 1998]. Al finalizar la estimación en los dos niveles de abstracción comentados obtendrán los informes que la herramienta genera automáticamente y confrontarán los resultados obtenidos.

Esta práctica representa el **27%** del tiempo dedicado a las prácticas de la asignatura.

Práctica 3. Planificación temporal del proyecto

Las actividades prácticas de esta asignatura se completan con la planificación temporal de todas las tareas en las que se ha descompuesto el proyecto. Este bloque práctico es el que más trabajo requiere, por lo que su duración es bastante mayor que la de los anteriores, se le dedican nueve sesiones de dos horas cada una, reservando una hora de las dos primeras sesiones para la explicación del manejo de la herramienta. Se utiliza en este caso la herramienta automatizada **Microsoft Project** sobre plataforma **Windows**. La razón de su elección se debe a las múltiples posibilidades que ofrece en la automatización de las actividades de planificación, refinamiento del plan, seguimiento, gestión de proyectos múltiples, uso de datos compartidos, generación de informes...

Los alumnos deben realizar las siguientes tareas:

- Elaboración de un calendario de trabajo.
- Identificación de las fases del proyecto y descomposición en tareas y subtareas si se considera necesario.
- Asignación de recursos a las diferentes tareas.
- Estimación de las duraciones de las mismas.
- Identificación de hitos.
- Establecimiento de dependencias entre las tareas. Éstas pueden ser de tres tipos (*Start-to Start*, *Finish-to-Finish* y *Finish-to Start*). La herramienta permite también la introducción de solapamiento y desplazamiento entre tareas dependientes, tal como se refleja en el método ROY, estudiado en teoría.
- Revisión y prueba de los resultados obtenidos automáticamente, que se muestran tanto en forma tabular como gráfica (Diagramas de Gantt, PERT...).
- Refinamiento de la planificación considerando diferentes alternativas.

Esta práctica representa el **60%** del tiempo dedicado a las prácticas de la asignatura.

5.5 Ingeniería del Software en el Tercer Ciclo

Como se comentó en el Capítulo 2 de este Proyecto Docente e Investigador, con la creación del Departamento de Informática y Automática en octubre de 1996 se ponen en marcha los estudios de doctorado en Informática en la Universidad de Salamanca.

Actualmente, se está desarrollando el período de docencia correspondiente al bienio 2001-2003. En la Tabla 2.5 se recogen los cursos ofertados para dicho período, y una información más amplia se puede obtener en la web del Departamento de Informática y Automática [DPTOIA, 2002].

De los diferentes cursos impartidos hay dos que tienen una relación especialmente estrecha con la Ingeniería del Software, éstos son *Ingeniería del Software Avanzada* y *Tecnología de Objetos Aplicada a la Creación de Aplicaciones Distribuidas y de Tiempo Real*, que se describen someramente a continuación.

5.5.1 Ingeniería del Software Avanzada

Este curso de doctorado consta de tres créditos, y cuenta entre sus objetivos los siguientes:

- Sentar una base común con respecto a la Ingeniería del Software para los alumnos de tercer ciclo.
- Presentar temas de investigación actuales en la Ingeniería del Software.
- Introducir el proceso de desarrollo basado en la reutilización del software.

Puede llamar la atención en primer objetivo, pero dada la naturaleza dispersa del alumnado de tercer ciclo en el Departamento de Informática y Automática, tanto por procedencia geográfica como por estudios previamente cursados, se encuentra que los alumnos suelen carecer de una base sólida sobre la que impartir un curso con contenidos avanzados en la disciplina de Ingeniería del Software.

El temario que actualmente se imparte se presenta a continuación, aunque cabe decir que sus contenidos son tendentes a sufrir modificaciones en próximos bienios.

Tema 1: UML

Génesis y evolución de UML. Visión general de UML. Las vistas. Vista estática. Vista de casos de uso. Vista de la máquina de estados. Vista de actividad. Vista de interacción. Vistas físicas. Vista de gestión de modelos.

Tema 2. Proceso Unificado de Desarrollo de Software

Evolución. ¿Qué es el Proceso Unificado? Características. La vida del Proceso Unificado. El producto. El proceso. Proceso dirigido por casos de uso. Proceso centrado en la arquitectura. Flujos de trabajo.

Tema 3. Introducción a la Reutilización del Software

Introducción. Ciclo de vida de la reutilización. Reutilización y Orientación a Objetos. Reutilización en las fases del ciclo de vida.

Tema 4. Proceso de Desarrollo y reutilización

Introducción. Cambios en los procesos. Ingeniería de dominios y de sistemas de aplicación. Arquitectura de componentes y aplicaciones. Procesos de ingeniería del software. Aplicaciones y sistemas de componentes. Reutilización de casos de uso.

Tema 5. Ingeniería de dominio

Introducción. Ingeniería de dominio. Líneas de productos. Componentes. *Frameworks*. Mecanos.

Como bibliografía básica para el curso se recomienda la siguiente:

- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.
- **García Peñalvo, F. J., Barras, J.-A., Laguna Serrano, M. Á., Marqués Corral, J. M.** "*Líneas de Productos, Componentes, Frameworks y Mecanos*". Informe Técnico (DPTOIA-IT-2002-004), Universidad de Salamanca (España). http://tejo.usal.es/inftec/DPTOIA-IT-2002-004.pdf. Marzo, 2002.
- **Jacobson, I., Booch, G., Rumbaugh, J.** "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000.
- **Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G.** "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.
- **Jacobson, I. Ericsson, M., Jacobson, A.** "*The Object Advantage-Business Process Reenginering with Object Technology*". Addison Wesley, 1994.
- **Jacobson, I., Griss, M., Patrik, J.** "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press, 1997.
- **OMG.** "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.
- **Prieto-Díaz, R., Arango, G. (Editors).** "*Domain Analysis and Software Systems Modeling*". IEEE Computer Society Press, 1991.
- **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.

5.5.2 Tecnología de Objetos Aplicada a la Creación de Aplicaciones Distribuidas y de Tiempo Real

Este curso de doctorado consta de tres créditos, y cuenta entre sus objetivos los siguientes:

- Introducir al alumno en la aplicación de la tecnología de objetos para el diseño de aplicaciones distribuidas y con características de tiempo real.
- Presentar el *framework* CORBA.
- Ofrecer los fundamentos básicos para el desarrollo de una aplicación distribuida usando CORBA.

El temario que actualmente se imparte se presenta a continuación, aunque cabe decir que sus contenidos son tendentes a sufrir modificaciones en próximos bienios.

Tema 1. Arquitecturas software distribuidas

Aplicaciones cliente/servidor distribuidas. Servicios de un sistema operativo distribuido. Introducción a los patrones de diseño. Patrón *broker*.

Tema 2. Introducción a CORBA

Introducción. ¿Qué es CORBA? Arquitectura de referencia OMG. Arquitectura CORBA. CORBA IDL. Dinámica de los sistemas basados en *brokers*. Dinámica de los sistemas basados en CORBA.

Tema 3. Aspectos de programación en CORBA

Introducción. Conceptos básicos. Arquitectura CORBA. Desarrollo de un ejemplo.

Tema 4. CORBA en tiempo real

¿Por qué RT-CORBA? El estándar RT-CORBA. TAO.

Como bibliografía básica para el curso se recomienda la siguiente:

- **Aklecha, V.** "*Object-Oriented Frameworks Using C++ and CORBA Gold Book*". Coriolis Technology Press, 1999.
- **Buschmann, F., Meunier, R., Rohnert H., Sommerlad P., Stal, M.** "*Pattern Oriented Software Architecture: A System of Patterns*". John Wiley & Sons, 1996.
- **García Peñalvo, F. J., González González, J., Álvarez Navia, I., Moreno García, Mª N., Curto Diego, B., Moreno Rodilla, V.** "*Fundamentos para el Desarrollo de Aplicaciones Distribuidas Basadas en CORBA*". Informe Técnico

(DPTOIA-IT-2002-001), Universidad de Salamanca (España). http://tejo.usal.es/inftec/DPTOIA-IT-2002-001.pdf. Febrero, 2002.

- **Gomaa, H.** "*Designing Concurrent, Distributed, and Real-Time Applications with UML*". Addison-Wesley, 2000.
- **Henning, M., Vinoski, S.** "*Programación Avanzada en CORBA con C++*". Addison Wesley, 2002.
- **Lewis, T. G.** "*Where Is Client/Server Software Headed*". IEEE Computer, 28(4):49-55. April 1995.
- **Microsoft Corporation.** "*DCOM Technical Overview*". White Paper, Microsoft Developer Network, 1996.
- **Mowbray, T. J., Malveau, R. C.** "*CORBA Design Patterns*". John Wiley & Sons, 1997.
- **Object Management Group.** "*CORBA 2.6 Specification*". Document formal/01-02-35. 2001.
- **Schmidt, D., Levine, D. L., Cleeland, C.** "*Architectures and Patterns for Developing High-performance, Real-time ORB Endsystems*". In *Advances in Computers*, Academic Press, 1999.
- **Schmidt, D., Stal., M., Rohnert, H., Buschmann, F.** "*Pattern-Oriented Software Architecture. Patterns for Concurrent and Networked Objects*". Vol. 2. John Wiley & Sons, 2000.
- **Siegel, J. (Editor).** "*CORBA 3 Fundamentals and Programming*". 2nd edition. Jonh Wiley & Sons, 2000.
- **SUN Microsystems.** "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 12/4/2002]. March 2002.
- **Tanenbaum, A. S.** "*Modern Operating Systems*". Prentice-Hall, 1992.
- **Tari, Z., Bukhres, O.** "*Fundamentals of Distributed Object Systems. The CORBA Perspective*". John Wiley & Sons, 2001.

5.6 Referencias

[Abran et al., 2001] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair). "*Guide to the Software Engineering Body of Knowledge SWEBOK*". IEEE-CS Press, 2001.

[ACM/IEEE-CS, 2001] The Joint Task Force on Computing Curricula: IEEE Computer Society and Association for Computing Machinery. "*Computing Curricula 2001 – Computer Science*". Final Report, http://www.computer.org/education/cc2001/final/cc2001.pdf. [Última vez visitado, 27-12-2001]. December 15, 2001.

[Aldis, 1995] Aldis, M. "*A Manager's Guide to PCTE. How To Control Software Cost and Quality Using Open Tool Integration, Frameworks and Repositories*". PCTE Association, 1995.

[Ambler, 1994] Ambler, S. W. "*In Search of a Generic SDLC for Object Systems*". Object Magazine, 4(6): 76-78, 1994.

[Andreu et al., 1996] Andreu, R., Ricart, J., Valor, J. "*Estrategia y Sistemas de Información*". 2ª Ed. McGraw-Hill (serie de management), 1996.

[Arango y Prieto-Diaz, 1991] Arango, G., Prieto-Díaz, R. "*Domain Analysis: Concepts And Research Directions*". In *Domain Analysis and Software Systems Modeling*. Prieto-Díaz, R., Arango, G. (eds.). IEEE-CS Press, 1991.

[Babich, 1986] Babich, W. A. "*Software Configuration Management Coordination for Team Productivity*". Addison Wesley, 1986.

[Baetjer, 1998] Baetjer, H., Jr. "*Software as Capital*". IEEE Computer Society Press, 1998.

[Basili y Rombach, 1988] Basili, V. R., Rombach, H. D. "*The TAME Project: Towards Improvement-Oriented Software Environments*". IEEE Transaction on Software Engineering, 14(6):758-773, 1988.

[Basili y Turner, 1975] Basili, V., Truner, A. "*Iterative Enhancement: A practical Technique for Software Development*". IEEE Transactions on Software Engineering, 1(4):390-396, 1975.

[Basili y Weiss, 1984] Basili, V. R., Weiss, D. "*A Methodology for Collecting Valid Software Engineering Data*". IEEE Transaction on Software Engineering, 10(6):728-738, 1984.

[Beck y Cunningham, 1989] Beck, K., Cunningham, W. "*A Laboratory for Teaching Object-Oriented Thinking*". In Proceedings of the 1989 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications (October 2 - 6, 1989, New Orleans, LA USA); Reprinted in SIGPLAN Notices, 24(10):1-6. 1989.

[Berard, 1992] Berard, E. W. "*Comparison of Object-Oriented Development Methodologies*", Berard Software Engineering Inc., Gaithersburg, MD, 1992.

[Bertalanffy, 1968] Bertalanffy, L. von. "*General Systems Theory: Foundations, Development, Applications*". George Brazillier, New York, 1968.

[Bjorner y Jones, 1978] Bjorner, D., Jones, C. "*The Vienna Development Method*" NY, Springer Verlag, 1978.

[Blasco, 2000] Blasco Martín, V. "*ER CASE*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Septiembre de 2000.

[BOE, 1997] *Boletín Oficial del Estado de 4 de Noviembre de 1997*; Resolución de 15 de Octubre, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero Técnico en Informática de Sistemas.

[BOE, 1999] *Boletín Oficial del Estado de 1 de Julio de 1999*; Resolución de 10 de junio de 1999, de la Universidad de Salamanca, por la que se publica el Plan de Estudios de Ingeniero en Informática (2º ciclo).

[Boehm, 1981] Boehm, B. W. "*Software Engineering Economics*". Prentice Hall, Englewood Cliffs, NJ, 1981.

[Boehm, 1988] Boehm, B. W. "*A Spiral Model of Software Development and Enhancement*". IEEE Computer, 21(5):61-72, May 1988.

[Boehm et al., 1980] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., Mac Leod, G. J., Merritt, M. J. "*Characteristics of Software Quality*". North-Holland, 1980.

[Boehm et al., 1984] Boehm, B. W., Gary, T. E., Seewaldt, T. "*Prototyping versus Specifiying: A Multi-Project Experiment*". IEEE Transactions on Software Engineering, SE-10(3): 290-303, 1984.

[Boehm et al., 1995] Boehm, B. W., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R. "*Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*". Annals of Software Engineering, Vol. 1:57-94, 1995.

[Boehm et al., 2000] Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., Abts, C. "*Software Cost Estimation with Cocomo II*". Prentice Hall, 2000.

[Boehm y Ross, 1989] Boehm, B., Ross, R. "*Theory-W Software Project Management: Principles and Examples*". IEEE Transaction on Software Engineering, 15(7):902-916. July 1989.

[Bollinger et al., 2001] Bollinger, T., Gabrini, P., Martin, L. "*Software Construction*". Chapter 4 in [Abran et al., 2001].

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Booch et al., 1996] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

[Booch y Rumbaugh, 1995] Booch, G., Rumbaugh, J. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

[Brackett, 1990] Brackett, J. W. "*Software Requirements*". SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January 1990.

[Brodsky, 1999] Brodsky, S. "*XMI Opens Application Interchange*". White Paper. IBM. March 1999.

[Brooks, 1995] Brooks, F. "*The Mythical Man-Month*". Anniversary Edition, Addison-Wesley, 1995.

[Bruegge y Dutoit, 2000] Bruegge, B., Dutoit, A. H. "*Object-Oriented Software Engineering. Conquering Complex and Changing Systems*". Prentice Hall, 2000.

[Bruno y Manchetto, 1986] Bruno, G., Manchetto, C. "*Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems*". IEEE Transactions on Software Engineering, 12(2):346-357, 1986.

[Budgen, 1999] Budgen, D. "*Software Design Methods: Life Belt or Leg Iron*". IEEE Software, 16(5):133-136. September/October 1999.

[Cañete et al., 1999] Cañete, J. M., Galán, F. J., Toro M. "*A Proposal for the Formalization of the OCL Language Based on Algebraic Specifications*". In the proceedings of the 4th Workshop MENHIR, F. J. García, J. M. Marqués (eds.). (Sedano, Burgos, Spain, May 1999). Pages 80-84. 1999.

[Carrington, 2001] Carrington, D. "*Software Engineering Tools and Methods*". Chapter 10 in [Abran et al., 2001].

[Clavel et al., 1996] Clavel, M., Eker, S., Lincoln, P., Meseguer, J. "*Principles of Maude*". In Proceedings of First International Workshop on Rewriting Logic and its Applications, WRLA'96, J. Meseguer (editor). (Asilomar, California, September 3-6, 1996). Pages 65-89. Volume 4 of Electronic Notes in Theoretical Computer Science. Elsevier, September 1996.

[Clavel et al., 1998] Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P. "*An Introduction to Maude (beta version)*". Manuscript, SRI International, March 1998.

[Coad y Yourdon, 1991] Coad, P., Yourdon, E. "*Object-Oriented Analysis*". 2nd Edition. Yourdon Press, 1991.

[Conde, 2002] Conde González, M. Á. "*Left CASE: Componente DFD v1.0*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Marzo de 2002.

[Coolahan y Roussopoulos, 1983] Coolahan J. E., Roussopoulos, N. "*Timing Requirements for Time Driven Systems Using Aumengted Petri Net*". IEEE Transactions on Software Engineering, 9(5):603-616, 1983.

[Costa, 2001] Costa Alba, Á. "*Left CASE: Componente DTE v1.0*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Septiembre de 2001.

[Crosby, 1979] Crosby, P. "*Quality is Free*". McGraw-Hill, 1979.

[Cuesta, 2001] Cuesta Carranza, A. "*CRC CASE*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Marzo de 2001.

[Chen, 1976] Chen, P. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March 1976.

[Davis, 1982] Davis, A. M. "*Rapid Prototyping Using Executable Requirements Specifications*". ACM Software Engineering Notes, 7(5):39-44, 1982.

[Davis, 1992] Davis, A. M. "*Operational Prototyping: A New Development Approach*". IEEE Software, 9(5):70-78. September 1992.

[Davis y Sitaram, 1994] Davis, A., Sitaram, P. "*A Concurrent Process Model for Software Development*". Software Engineering Notes, ACM Press, 19(2):38-51. February 1994.

[Davis et al., 1997] Davis, G. B., Gorgone, J. T., Couger, J. D., Feinstein, D. L., Longenecker, Jr. H. E. (Editors). "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS y AITP, 1997.

[DeMarco, 1979] DeMarco, T. "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

[Deveaux et al., 1999] Deveaux, D., Fleurquin, R., Frison, P. "*Software Engineering Teaching: A 'Docware' Approach*". In Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education (ITiCSE '99). (June 27-July 1, 1999, Cracow, Poland). Pages 163-166. ACM. 1999.

[Donadi, 1992] Donadi, M. H. "*Teaching Practical Object-Oriented Software Engineering*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 251-256. ACM. 1992.

[DPTOIA, 2002] Departamento de Informática y Automática. "*Web del Departamento de Informática y Automática de la Universidad de Salamanca*". http://dptoia.usal.es, 2002.

[Durán, 2000] Durán Toro, A. "*Un Entorno Metodológico de Ingeniería de Requisitos para Sistemas de Información*". Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/publicaciones/tesis.pdf.zip. Septiembre de 2000.

[Durán, 2002] Durán Toro, A. "*REM Web Page*". Universidad de Sevilla. http://www.lsi.us.es/~amador/REM/REM_english_main.html. [Última vez visitada, 11/01/2002]. 2002.

[Durán et al., 2002] Durán, A., Ruiz, A., Bernárdez, B., Toro, M. "*Verifying Software Requirements with XSLT*". ACM Software Engineering Notes, 27(1):39-44. January 2002.

[Durán y Bernárdez, 2001a] Durán Toro, A., Bernárdez Jiménez, B. "*Metodología para el Análisis de Requisitos de Sistemas Software. (versión 2.2)*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. http://www.lsi.us.es/~amador/publicaciones/metodologia_analisis.pdf.zip. [Última vez visitado, 11-1-2002]. Sevilla, diciembre de 2001.

[Durán y Bernárdez, 2001b] Durán, A., Bernárdez, B. "*Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.2)*" Informe Técnico LSI-2000-10 (revisado), Universidad de Sevilla, octubre 2001. También disponible en http://www.lsi.us.es/~amador/#publicaciones/metodologia_elicitacion.pdf.zip [Última vez visitado: 8/1/2002], 2001.

[Durr y Katwijk, 1992] Durr, E. H., van Katwijk, J. "*VDM++ -- A Formal Specification Language for Object-oriented Designs*". In *Computer Systems and Software Engineering*, Proceedings of CompEuro'92. IEEE Computer Society Press. Pages 214-219. 1992.

[Durr y Plat, 1994] Durr, E. H., Plat, N. (Editors). "*VDM++ Language Reference Manual. Afrodite*". (ESPRIT-III project number 6500) Document AFRO/CG/ED/LRM/V9. Cap Volmac, 1994.

[EC, 2001] EC Institute. "*The EC Institute Body of Knowledge for Electronic Commerce*". January 2001.

[EIA CDIF Division, 1996] EIA CDIF Division. "*Conformance to the Standards Comprising the CDIF Family of Standards*". EIA CDIF Division, Formal Document, CDIF-DOC-N3. March 26, 1996.

[El Emam, 2001] El Emam, K. "*Software Engineering Process*". Chapter 9 in [Abran et al., 2001].

[El Emam et al., 1998] El Emam, K., Drouin, J.-N., Melo, W. (Editors). "*SPICE: The Theory and Practice of Software ProcessImprovement and Capability Determination*". IEEE CS Press, 1998.

[Everett, 1995] Everett, W. "*Reliability and Safety of Real-Time Systems*". IEEE Computer, 28(5):13-16. May 1995.

[Evergreen, 1994] Evergreen CASE Tools. "*Easy CASE Versión 4.1 para Windows. Guía de Acceso Rápido*". Evergreen CASE Tools, 1994.

[Fairley, 1985] Fairley, R. "*Software Engineering Concepts*". McGraw-Hill, 1985.

[Fell et al., 1996] Fell, H. J., Proulx, V. K., Casey, J. "*Writing across the Computer Science Curriculum*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 204-209. ACM. 1996.

[Fenton y Pfleeger, 1997] Fenton, N. E., Pfleeger, S. L. "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.

[Finkelstein, 1998] Finkelstein, A. "*Interoperable Systems: An Introduction*" Chapter 1 in *Information Systems Interoperativity*; B. Krämer, M. Papazoglou, H.-W. Schmidt editors. RSP-John Wiley and Sons Inc. Pages 1-9. 1998.

[Firesmith, 1993] Firesmith, D. G. "*Object Oriented Requirements Analysis and Logical Design*". Wiley, 1993.

[GAFC-USAL, 2001] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 2001/2002*". Ediciones Universidad de Salamanca, 2001.

[Gane y Sarson, 1981] Gane, C., Sarson, T. "*Análisis Estructurado de Sistemas*". Ateneo, 1981.

[Garbajosa y Bonilla, 1995] Garbajosa, J., Bonilla, A. "*Integración de Herramientas CASE*". Capítulo 22 en [Piattini y Daryanani, 1995]. 1995.

[García, 1999] García Peñalvo, F. J. "*Apuntes de la Asignatura Ingeniería del Software*". Revisión IV. Tercer curso de la Ingeniería Técnica en Informática de Sistemas de la Universidad de Salamanca. Diciembre, 1999.

[García, 2000] García Peñalvo, F. J. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[García, 2001] García Peñalvo, F. J. "*Docencia Práctica en los Laboratorios de las Ingenierías en Informática Apoyada en Herramientas CASE – Memoria de Resultados*". Departamento de Informática y Automática, Universidad de Salamanca. Consejería de Educación y Cultura de la Junta de Castilla y León. Noviembre de 2001.

[García et al., 2000a] García Peñalvo, F. J., Moreno García, Mª N., García-Bermejo Giner, J. R., Luis Reboredo, A. de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García et al., 2000b] García Peñalvo, F. J., Moreno García, Mª N., Moreno Montero, Á. Mª, González Talaván, G., Curto Diego, B. "*ADAM CASE. Utilización de Herramientas CASE Frontales en las Prácticas de Laboratorio de la Asignatura de Ingeniería del*

Software". Actas del 2º Simposio Internacional de Informática Educativa SIIE'2000. Editores M. Ortega y J. Bravo. (Puertollano - Ciudad Real), 15-17 de noviembre de 2000). Resumen en página 54 y ponencia en versión digital (CD-ROM). 2000.

[García et al., 2001a] García Peñalvo, F. J., Álvarez Navia, I., Hernández Herrero, J. B., González Pérez, S., Costa Alba, Á., Conde González, M. Á. "*Left CASE: Herramienta CASE basada en componentes Bonobo*". Actas del Segundo Taller de Trabajo en Ingeniería del Software basada en Componentes Distribuidos - IScDIS'01, desarrollado dentro de las VI Jornadas de Ingeniería del Software y Bases de Datos, JISBD'2001 (Almagro – Ciudad Real, 21-23 de noviembre de 2001). J. Hernández, J. Pavón, D. Sevilla y A. Vallecillo (Eds.). Informe Técnico UM-DITEC-2001-112001, Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia. (http://webepcc.unex.es/~juan/iscdis01/informeTecnico.html). Páginas 1-9. Diciembre, 2001.

[García et al., 2001b] García, F. J., Moreno, M. N., Moreno, A. Mª, González, G., Curto, B., Blanco, F. J. "*ADAM CASE. Using upper CASE tools in Software Engineering Laboratory*". In *Computers and Education: Towards an Interconnected Society*. M. Ortega and J. Bravo editors. Pages 149-158. Kluwer Academic Publishers. 2001.

[García y Pardo, 1998] García Peñalvo, F. J., Pardo Aguilar, C. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Ge y Sun, 2000] Ge, Y., Sun, J. "*E-Commerce and Computer Science Education*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 250-255. ACM Press. 2000.

[Gersting, 1994] Gersting, J. L. "*A Software Engineering 'Frosting' on a Traditional CS-1 Course*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 233-237. ACM. 1994.

[Gilb, 1988] Gilb, T. "*Principles of Software Engineering Management*". Addison-Wesley, 1988.

[Ginige y Murugesan, 2001] Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. January-March 2001.

[Glass, 1996] Glass, R. L. "*The Relationship between Theory and Practice in Software Engineering*". Communications of the ACM, 39(11):11-13. November 1996.

[Goguen et al., 2000] Goguen, J. A., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.-P. "*Introducing OBJ*". In *Software Engineering with OBJ: Algebraic Specification in*

Action, G. Malcolm (editor). Kluwer Academic Publisher, 2000. Also available in http://www-cse.ucsd.edu/users/goguen/ps/iobj.ps.gz. [Última vez visitado, 22-1-2002].

[Goguen y Malcolm, 1997] Boguen, J. A., Malcolm, G. "*Algebraic Semantics of Imperative Programs*". MIT Press, 1997.

[Gomaa, 1981] Gomaa, H. "*The Impact of Rapid Prototyping on Specifying User Requirements*". 5th International Conference of Software Engineering, Washington D.C., IEEE-CS Press, pages 333-342, 1981.

[González, 2001] González Pérez, S. "*Left CASE: Componente DER v1.0*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Septiembre de 2001.

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd edition. Addison-Wesley, 1994.

[Graham, 1995] Graham, I. M. "*Migrating to Object Technology*". Addison-Wesley, 1995.

[Granger y Little, 1996] Granger, M. J., Little, J. C. "*Integrating CASE Tools into the CS/CIS Curriculum*". In Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE'96. (June 2-6, 1996, Barcelona, Spain). Pages 130-132. ACM, 1996.

[Hall, 1990] Hall, A. "*Seven Myths of Formal Methods*". IEEE Software, 7(5):11-19. September/October 1990.

[Hanna, 1993] Hanna, M. "Maintenance Burden Begging for a Remedy". Datamation, pp. 53-63. April 1993.

[Harel, 1987] Harel, D. "*Statecharts: A Visual Formalism for Complex Systems*". Science of Computer Programing, 8(3):231-274. 1987.

[Harel et al., 1990] Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., Trakhtenbrot, M. B. "*STATEMATE: A Working Environment for the Development of Complex Reactive Systems*". IEEE Transactions on Software Engineering, 16(3):403-414. April 1990.

[Harel y Naamad, 1996] Harel, D., Naamad, A. "*The STATEMATE Semantics of Statecharts*". ACM Transactions on Software Engineering and Methodology, 5(4):293-333. October 1996.

[Harel y Politi, 1998] Harel, D., Politi, M. "*Modeling Reactive Systems with Statecharts: The STATEMATE Approach*". McGraw-Hill, 1998.

[Hatley y Pirbhai, 1987] Hatley, D. J., Pirbhai, I. "*Strategies for Real-Time System Specification*". Dorset House Publishing, 1987.

[Henderson-Sellers y Edwards, 1990] Henderson-Sellers, B., Edwards, J. M. "*The Object-Oriented Systems Life Cycle*". Communications of the ACM, 33(9):143-159. September 1990.

[Hernández, 2001] Hernández Herrero, J. B. "*Left CASE: Componente UML v1.0*". Proyecto Fin de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias - Universidad de Salamanca. Septiembre de 2001.

[Hoare, 1985] Hoare, C. A. R. "*Communicating Sequential Processes*". Prentice-Hall, 1985.

[Hybertson et al., 1997] Hybertson, D. W., Ta, A. D., Thomas, W. M. "*Maintenance of COTS-intensive Software Systems*". Journal of Software Maintenance: Research and Practice, 9(4):203-216, 1997.

[i-Logix, 1989] i-Logix. "*The Statemate Approach to Complex Systems*". I-Logix Inc., Burlington, MA, 1989.

[IEEE, 1998] IEEE. "*Standard for a Software Maintenance*". IEEE Std. 1219, 1998.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. 4–Volume Set*". IEEE Computer Society Press, 1999.

[IEEE-CS, 2001a] IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part I*". Vol.8, N1, January-March 2001.

[IEEE-CS, 2001b] IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part II*". Vol.8, N2, April-June 2001.

[IFAD, 1998] The VDM Tool Group. "*The IFAD VDM++ Language*". Technical Report, IFAD, October 1998.

[ISO, 1994a] ISO 9000. "*Quality Management and Quality Assurance Standards*". 1994.

[ISO, 1994b] ISO 9001. "*Quality Systems*". 1994.

[ISO/IEC, 1990] ISO/IEC. "I*nformation Technology - Information Resources Dictionary System (IRDS) - Framework*". ISO/IEC intl. Standard edition, 1990.

[ISO/IEC, 1991] ISO/IEC. "*Software Product Evaluation – Quality Characteristics and Guidelines for their Use*". ISO/IEC 9126, 1991.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[ISO/IEC, 1997] ISO/IEC. "*Software Engineering - Software Maintenance*". ISO/IEC. 14764, 1997.

[Jackson, 1995] Jackson, M. "*Critical Reading for Software Developers*". IEEE Software, 12(6):103-104. November 1995.

[Jacobson, 1987] Jacobson, I. "*Object Oriented Development in an Industrial Environment*". In Proceedings of the 1987 OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications. (October 4-8, 1987, Orlando, FL USA). Pages 183-191. ACM, 1987.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Jacobson et al., 1994] Jacobson, I. Ericsson, M., Jacobson, A. "*The Object Advantage-Business Process Reenginering with Object Technology*". Addison Wesley, 1994.

[Jacobson et al., 1997] Jacobson, I., Griss, M., Patrik, J. "S*oftware Reuse. Architecture, Process and Organization for Business Success*". ACM Press, 1997.

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

[Jensen, 1997a] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jensen, 1997b] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jensen, 1997c] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jones, 1991] Jones, C. B. "*Systematic Software Development Using VDM*". Prentice-Hall, 1991.

[Kelley, 1961] Kelley, J. E. "*Critical-Path Planning and Scheduling Mathematical Basis*". Operations Research, 9:296-320, 1961.

[Kernighan y Ritchie, 1988] Kernighan, B. W., Ritchie, D. M. "*The C Programming Language*". Prentice-Hall, 1988.

[Kurki-Suonio, 1993] Kurki-Suonio, R. "*Stepwise Design of Real-Time Systems*". IEEE Transactions on Software Engineering, 19(1):56-69. 1993.

[Lamsweerde, 2000] van Lamsweerde, A. "*Formal Specification: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 147-159. ACM Press, 2000.

[Land, 1982] Land, F. "*Adapting to Changing User Requirements*". Information and Management, 5:59-75, 1982.

[Lehman, 1997] Lehman, M. M. "*Laws of Software Evolution Revisited*". Academic Press Inc., 1997.

[Lehman y Belady, 1985] Lehman, M. M., Belady, L. A. "*Program Evolution: Processes of Software Change*". Academic Press, 1985.

[Liu y Shyamasundar, 1990] Liu, L. Y., Shyamasundar, R. K. "*Static Analysis of Real-Time Distributed System*". IEEE Transactions on Software Engineering, 6(3):373-388. 1990.

[MacDonell y Gray, 2001] MacDonell, S. G., Gray, A. R. "*Software Engineering Management*". Chapter 8 in [Abran et al., 2001].

[Malcom et al., 1959] Malcom, D. C., Roseboom, J. H., Clark, C. E., Fazar, W. "*Application of a Technique for Research and Development Program Evaluation*". Operations Research, 7:646-670, 1959.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[MAP, 2001] Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

[Marqués, 1999] Marqués Corral, J. M. "*Estándares de documentación. Laboratorio de Ingeniería del Software I. Ingeniería Técnica en Informática de Sistemas de la Universidad de Valladolid*". Departamento de Informática de la Universidad de Valladolid. Septiembre, 1999.

[Martin, 1991] Martin, J. "*Rapid Application Development*". Prentice Hall, 1991.

[McCabe et al., 1985] McCabe, T. J., et al. "*Structured Real-Time Analysis and Design*". In Proceedings of COMPSAC-85. Pages 40-51. IEEE, October 1985.

[McCall et al., 1977] McCall, J. A., Richards, P. K., Walters, G. F. "*Factors in Software Quality*". RADC TR-77-369, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055, 1977.

[McCauley et al., 1996] McCauley, R. A., Jackson, U., Manaris, B. "*Documentation Standards in the Undergraduate Computer Science Curriculum*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 242-246. ACM. 1996.

[McDermid y Rook, 1993] McDermid, J., Rook, P. "*Software Development Process Models*". In Software Engineer's Reference Book, CRC Press, 1993.

[McDonald y McDonald, 1993] McDonald, G., McDonald, M. "*Developing Oral Communication Skill of Computer Science Undergraduates*". In Proceedings of the Twenty-Fourth SIGCSE Technical Symposium on Computer Science Education, SIGCSE'93. (Feb. 18-19, 1993, Indianapolis, IN, USA). Pages 279-282. ACM. 1993.

[Meyer, 1988] Meyer, B. "*Object Oriented Software Construction*". Prentice Hall, 1988.

[Microsoft, 1996] Microsoft Corporation. "*DCOM Technical Overview*". White Paper, Microsoft Developer Network, 1996.

[Michalski et al., 1997] Michalski, R. S., Bratko, I., Kubat, M. "*Machine Learning and Data Mining. Methods and Applications*". John Wiley and Sons, 1997.

[Moder et al., 1983] Moder, J. J., Philips, C. R., Davis, E. W. "*Project Management with CPM, PERT and Precedence Diagramming*". 3rd Edition. VanNostrand Reinhold, 1983.

[Moitra, 1999] Moitra, D. "*Software Engineering in the Small. Practical Software Engineering and Management*". IEEE Computer, 32(10):39-40. October 1999.

[Monarchi y Puhr, 1992] Monarchi, D. E., Puhr, G. I. "*A Research Typology for Object-Oriented Analysis and Design*". Communicatios of the ACM, 35(9):35-47. September 1992.

[Nachbar, 1998] Nachbar, D. "*Bringing Real-World Software Development into the Classroom: A Proposed Role for Public Software in Computer Science Education*". In Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '98). (February 25 - March 1, 1998, Atlanta, GA – USA). ACM. Pages 171-175. 1998.

[Nerson, 1992] Nerson, J. "*Applying Object-Oriented Analysis and Design*". Communications of the ACM, 35(9):63-74. September 1992.

[Nierstrasz, 1992] Nierstrasz, O. M. "*Component Oriented Software Development*", Communications of the ACM, 35(9):160-165. September 1992.

[OBJ, 2002] "*The OBJ Family*". http://www-cse.ucsd.edu/users/goguen/sys/obj.html. [Última vez visitado, 22/01/2002]. 2002.

[OMG, 2001a] Object Management Group. "*CORBA 2.6 Specification*". Document formal/01-02-35. 2001. http://www.omg.org/cgi-bin/doc?formal/01-12-35. [Última vez visitado, 23/1/2002]. December 2001.

[OMG, 2001b] Object Management Group. "*Meta Object Facility (MOF) Specification. Version 1.3.1*". Object Management Group Inc. November 2001.

[OMG, 2001c] Object Management Group. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc.

http://www.celigent.com/omg/umlrtf/artifacts.htm. [Última vez visitado, 01/09/2001]. September 2001.

[Osborne y Chifofsky, 1990] Osborne, W. M., Chifofsky, E. J. "*Fitting Pieces to the Maintenance Puzzle*". IEEE Software, 7(1):11-12. January 1990.

[Paulk et al., 1993a] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

[Paulk et al., 1993b] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

[Peterson, 1977] Peterson, J. "*Petri Nets*". ACM Computing Surveys 9(3):223-252, 1977.

[Peterson, 1981] Peterson, J. "*Petri Net Theory and the Modeling of Systems*". Prentice-Hall, 1981.

[Petri, 1962] Petri, C. A. "*Kommunikation mit Automaten*". Ph.D. Diss. University of Bonn. Bonn, FRG, 1962.

[Piattini y Daryanani, 1995] Piattini Velthuis, M. G., Daryanani Daryanani, S. N. "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

[Pigoski, 2001] Pigoski, T. M. "*Software Maintenance*". Chapter 6 in [Abran et al., 2001].

[Pohl et al., 1998] Pohl, P., Nuseibeh, B., Finkelstein, A., Kramer, J. "*Interoperability: A System Specification Perspective*". Chapter 11 in *Information Systems Interoperativity*, B. Krämer, M. Papazoglou, H-W. Schmidt editors. Pages 303-329. RSP-John Wiley and Sons Inc. 1998.

[Pressman, 2000] Pressman, R. S. "*Software Engineering: A Practitioner's Approach – European Adaptation*". 5th Edition. McGraw-Hill, 2000.

[Rasala, 1997] Rasala, R. "*Design Issues in CS Education*". SIGCSE Bulletin. December 1997.

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Rout, 1995] Rout, T. P. "*SPICE: A Framework for Software Process Assessment*". Software Process Improvement and Practice, 1(1):57-66, 1995.

[Roy, 1960] Roy, B. "*Contribution de la Théorie des Graphes à l'étude des Problèmes dordennancement*". Congreso de International Federation of Operations Research Society (IFORS), 1960.

[Rumbaugh, 1992] Rumbaugh, J. "*Over the Waterfall and Into the Whirlpool*", Journal of Object-Oriented Programming (JOOP), 5(2):23-26. May 1992.

[Rumbaugh, 1994] Rumbaugh, J. "*The Functional Model*". Rational Whitepapers - OMT Papers. Rational Software Corporation. http://www.rational.com. March 1994.

[Rumbaugh, 1995a] Rumbaugh, J. "*OMT: The Object Model*". Journal of Object-Oriented Programming (JOOP), 7(8):21-27. January 1995.

[Rumbaugh, 1995b] Rumbaugh, J. "*OMT: The Dynamic Model*". Journal of Object-Oriented Programming (JOOP), 7(9):6-12. February 1995.

[Rumbaugh, 1995c] Rumbaugh, J. "*OMT: The Functional Model*". Journal of Object-Oriented Programming (JOOP), 8(1):10-14. March-April 1995.

[Rumbaugh, 1995d] Rumbaugh, J. "*OMT: The Development Process*". Journal of Object-Oriented Programming (JOOP), 8(2):8-16,76. May 1995.

[Rumbaugh, 1996] Rumbaugh, J. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., Booch, G. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

[Swanson, 1976] Swanson, E. B. "*The Dimensions of Maintenance*" Proceedings of the 2nd International Conference on Software Engineering. Pages 492-497. IEEE Computer Society Press, 1976.

[Sawyer y Kotonya, 2001] Sawyer, P, Kotonya, G. "*Software Requirements*". Chapter 2 in [Abran et al., 2001].

[Scott y Nisse, 2001] Scott, J. A., Nisse, D. "*Software Configuration Management*". Chapter 7 in [Abran et al., 2001].

[Schneidewind, 1985] Schneidewind, N. F. "*The State of Maintenance*". Conference on Software Maintenance-1985. Pages 114-119. IEEE. November 1985.

[Shaler y Mellor, 1992] Shaler, S., Mellor, S. "*Object Life Cycles: Modeling the World in States*". Prentice-Hall, 1992.

[Silva, 1985] Silva, M. "*Las Redes de Petri en la Automática y la Informática*". Editorial AC, 1985.

[Sobel, 2000] Sobel, A. E. K. "*Empirical Results of a Software Engineering Curriculum Incorporating Formal Methods*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 157-161. ACM Press. 2000.

[Spivey, 1988] Spivey, J. M. "*Understanding Z: A Specification Language and its Formal Semantics*". Cambridge University Press, 1988.

[Spivey, 1992] Spivey, J. M. "*The Z Notation: A Reference Manual*". Prentice-Hall, 1992.

[SRI, 2002] SRI International. "Maude Home Page". http://maude.csl.sri.com/. [Última vez visitado, 21/1/2002]. 2002.

[Stepney et al., 1992a] Stepney, S., Barden, R., Cooper, D. (Editors). "*Object Orientation in Z*". Workshops in Computing. Springer-Verlag, 1992.

[Stepney et al., 1992b] Stepney, S., Barden, R., Cooper, D. (Editors). "*A Survey of Object Orientation in Z*". Software Engineering Journal, 7(2):150-160, 1992.

[Stevens, 2001] Stevens, K. T. "*Experiences Teaching Software Engineering for the First Time*". In Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education – ITiCSE'01. (June 2001, Canterbury, UK). Pages 77-80. ACM Press. 2001.

[Szyperski, 1998] Szyperski, C. "*Component Software – Beyond Object-Oriented Programming*". Addison-Wesley, 1998.

[Takang y Grubb, 1997] Takang, A., Grubb, P. "*Software Maintenance Concepts and Practice*". International Thomson Computer Press, 1997.

[Tremblay, 2001] Tremblay, G. "*Software Design*". Chapter 3 in [Abran et al., 2001].

[Tuya, 2001] Tuya González, P. J. "*Manual de Procedimientos para las Prácticas de Ingeniería del Software I y II*". Versión 2.01 Universidad de Oviedo. http://edic.lsi.uniovi.es/isoft/procedimientos/index.html. [Última vez visitado, 14-1-2002]. Septiembre 2001.

[USC, 1998] USC. "*COCOMO II Model Manual*". University of Southern California. USC COCOMO II.1998.0. http://sunset.usc.edu/research/COCOMOII/index.html. 1998.

[Vaquero, 1999] Vaquero Sánchez, A. "*La Lengua Española en el Contexto Informático*". Revista de Enseñanza y Tecnología. ADIE. (13):5-12. Enero-Abril, 1999.

[Wallace y Reeker, 2001] Wallace, D., Reeker, L. "*Software Quality*". Chapter 11 in [Abran et al., 2001].

[Ward y Mellor, 1985] Ward, P. T., Mellor, S. J. "*Structured Development for Real-Time Systems. Volume 1: Introduction and Tools*". Yourdon Press/Prentice-Hall, 1985.

[WebE, 2001] "*Actas del I Taller sobre Ingeniería del Software Orientada al Web (Web Engineering*". Almagro, Ciudad Real, 22 de noviembre de 2001. http://www.dlsi.ua.es/webe01. [Última vez visitado, 7/1/2002]. 2001.

[Wilson y Krogh, 1990] Wilson, R. G., Krogh, B. H. "*Petri Net Tools for the Specification and Analysis of Discrete Controllers*". IEEE Transactions on Software Engineering, 16(1):39-50. 1990.

[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wirth, 1971] Wirth, N. "*Program Development by Stepwise Refinement*". Communication of the ACM, 14(4): 221-227. April 1971.

[Wood y Silver, 1995] Wood, J., Silver, D. "*Joint Application Development*". John Wiley & Sons, 1995.

[Yourdon, 1985] Yourdon, E. "*Structured Walkthroughs*", Prentice-Hall, 1985.

[Yourdon, 1989] Yourdon, E. "*Modern Structured Analysis*". Prentice-Hall, 1989.

[Yourdon Inc., 1993] Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

[Zucconi, 1989] Zucconi, L. "*Techniques and Experiences in Capturing Requirements for Real Time Systems*". ACM Software Engineering Notes, 14 (6):51-55. 1989.

Capítulo 6
Proyecto de Investigación

La labor de un Profesor Titular de Universidad no se centra exclusivamente en las labores docentes. La capacidad investigadora hacen de ésta otra de las tareas a desarrollar, lo que supone un complemento ideal a la formación del profesor y redundará sin duda en la calidad de la labor docente desarrollada. El desarrollo de programas de investigación es consustancial a la naturaleza y fines de la Universidad como institución. Tanto la Ley de Reforma Universitaria como la actual Ley Orgánica de Universidades establecen que los Departamentos son los órganos básicos encargados de organizar y desarrollar la investigación propia de sus áreas de conocimiento. En consecuencia, la existencia y desarrollo de este tipo de programas por parte de los Departamentos constituye una necesidad ineludible. A fin de mejorar los resultados y calidad de la actividad investigadora desarrollada, el programa de investigación ha de estructurarse en líneas de investigación. De esta forma se consigue, por una parte, que no se produzca una dispersión en los objetivos favoreciendo la eficacia de la labor investigadora y, por otra parte, una organización efectiva y operativa de los recursos humanos y materiales disponibles. A continuación se proponen dos Proyectos de Investigación que pueden enmarcarse en las líneas de "Aplicaciones web para Comercio Electrónico" y "Reutilización del Software" que se desarrollan actualmente en el Departamento de Informática y Automática en la Universidad de Salamanca.

6.1 Introducción

El desarrollo de programas de investigación es consustancial a la naturaleza y fines de la Universidad como institución. La Ley de Reforma Universitaria [BOE, 1983] establece en su Artículo octavo que los Departamentos son los órganos básicos encargados de organizar y desarrollar la investigación propia de sus Áreas de Conocimiento. En consecuencia, la existencia y desarrollo de este tipo de programas por parte de los Departamentos constituye una necesidad ineludible.

En el Artículo 108 del Capítulo III de los Estatutos de la Universidad de Salamanca [USAL, 1997] se dice: "*La Universidad de Salamanca promoverá, como uno de los objetivos esenciales de su actividad, la formación de investigadores y el fomento y coordinación de la investigación científica y técnica. La investigación se configura como fundamento de la docencia y como medio para el desarrollo científico y cultural de la sociedad*".

Actualmente, nadie pone en duda que la investigación es una tarea imprescindible para que una sociedad progrese, ya que, como consecuencia de ella, se innovan los sistemas industriales y culturales, se generan nuevos productos y se alcanza la independencia tecnológica. De aquí que las administraciones, castellano-leonesa, española y comunitaria, se ocupen de la investigación, cada una en su propio ámbito. Para ello actúan de dos formas principales:

- Organizando un Sistema de Investigación.
- Promoviendo líneas de actuación concretas, fundamentalmente mediante su financiación.

Una de las características más relevantes de la organización de la investigación es que ésta no es una tarea exclusiva de ningún sector o institución en particular. Por el contrario, involucra a múltiples sectores, públicos y privados, regionales, nacionales e internacionales. Una visión simplificada de estas relaciones se da en la Figura 6.1.

La Administración actúa junto con los sectores empresarial e industrial, los organismos públicos de investigación y los sistemas económico y científico internacionales. Cada uno de estos sectores interacciona con los demás poniendo de manifiesto las profundas relaciones existentes entre los conocimientos, su aplicación práctica y la sociedad.

Figura 6.1. Sectores que intervienen en la investigación

Un Proyecto de Investigación debe reflejar las áreas de trabajo en curso y las propuestas de trabajo futuro, pero es difícil exponer las áreas de trabajo sin explicar el porqué y el cómo llegaron a surgir. Un Proyecto de Investigación está siempre estrechamente ligado al currículo de quien lo realiza, pues es el camino recorrido el que mejor puede explicar las propuestas sobre caminos futuros.

Para que la investigación resulte eficaz, es necesario fijar unas directrices o líneas de investigación que encaucen el trabajo, aúnen esfuerzos y eviten la dispersión de objetivos. Sin menosprecio del trabajo personal, se tiene la creencia de que la presencia de grupos de trabajo con líneas de investigación definidas es la mejor manera de afrontar la investigación.

La tarea investigadora, y por lo tanto un programa de investigación, ha de contemplar los siguientes aspectos:

- La delimitación del área de investigación con una clara identificación de los objetivos perseguidos y las tareas a realizar por el equipo investigador involucrado.
- La formación de nuevos investigadores tanto para la propia Universidad como para la sociedad. Esto se consigue fomentando la inquietud investigadora de los alumnos mediante su participación en proyectos de investigación.
- El desarrollo de la investigación mediante la colaboración con empresas para conocer y solucionar problemas concretos, así como para favorecer los procesos de transferencia tecnológica.

La delimitación del área de investigación se realiza mediante la enumeración de los objetivos generales que pretenden conseguirse y los beneficios que el logro de los mismos supone. Esta formulación ha de completarse con el establecimiento de una metodología y plan

de trabajo que permita la consecución de los objetivos planteados. La metodología utilizada y aquí propuesta, implica la división en actividades asignadas a los miembros del grupo de investigación, con la previsión de determinados plazos de ejecución. La coordinación general se realiza por parte del responsable del proyecto a través de reuniones periódicas, semanales siempre que esto sea posible, de puesta en común. Las actividades se agrupan en tareas y módulos. Para cada módulo, además de un nombre identificativo, se especifican la duración prevista, una descripción general de la actividad a desarrollar y los resultados esperados. La descomposición en tareas del módulo establece el plan de trabajo detallado previsto. Para cada tarea, además de un nombre y código identificativo, se especifica la fecha de inicio y finalización previstas, el responsable de la tarea, las entradas o tareas que han de precederla, una breve descripción de las actividades a desarrollar y la relación de participantes en la misma.

La formación de nuevos investigadores está principalmente ligada a la realización de tesis doctorales. Esto implica que un programa de investigación ha de incluir el programa del curso o cursos de doctorado relacionados con la materia de investigación. Esto permite una rápida integración en el equipo de investigación de aquellos alumnos de doctorado que decidan realizar una tesis doctoral en el marco de la línea de investigación en la que se ubica este programa de investigación. En este apartado, además del programa de doctorado, hay que considerar la realización de los proyectos fin de carrera. Éstos permiten la participación de los alumnos en las labores de investigación, lo que les proporciona una formación científico-técnica complementaria muy interesante y descarga de cierto tipo de tareas al equipo investigador.

En esta memoria se proponen dos proyectos de investigación concretos, uno de los cuales enlaza directamente con las líneas de investigación existentes en el Departamento al que temporalmente está asignada la plaza a la que se concursa, mientras que otro amplía dichas líneas hacia los sistemas de comercio electrónico en el contexto de la web. Ya se ha trabajado en temas afines durante los últimos años y se cree que se pueden llevar a cabo con los medios humanos y materiales de que se dispone.

El resto del capítulo se organiza como sigue: a continuación, tras un breve comentario acerca de la Universidad y la Investigación, se presenta la trayectoria investigadora del candidato. Posteriormente se presentan los dos proyectos de investigación propuestos por el candidato.

6.2 La Universidad y la Investigación

La Universidad es parte esencial en todo sistema de investigación. En el esquema de organización (Figura 6.1), las Universidades nacionales están presentes en los organismos

públicos de investigación, constituyendo su cuerpo básico junto con el Consejo Superior de Investigaciones Científicas (CSIC), mientras que las Universidades extranjeras se incluyen en el sistema científico internacional. En lo que sigue se adopta la óptica universitaria de nuestro país al tratar la investigación.

La ordenación legal de la Universidad Española expresa claramente la importancia de la investigación en su seno. Así, la LRU [BOE, 1983] expresaba en su Artículo 1 que la investigación es uno de los medios fundamentales para realizar la educación superior, mientras que la actual LOU [BOCG, 2001] literalmente dice en su Artículo 39: "*La investigación, fundamento de la docencia, medio para el progreso de la comunidad y soporte de la transferencia social del conocimiento, constituye una función esencial de las Universidades*".

Esto es, la actual investigación universitaria tiene objetivos sociales, involucrándose con el entorno exterior, lo que le permite estar integrada en una organización como la de la Figura 6.1. De esta forma se supera la visión individualizada clásica del investigador que investiga sin mirar a su alrededor. Por otra parte, la investigación se rige en términos de autonomía universitaria en cuanto a la selección, formación y promoción del personal investigador, en la elaboración de planes de investigación y en la creación de estructuras que la soporten.

La investigación se convierte así en una función básica de la Universidad, junto con la docencia y la potenciación del desarrollo cultural, económico y social en su área de influencia.

De estas funciones, la docencia es la que hoy en día justifica, casi en su totalidad, la existencia de la Universidad en la sociedad española. Sin embargo, la investigación va impregnando cada vez más rápido el mundo universitario y haciendo que el personal universitario dedique más y más tiempo al desarrollo de la investigación.

Este no es el momento de analizar el porqué de esta situación. Simplemente remarcar que la docencia es una actividad fundamental para el profesor, pues le obliga a un esfuerzo de recopilación y síntesis de un área del saber y a una mejora de su capacidad de comunicación, que se refleja en una mejor capacitación para el desarrollo de su labor investigadora.

Sin embargo, en el ámbito de unos estudios universitarios la docencia no puede encerrarse en sí misma, pues terminaría aislándose de la realidad exterior y quedándose obsoleta en sus contenidos. Esto es particularmente cierto en aquellas áreas de conocimiento que están en rápida expansión, donde sólo mantenerse informado de sus continuos avances es una tarea ardua.

Es aquí donde la investigación encuentra su plena justificación en el marco de la Universidad. La investigación proporciona un medio continuo de renovación y ampliación de conocimientos. Esto repercutirá no sólo en el grupo investigador, sino que a través de la

docencia encontrará un cauce idóneo para la difusión de los resultados obtenidos. La investigación enriquece la actividad docente y la convierte en algo vivo, en constante evolución.

Los cursos de doctorado, por ejemplo, son un excelente punto de encuentro entre la docencia y la investigación. En estos cursos se debería enseñar a investigar e iniciar algunos caminos concretos. Al profesor le deben suponer siempre un esfuerzo de actualización y formación continua y, en muchos casos, también una labor de síntesis de los logros propios y ajenos.

La investigación requiere equipos humanos con elevada preparación científica, y la Universidad es una fuente inagotable de recursos humanos con capacidad investigadora. Esto evidencia que la docencia y la investigación sean tareas íntimamente ligadas y que difícilmente se justifican la una sin la otra dentro del marco de la Universidad.

La otra función que se le debe atribuir a la Universidad es la de potenciar el desarrollo de la sociedad. La Universidad no puede ser un ente aislado, cuya única relación con el exterior sea el continuo flujo de alumnos que pasan por sus aulas. Su labor ha de repercutir en el resto de la sociedad, que la soporta y mantiene. Esto debe reflejarse en, al menos, dos aspectos: selección de las líneas prioritarias de investigación y difusión de resultados.

La sociedad debe de fijar los objetivos prioritarios de la comunidad investigadora. La investigación, en general, sólo es posible si está sustentada por la investigación básica, cuyos resultados prácticos no son tangibles a corto plazo, pero que sienta los fundamentos de lo que en el futuro será investigación aplicada. Por tanto, el sistema de financiación público debe de garantizar la investigación en aquellos campos que, por su naturaleza más teórica, no puedan encontrar otras vías de financiación.

Por otra parte, es obligación de la Universidad el conseguir la adecuada difusión de su labor investigadora. Ésta se llevará a cabo mediante actividades como realización de proyectos conjuntos con la empresa, realización de ensayos de laboratorio, asesoramiento técnico, preparación de cursos y formación de personal.

Con este planteamiento, la Universidad y sus profesores están en condiciones de desarrollar una investigación moderna, internacionalmente competitiva y socialmente útil. Sin embargo, la realización práctica de esta actividad no está exenta de problemas, entre los que destacan:

- El exceso de carga docente.
- La abundante carga de gestión, tanto en lo que se refiere a la administración normal del Departamento como en la de los proyectos de investigación.
- La falta de uniformidad en la concepción de los equipos de investigación.

6.2.1 La investigación en grupo

En contraposición con la anticuada visión del investigador individual aislado, el grupo de investigación es la unidad básica para realizar investigación moderna y competitiva. Sin embargo, no hay que olvidar que el grupo reúne a un conjunto de personas, cada una con su formación, inclinación e intereses particulares. Una adecuada organización del grupo de investigación debe dar respuesta a esta doble realidad. Por un lado, tiene que constituir un equipo de trabajo donde todos colaboran en alcanzar objetivos comunes. Por otro lado, tiene que integrar a las distintas personas que lo componen, con sus peculiaridades concretas. El grupo debe, entonces, configurarse con una determinada organización jerárquica donde se recojan los distintos niveles de madurez de sus miembros, de los cuales surge naturalmente un director de grupo como el investigador con autoridad más reconocida.

Con el fin de evitar tensiones y agravios comparativos, que sólo perjudican la buena marcha del grupo, hay que dedicar el esfuerzo necesario para ordenar adecuadamente la promoción de los distintos investigadores. Distintos factores deben ser considerados para esa ordenación, entre los que cabe destacar: el respeto a las iniciativas personales, el estímulo de la transparencia entre los integrantes del grupo, la consideración de la antigüedad (como un elemento diferenciador más, pero nunca como el único argumento a utilizar) y, sobre todo, una equitativa distribución de las actividades cuyos criterios principales sean equilibrar las cargas docentes e investigadoras, atender a la formación e intereses de cada uno, y posibilitar la publicación de los trabajos.

Este último aspecto, la publicación de resultados es una de las obligaciones del investigador: dar a conocer a la comunidad internacional los resultados de interés que se hayan obtenido. Por esto, se ha de dedicar esfuerzo a la publicación de los resultados obtenidos, sean parciales o totales. Además, las publicaciones tienen un interés particular para el investigador, en tanto que sirven para calificar su labor académica (útil, pues, en la promoción profesional) y en tanto que es una de las formas usuales de evaluar los proyectos de investigación y, en consecuencia, de justificar la concesión de subvenciones a los mismos.

6.2.2 La materia a investigar

La materia a investigar es otro de los aspectos básicos en un proyecto de investigación. Una primera consideración de carácter fundamental es si debe enfocarse hacia la generalización o hacia la especialización. La primera orientación aventaja a la segunda en cuanto a la mayor consecución de valores formativos y a proporcionar un individuo más completo. En

contrapartida ofrece una aportación menor al progreso social, una mayor dificultad de aplicación práctica y unos resultados menos competitivos. Así, parece claro que tanto en la Ingeniería del Software como, en general, en todos los campos científicos y técnicos, actualmente hay que dirigir la investigación hacia la especialización.

La segunda consideración consiste en dilucidar sobre la conveniencia de la investigación básica o aplicada. El objetivo de la investigación básica es el incremento del conocimiento científico en general. Sus resultados característicos son el descubrimiento de nuevas teorías y regularidades legales, así como la explicación o predicción de fenómenos de una cierta clase. La investigación estratégica es investigación básica orientada a un determinado ámbito de la realidad, en el que se espera obtener conocimientos científicos nuevos que potencialmente sean interesantes para posibles aplicaciones tecnológicas.

El objetivo de la investigación aplicada es utilizar el método científico para incrementar el conocimiento de las propiedades y el comportamiento de sistemas concretos. La investigación aplicada se considera por definición orientada a objetivos específicos. Pero éstos pueden ser de dos tipos: objetivos de interés estrictamente científico y objetivos de interés tecnológico. En el primer caso se habla de investigación científica aplicada. En el segundo se suele hablar de investigación tecnológica. La diferencia estriba en los criterios de valoración de los resultados que en uno y otro caso se utilizan. En el primer caso lo que interesa es obtener conocimiento verdadero acerca de la realidad estudiada. En el segundo, conocimiento útil con vistas a la resolución de problemas prácticos, es decir, al diseño de sistemas tecnológicos.

Los límites entre investigación científica aplicada e investigación tecnológica son difusos. La razón es que el conocimiento científico de las propiedades de sistemas concretos puede ser por sí mismo útil para posibles desarrollos tecnológicos. Y a la inversa: los resultados obtenidos investigando sobre propiedades de sistemas artificiales pueden tener un valor científico intrínseco.

Por último, se va a delimitar más específicamente la materia objeto de investigación en este caso particular. Como ocurre con un Proyecto Docente, la concreción de la materia se va haciendo en pequeños pasos, desde lo más general, tratado aquí, hasta lo más específico, la descripción de los proyectos de investigación concretos. Siguiendo la metodología propuesta en [Tucker et al., 1990], es importante destacar la necesidad de contar con la fuerza científica de quienes realizarán la investigación y con las restricciones locales existentes donde se han de realizar esas actividades. En el contexto de este Proyecto Investigador se entiende como fuerza científica la experiencia, destreza e intereses de los investigadores y como restricciones locales

las derivadas del área de conocimiento de la plaza objeto de esta convocatoria y las procedentes del contexto universitario concreto donde se desarrollará la investigación.

6.2.3 La financiación

La actividad investigadora tiene asociados diversos costes sin los que es imposible realizarla. Su financiación es, por tanto, uno de los aspectos básicos a tener en cuenta. Las principales cuestiones cuyo coste económico debe ser atendido durante la investigación son:

- La adquisición de material inventariable. En éste se incluyen los equipos necesarios para la ejecución de la investigación y la bibliografía que suministra las fuentes de conocimiento para ella.
- La compra de material fungible, que abarca a los componentes de laboratorio, material de oficina, productos informáticos, documentación no inventariable, fotocopias...
- El pago de viajes y dietas que lleva consigo tanto la gestión de la investigación como algunas actividades asociadas, como la asistencia a congresos y cursos o las estancias en otros centros.
- Las remuneraciones personales. En ellas se incluyen, por una parte, el pago de personal contratado específicamente para la ejecución de un proyecto concreto (investigadores, técnicos y administrativos), y por otra, las remuneraciones adicionales del profesorado universitario participante, que la legislación actual permite.
- Otros gastos, entre los que cabe destacar dos cuya financiación, en la actualidad, no está satisfactoriamente resuelta a escala institucional: son los costes de amortización y de mantenimiento de los equipos utilizados.

Para afrontar estos gastos existen varias fuentes de financiación, que pueden agruparse de la siguiente forma:

- Aportaciones institucionales de carácter general. Fondos presupuestarios que las distintas instituciones (Universidad de Salamanca y Junta de Castilla y León, entre otras) asocian a la investigación sin que estén involucrados forzosamente a un proyecto de investigación concreto. Se trata de los fondos asignados a grupos o Departamentos, becas de formación del personal investigador, dedicados a actividades específicas (tales como la organización de congresos o la asistencia a

ellos)... En general, la cuantía económica de estas aportaciones es claramente insuficiente para mantener activo un campo de investigación en el caso de Departamentos Universitarios.

- Proyectos con organismos oficiales, como los del Plan Nacional de I+D. Estos proyectos, cuya duración usual es de dos o tres años, proporcionan una financiación suficiente y permiten una cierta planificación a medio plazo. Permiten financiar prácticamente todos los gastos salvo la remuneración personal.
- Proyectos con empresas. En general no financian dotaciones para material inventariable pero sí admiten las remuneraciones personales. En estos proyectos es posible becar/contratar alumnos en prácticas lo que supone un beneficio múltiple (al alumno, al grupo y a la empresa).

Una financiación muy adecuada para un grupo de investigación consolidado correspondería a un modo mixto de recibir fondos. Así, junto a proyectos específicos, debería disponer de un presupuesto consolidado de aportaciones institucionales.

6.2.4 La evaluación

El último aspecto básico que se discute es el de la evaluación de los proyectos de investigación. La actividad de evaluación fundamentalmente se centra en el comienzo y en el final del proyecto, evaluación previa y posterior, respectivamente. Los agentes evaluadores, obviamente, no participarán en el proyecto.

La evaluación previa suele basarse en dos criterios principales. Por un lado, la adecuación o idoneidad del proyecto a los propósitos del organismo financiador (sea público o privado). Por otro lado, la factibilidad de lo que se pretende obtener con el proyecto. Junto a estos criterios suelen considerarse otros como la probabilidad de revertir o transferir los beneficios resultantes, la ajustada financiación, la participación y coordinación de grupos..., incluyendo el valor curricular de los participantes.

La evaluación posterior se realiza sobre los resultados obtenidos en el proyecto. Éstos suelen consistir en informes y demostraciones. Generalmente los valores que se miden son la adecuación y grado de consecución de los resultados respecto a los objetivos planteados y el impacto de esos resultados, que típicamente se evalúa en función de la eficiencia del producto generado en proyectos de investigación aplicada y de las publicaciones y grados académicos generados (tesis doctorales, artículos, ponencias...) en los proyectos de investigación básica.

6.3 Trayectoria investigadora del candidato

La actividad universitaria de este candidato se inició en el mes de diciembre de 1995 en la Escuela Universitaria Politécnica de la Universidad de Burgos.

No obstante, la actividad investigadora propiamente dicha comienza en noviembre de 1996 cuando el candidato entra a formar parte del grupo de investigación GIRO (Grupo de Investigación en Reutilización y Orientación a Objetos) del Departamento de Informática de la Universidad de Valladolid, de reciente creación, dirigido por el Dr. D. José Manuel Marqués Corral.

La vinculación al grupo GIRO ha marcado, y sigue marcando, una de las principales líneas de investigación del candidato: la reutilización, y más concretamente, la reutilización sistemática del software soportada por el paradigma de la orientación al objeto. Esta línea de investigación está completamente vinculada a la disciplina de Ingeniería del Software.

La actividad investigadora del candidato en el seno del grupo GIRO hasta la actualidad se puede dividir en dos etapas, las cuales están marcadas por sendos proyectos CICYT: los proyectos MENHIR y DOLMEN.

El proyecto MENHIR (Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos) [MENHIR, 1998] - agosto de 1997 a julio de 2000 - fue un proyecto coordinado y financiado por la CICYT (TIC97-0593-C05-01), en el que participaron grupos de investigación de seis Universidades españolas (Politécnica de Valencia, Murcia, Sevilla, Granada, Castilla-La Mancha y Valladolid), coordinadas por el Dr. D. Isidro Ramos Salavert, Catedrático del Área de Lenguajes y Sistemas Informáticos de la Universidad Politècnica de València.

Dentro de este proyecto el grupo GIRO participaba con el subproyecto: *Definición y diseño de "mecanos" reutilizables como soporte a la construcción rápida de aplicaciones*, siendo el investigador principal el Dr. D. José Manuel Marqués Corral.

El proyecto MENHIR supone para el candidato la entrada en el mundo de la investigación universitaria y el contexto en el que realizar su tesis doctoral, por título *Modelo de reutilización soportado por estructuras complejas de reutilización denominadas mecanos* [García, 2000b], siendo ésta uno de los principales soportes a los resultados obtenidos por el grupo dentro del proyecto coordinado.

El proyecto DOLMEN (Objetos Distribuidos, Lenguajes, Modelos y ENtornos) - diciembre de 2000 a noviembre de 2003 - supone la continuación del proyecto MENHIR, siendo de nuevo un proyecto coordinado y financiado por la CICYT (TIC2000-1673-C06-01), en el que participan los mismos grupos y de nuevo coordinados por el Dr. D. Isidro Ramos Salavert.

En este proyecto el grupo GIRO participa con el subproyecto: *Aplicaciones del Modelo Mecano a la Ingeniería del Software basada en reutilización*, siendo el investigador principal el Dr. D. Miguel Ángel Laguna Serrano, del Departamento de Informática de la Universidad de Valladolid.

Con la incorporación del candidato al Departamento de Informática y Automática de la Universidad de Salamanca en octubre de 1998, se provoca la introducción de la reutilización sistemática del software en las líneas de investigación del Departamento, enunciada como *Ingeniería de Software basada en componentes: aspectos de reusabilidad, arquitectura multinivel, integración de métricas*, en la memoria de actividades del Departamento [DPTOIA, 2001].

El enfoque de reutilización que se aplica es el de las líneas de productos, y se hace en colaboración con el grupo de Robótica de este Departamento, que está interesado en la línea de investigación *Diseño de software para sistemas de fabricación flexible* [DPTOIA, 2001]. Fruto de esta colaboración se enuncia uno de los proyectos de investigación que se describen en esta memoria, y además se ha participado en los siguientes Contratos Artículo 11 LRU:

- Desarrollo de una aplicación informática para la monitorización remota (financiado por INPORCASA) – (2000-2001).
- Software de control y comunicaciones en tiempo real del equipo de inspección de contornos de conjuntos combustibles según especificación ESP-EQ-EC001 (financiado por ENUSA Industrias Avanzadas S.A) – (2001-2002).
- Software de monitorización y supervisión de salas de climatización (financiado por INPORCASA) – (2001-2002).

Por otro lado, con la llegada del candidato a este Departamento comienza la colaboración con el Unidad de Investigación del Instituto Universitario de Ciencias de la Educación –IUCE – (Grupo Canalejas) de la Universidad de Salamanca. Como resultado de esta colaboración el candidato abre otra línea de investigación que tiene como descriptor genérico *Informática educativa*.

Dentro de esta línea de trabajo el candidato ha participado en tres proyectos de investigación financiados por la Junta de Castilla y León:

- Guía Multimedia del Lenguaje Java (1998-1999).
- Componentes pedagógicos de educación superior en un espacio virtual (1999).

- Elaboración de un software educativo para la enseñanza de la ortografía (2001-2002).

Y, además, el candidato ha sido el investigador principal en otros cuatro proyectos de investigación financiados por la Junta de Castilla y León:

- Herramienta de Autor para el Desarrollo de Material Didáctico Multimedia (2001) [García, 2002].
- Docencia práctica en los laboratorios de las ingenierías en informática apoyada en herramientas CASE (2001) [García, 2001].
- Plataforma de trabajo cooperativo entre profesores de primaria y secundaria para la creación de recursos educativos en Red: Actividades de pensamiento complejo y creación de hipertextos multimedia (2002-2004).
- Desarrollo de una plataforma CASE basada en componentes para la docencia de Ingeniería del Software (2002-2003).

Uno de los puntos clave de la informática educativa lo forman los sistemas web, que se relacionan con la Ingeniería del Software a través de los aspectos de Interacción Persona-Ordenador [Lores, 2001] y de lo que ha dado en llamarse ingeniería web o *web engineering* [Ginige y Murugesan, 2001]. Gracias a la experiencia en el desarrollo de estos sistemas software, el candidato ha iniciado recientemente otra nueva línea de investigación que se puede resumir bajo el epígrafe de *Comercio Electrónico e Ingeniería Web*.

En relación con esta última línea de investigación, y debido a su reciente inicio, no se tienen resultados en forma de proyectos de investigación, aunque se han dirigido diversos proyectos fin de carrera y, actualmente, se están dirigiendo nuevos proyectos fin de carrera, trabajos de doctorado, trabajos de grado y una tesis doctoral.

Además, relacionado con esta última línea de investigación se realizará una propuesta de proyecto de investigación dentro de esta memoria, lo que plasma el compromiso del candidato con esta línea de investigación, que aparecerá como tal dentro de la Memoria de Actividades del Departamento de Informática y Automática correspondiente al año 2002. También cabe destacar la estrecha vinculación de los sistemas de comercio electrónico con otra de las líneas de investigación de este Departamento, *Sistemas de Agentes y Multiagentes*.

Como resumen de la trayectoria investigadora del candidato, se puede decir que ha estado marcada por la interacción con los diferentes grupos con los que se ha tenido relación (y aún se mantiene esta relación con todos ellos), distinguiéndose claramente tres líneas de investigación:

- *Reutilización sistemática del software.*
- *Comercio Electrónico e Ingeniería Web.*
- *Informática educativa.*

De estas tres líneas de investigación se van a presentar propuestas concretas de proyectos de investigación para las dos primeras.

6.4 Propuesta 1: e-CoUSAL – Una plataforma intermediaria de comercio electrónico para PYMES

El rápido crecimiento de Internet está estimulando un número creciente de negocios que tienen como campo de desarrollo la propia Red, entrando a formar parte de un sector en continuo auge, lleno de beneficios potenciales [Malone et al., 1987; Slonim y Bennet, 1996], aunque tampoco exento de problemas y polémicas, como es el del comercio electrónico.

Una de las principales características del comercio electrónico es que las empresas que deseen participar de esta modalidad de negocio deben entrar en una dinámica tecnológica alta para aumentar así su competitividad.

La incorporación de la empresa al comercio virtual supone una decisión estratégica no exenta de costes considerables y de riesgos que, en muchos casos, se erigen como una barrera insalvable impidiendo que la empresa pueda integrarse en un entorno que ayudaría a mejorar su capacidad competitiva de forma importante.

Cuanto más pequeño es el tamaño de la empresa más sufre estas barreras, obteniéndose así la ausencia generalizada de las PYMES en el comercio electrónico.

Con esta propuesta se pretende definir e implementar un modelo de comercio electrónico en el que se mezclan las características B2B (*Business to Business*) y las características B2C (*Business to Client*) de forma que la pequeña empresa tenga un fácil acceso al comercio electrónico.

Para conseguirlo se introducen dos conceptos clave en el modelo, una herramienta de autor y un sitio web intermediario entre el cliente final y las empresas que pretender vender.

La herramienta posibilita a la empresa crear y mantener sus contenidos, reduciendo así los costes de producción de contenidos, típicamente delegados a una tercera empresa, lo que era una fuente de problemas además de una fuente de gastos, especialmente si los contenidos representaban información muy volátil. Además, gracias a esta herramienta la PYME se

convierte en un elemento activo dentro de este modelo de negocio virtual, lo que enriquece más su participación y estrategia en su aventura digital.

El sitio web intermediario tiene dos objetivos fundamentales: almacenar y gestionar los catálogos de las empresas que desean vender sus productos en la Red, a la vez que sirve de escaparate de los productos para los clientes. Además, se puede convertir en una especie de centro comercial virtual especializado donde empresas de un mismo sector compitan y ofrezcan los productos a los clientes interesados, que utilizarán la interfaz del mediador como portal para llegar a los suministradores.

En la Figura 6.2 se presenta el esquema general del modelo que se persigue conseguir, en el que se presentan los principales agentes involucrados y sus relaciones.

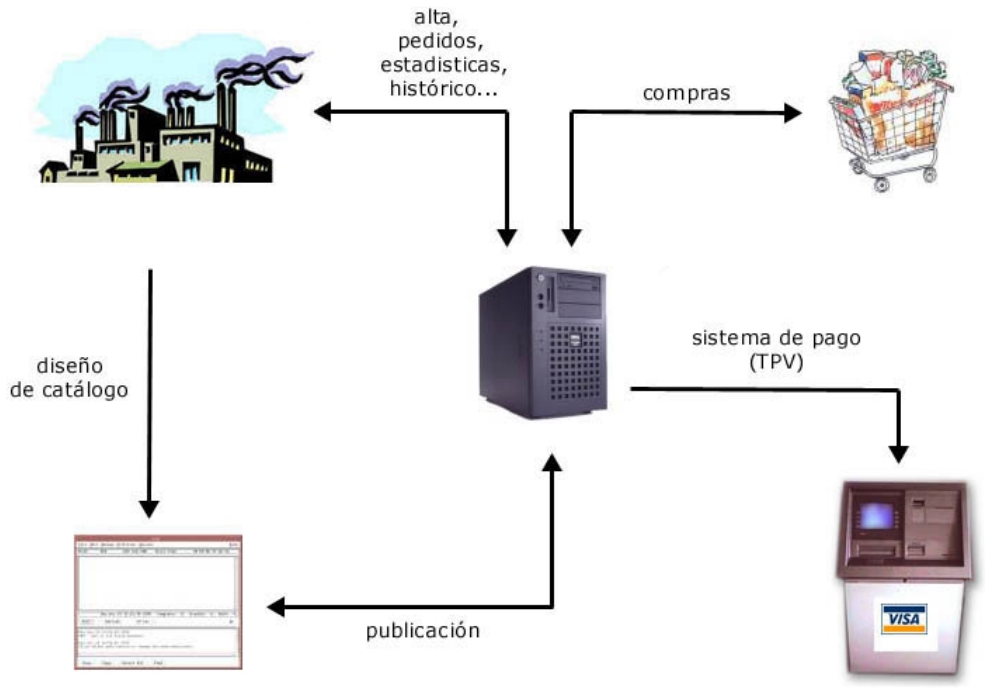


Figura 6.2. Esquema general del modelo

6.4.1 Estado del arte

En esta sección se realizan algunos comentarios sobre la situación actual en relación con el proyecto de investigación propuesto, trabajos afines y mención a la bibliografía más relevante sobre el tema.

6.4.1.1 El comercio electrónico

El comercio electrónico es, básicamente, hacer negocios en línea. En su forma más obvia supone vender productos en línea a los consumidores, pero en realidad, engloba cualquier tipo de negocio dirigido de forma electrónica. El comercio electrónico puede definirse, sencillamente, como la creación, dirección y extensión de las relaciones comerciales en línea [Kienan, 2000], o como el uso de medios y tecnologías electrónicas para dirigir el comercio [Whinston et al., 1997]. En el *OMG CommerceNet Whitepaper* se define comercio electrónico como la aplicación de las tecnologías de la información, específicamente inteligencia artificial y redes, a los problemas del comercio [McConnell, 1997].

En un sentido más amplio, también se puede definir el comercio electrónico como cualquier forma de transacción o intercambio de información comercial basada en la transmisión de datos sobre redes de comunicación como Internet [Vázquez, 1999]. En este sentido, el concepto no sólo incluye la compra y venta electrónica de bienes, información o servicios, sino también el uso de la Red para actividades anteriores o posteriores a la venta, como son la publicidad, la búsqueda de información sobre productos, proveedores..., la negociación entre comprador y vendedor sobre precio, condiciones de entrega..., la atención al cliente antes y después de la venta, la cumplimentación de trámites administrativos relacionados con la actividad comercial, la colaboración entre empresas con negocios comunes (a largo plazo o sólo de forma coyuntural). Estas actividades no tienen necesariamente que estar presentes en todos los escenarios de comercio electrónico.

Los mercados son lugares (ya sean físicos o figurativos) donde los bienes y los servicios [Casati y Shan, 2001] se intercambian entre compradores y vendedores [Fourie, 1991]. El intercambio de bienes y servicios entre un comprador y un vendedor se denomina una transacción [Williamson, 1985]. Independientemente de la naturaleza de los bienes y de los tipos de mercados involucrados, las transacciones pueden separarse en tres pasos básicos: *fase de información*, *fase de negociación* y *fase de ejecución* [Buxmann y Gebauer, 1998].

Sin embargo, los nuevos mercados que tienen lugar en Internet son diferentes de los mercados físicos tradicionales [Froehlich et al., 1999], por lo tanto se han de buscar nuevas estrategias y lógicas para los negocios [Benjamin y Wigand, 1995; Rayport y Sviokla, 1995].

Los mercados electrónicos en Internet están teniendo un mayor impacto en las funciones de mercado [Bailey y Bakos, 1997]. En la oferta de productos se incrementa la personalización de los productos ofrecidos y la agregación y disgregación de la información sobre los componentes de los productos [Bakos, 1998]. Aparece el concepto de producto digital (artículos, informes, imágenes, vídeos, música...) que puede distribuirse por la Red sin un coste añadido. Surgen

políticas de precios que permiten negociar el precio final en función de la demanda o aprovecharse de precios debidos a situaciones especiales. La logística se mejora al existir más información compartida entre cliente y proveedor, pudiéndose en el caso de los bienes digitales eliminarse por completo.

Participantes en el comercio electrónico

En el comercio electrónico participan como actores principales las empresas, los consumidores y las administraciones públicas. Así, normalmente se distinguen tres tipos básicos de comercio electrónico:

- Entre empresas o B2B (*Business to Business*) [Neches, 1996; Harting, 2000].
- Entre empresa y consumidor o B2C (*Business to Consumers*) [Neches, 1996].
- Entre empresa y administración o B2A (*Business to Administrations*) [Vázquez, 1999].

Las empresas pueden participar como usuarias (compradoras o vendedoras) y como proveedoras de herramientas o servicios de soporte para el comercio electrónico: proveedores de servicios de certificación de claves públicas, instituciones financieras... Por su parte, las administraciones públicas, actúan como agentes reguladores y promotores del comercio electrónico y como usuarias del mismo (por ejemplo en los procedimientos de contratación pública o de compras por la administración).

En un sentido amplio, los consumidores participarían en dos formas adicionales de comercio electrónico además del B2C: por una parte, el comercio electrónico directo entre consumidores (venta directa entre particulares) y, por otra, las transacciones económicas entre ciudadano y administración (pago de prestaciones sociales, pago de impuestos...).

El hecho de que el comercio electrónico esté ampliamente ligado a Internet se justifica porque, si bien las actividades de comercio electrónico entre empresas, por ejemplo mediante los sistemas de Intercambio Electrónico de Documentos EDI (*Electronic Data Interchange*), existen desde hace más de una década y son anteriores al uso comercial de Internet, ha sido esta apertura al uso comercial de Internet y, en particular, el desarrollo de la web, el elemento clave que ha hecho posible al comercio electrónico llegar al consumidor final y, en definitiva, ha provocado el actual crecimiento explosivo del comercio electrónico en todas sus formas.

Catálogos en línea

Una de las claves para el éxito de un buen sitio de comercio electrónico es ofrecer un entorno que facilite al usuario navegar por los productos ofrecidos, y finalmente realizar una compra [Textor, 1999].

La forma más típica de organizar los contenidos y productos ofrecidos en mediante un catálogo de productos en línea (e-catálogo), a través del cual los clientes finales interaccionan con la información de la empresa proveedora [Baron et al., 2000]. Un catálogo en línea puede definirse como la representación electrónica de la información sobre los productos y/o servicios de una organización [Segev et al., 1995].

En el marco del proyecto que aquí se presenta interesa el concepto de catálogo de múltiples vendedores (*multi-vendor electronic catalog*). Su misión es integrar información de los vendedores realizando una mezcla de datos y una reconciliación semántica. El catálogo de productos supone un punto de acceso centralizado para los productos y servicios que pueden ser comprados electrónicamente. Esta centralización puede ser física o virtual. En [Ginsburg et al., 1999] se distinguen tres modelos de catálogos electrónicos: modelo “hágalo usted mismo”, el modelo “tercera parte integradora” y el modelo “descubrimiento de conocimiento en tiempo real”. Estos modelos se diferencian en la configuración elegida para obtener y mantener la información de los productos a vender.

Mediadores en el comercio electrónico

Una PYME que quiera hacerse un hueco en el voraz mundo del comercio en Internet lo tiene muy difícil porque normalmente es una desconocida lejos de su entorno habitual de mercado, y carece de los medios técnicos y publicitarios para darse a conocer. Sin embargo, cuando varias de estas empresas se unen pueden formar una masa crítica que las haga atractivas para el cliente final. La forma más efectiva de darse a conocer en Internet es mediante un mediador o *broker* para comercio electrónico [Bichler et al., 1998; MarketScience, 2000], que puede servir como escaparate para los productos de estas empresas [Marathe y Diwakar, 2001].

El negocio de estos mediadores es unir a compradores y vendedores en un mercado electrónico, a la vez que ofrece servicios nuevos e innovadores [Bakos, 1991; Bailey y Bakos, 1997; Segev et al., 1999; Tsvetovat et al., 2000; Yamamoto y Sycara, 2001], o como competencia a intermediaros ya existentes [Chircu y Kauffman, 1999a; Chircu y Kauffman, 1999b].

Bailey y Bakos [Bailey y Bakos, 1997] enfatizan la necesidad de intermediarios en los mercados electrónicos. Basándose en un análisis de trece firmas B2B y B2C, ellos argumentan

que los mercados electrónicos necesitan agruparse en portales únicos de compra, proveedores de confianza, facilitadores de intercambio de información y filtros de información. El efecto de la comunicación electrónica descrita en [Malone et al., 1987] reduce el coste de las comunicaciones, pero el mismo efecto incrementa la cantidad de información que los proveedores deben manejar en un mercado electrónico. Intermediarios de filtrado pueden reducir esta sobrecarga de información [Bailey y Bakos, 1997].

La aparición de nuevos intermediarios electrónicos queda reflejada en otros trabajos de investigación [Sarkar et al., 1995]. Bakos también apunta que la mediación en los mercados electrónicos refleja la necesidad que tienen estos mercados de facilitar las transacciones e integrar elementos de seguridad. Estos roles de mediación se necesitan en diferentes situaciones que expanden la definición tradicional de transacción mercantil [Maes et al., 1999] e incluyen no sólo ventas, sino también compras de utilidades y transporte [Leebaert, 1998]. Hagel y Singer defienden la existencia de nuevos roles en los mercados electrónicos para agentes encargados de recopilar información [Hagel y Singer, 1999]. Estos mediadores de confianza, denominados infomediarios, pueden actuar en lugar de los compradores en su interacción con los vendedores, logrando un gran poder de compra al consolidar la demanda.

A. M. Chircu y R. J. Kauffman distinguen tres tipos de intermediarios [Chircu y Kauffman, 2000]: *intermediarios tradicionales* – firmas que ofrecen servicios para compradores y proveedores en un mercado tradicional y establecido; *intermediarios sólo de comercio electrónico* – comenzaron su negocio en el entorno de Internet, y sus clientes siempre acceden a ellos a través de la Red; y *intermediarios con potencial de comercio electrónico* – conducen su negocio utilizando tanto medios tradicionales como en línea.

Cuando los mediadores hacen su presencia se necesitan mecanismos y estándares para la interoperabilidad en el comercio electrónico. En [Bichler, et al., 1998] se distinguen tres tipos de soluciones:

- Interfaces estándares para la creación de componentes en el área del comercio electrónico. Basadas en tecnología CORBA (*Common Object Request Broker Architecture*) [OMG, 2001]. En concreto el *Electronic Commerce Domain Task Force* – ECDTF (http://ecdtf.omg.org/) de OMG está trabajando para estandarizar las interfaces de CORBA para componentes de comercio electrónico.
- Estándares de interoperabilidad centrados en documentos. Mucha de la información intercambiada en el B2B se hacía a través de mensajes EDI (*Electronic Data Interchange*). Pero los estándares EDI (ANSI X.12 - http://www.x12.org/ - ISO [ISO, 1996] o UN/EDIFACT) son complicados y caros

porque la mayoría de los mensajes viajan por redes propietarias y, además, no se integran bien con la infraestructura ubicua propia de la web. Una solución fue la utilización de Internet-EDI o mensajes XML/EDI (http://www.xmledi.com), porque es más sencillo validar y traducir estos mensajes a los formatos que se necesitan en los extremos de la comunicación [Laplante, 1998]. XML [Bray et al., 2000] está emergiendo como estándar para muchos lenguajes basados en la web y el comercio electrónico es un claro ejemplo de ello [Hernández y García, 2001]. Además, XML se utiliza profusamente como base para *frameworks* y protocolos de comunicación en sistemas de comercio electrónico [Glushko et al., 1999; Dogac y Cingil, 2001; García et al., 2001a; García et al., 2001b], así como para dotar de contenido semántico y ontológico a la web en general y a las plataformas de comercio electrónico en particular [Smith y Poulter, 1999; Omelayenko, 2001].

- Soporte de múltiples estándares. Un mediador genérico deberá entender varios estándares, combinando las interfaces y los documentos [Fingar, 2000].

6.4.1.2 Comercio electrónico y agentes

La tecnología basada en agentes tiene en el área del comercio electrónico uno de los campos de aplicación más importantes, donde los agentes no están limitados sólo a actividades de recolección de información, sino que de manera creciente se dedican a procesos cada vez más complejos para dar soporte a las compras y ventas propias del comercio electrónico. Para ello, se utilizan sistemas multiagentes y agentes adaptativos que pueden adaptar autónomamente algunas de sus propiedades, como por ejemplo su comportamiento [Gil et al., 2001].

Introducción al concepto de agente

Un agente es un sistema informático que está situado en un entorno, y que es capaz de realizar acciones autónomas en dicho entorno para conseguir sus objetivos de diseño [Sen y Weiss, 1999]. Generalmente, un agente se compone de cuatro elementos básicos: un componente sensor, un componente motor, una base de conocimiento o de información y un motor de razonamiento. Los componentes sensor y el motor permiten que el agente interaccione con su entorno (por ejemplo, llevando a cabo una acción o intercambiando datos con otros agentes). La base de conocimiento contiene la información que el agente tiene de su entorno. El motor de razonamiento permite a un agente realizar procesos de inferencia, planificación y aprendizaje (por ejemplo, deducir nueva información, generar secuencias de comportamiento e incrementar la eficiencia de la interacción con su entorno) [Weiß y Dillenbourg, 1999].

Un agente inteligente [Wooldridge y Jennings, 1995; Lang, 1995] es un agente que es capaz de realizar acciones flexibles de una forma autónoma para conseguir sus objetivos de diseño, donde la flexibilidad significa: autonomía, reactividad, pro-actividad y habilidad social [Wooldridge y Jennings; Weiss, 2001]. Una definición de agente inteligente puede ser: "Programas que actúan en lugar de sus usuarios humanos para realizar laboriosas tareas de recolección de información" [Sycara et al., 1996].

Otras propiedades adicionales, tales como la racionalidad, la coherencia, la capacidad de adaptación, o la movilidad, se utilizan para caracterizar a un agente [García, 2000a].

La racionalidad es una característica propia del ser humano. Un agente se considera racional cuando tiene unos conocimientos de su entorno, unos objetivos deseables y unas reglas que determinan cómo alcanzar los objetivos a partir del conocimiento que se tiene del medio. Esta característica de racionalidad le permite a un agente tomar decisiones sin intervención humana. Se está modelando la racionalidad propia del hombre, eso sí, de momento en problemas muy simples.

La coherencia implica que el conocimiento que un agente tiene de su mundo se almacena en una base de conocimiento interno al propio agente. Todo este conocimiento debe guardar un alto grado de coherencia para que el comportamiento del mismo sea el esperado.

La adaptación se ajusta a la forma de considerar la organización de un agente en su interacción con su entorno o con otros agentes, esto es, un agente adaptativo es aquél que es capaz de controlar sus propiedades (de comunicación, de comportamiento...) en consecuencia del agente con el que está interaccionado y del contexto en el que está inmerso. En otras palabras, el agente adaptativo debe tener la capacidad de modificar las propiedades de sus actividades para satisfacer sus necesidades, tanto internas como externas [Guessoum et al., 2001].

La movilidad es una característica opcional que pueden poseer los agentes. Un agente móvil es aquél que se puede mover físicamente por los nodos de una red para poder llevar a cabo sus tareas. El objetivo de la movilidad puede ser una mejor distribución de la carga de procesamiento, una mejor compartición de recursos, una lógica distribuida...

Los sistemas multiagente [Genesereth y Ketchpel, 1994] son sistemas complejos con respecto a su estructura y su funcionalidad. El diseño de una estrategia de interacción efectiva requiere un conocimiento previo de cualquier agente que pueda estar involucrado en el sistema. El concepto de sistema multiagente amplía la noción de agente inteligente en al menos dos maneras. En primer lugar, el usuario de un agente, quien imparte objetivos y delega tareas,

puede ser un humano u otro agente. Y en segundo lugar, un agente debe estar diseñado con mecanismos explícitos para comunicarse con otros agentes [Sycara et al., 1996].

Agentes en el comercio electrónico

Internet es un nicho de negocio al que sólo se le puede sacar todo su potencial estableciendo nuevos modelos de negocio centrados en la Red, escalables y efectivos en cuanto al coste. Se necesitan nuevas cadenas de suministro, canales de distribución y mercados dinámicos que utilicen procesos de computación distribuidos e inteligentes, los agentes [Ma, 1999].

La tecnología de agentes ha demostrado su potencial en las aplicaciones de comercio electrónico en múltiples ocasiones [Chavez y Maes, 1996; Doorenbos et al., 1997]. A continuación se repasan algunos trabajos que, bien directa o indirectamente, están relacionados con la tecnología de agentes en el comercio electrónico.

P. Maes y B. Sheth han trabajado en agentes de filtrado de información que se utilizan en los aspectos de personalización [Sheth y Maes, 1993; Maes, 1994; Sheth, 1994].

Armstrong y Durfee [Armstrong y Durfee, 1998] investigan sobre el uso de los agentes para comprar y vender información en los mercados digitales, como por ejemplo el caso de las bibliotecas digitales. Se centran en analizar los factores que influyen la eficacia de los modelos de agentes para el aprendizaje en situaciones intrínsecamente impuestas. Presentan un modelo que ofrece mecanismos por los que los agentes adaptan dinámicamente las estrategias que usan para comprobar cómo el cambio dinámico de estrategias afecta a la cooperación.

Foisel, Chevrier y Haton [Foisel et al., 1998] presentan un modelo basado en la noción de interacción para construir organizaciones adaptativas y diversos experimentos sobre el problema de la distribución de múltiples materias primas. La organización del modelo se representa por dos aspectos, uno estático para soportar la estructura y otro dinámico para permitir que la estructura evolucione. La organización se construye sobre un modelo de interacción, compuesto de tres elementos: una motivación u objetivo, los agentes involucrados en la interacción y la forma en que tiene lugar los intercambios.

Vidal y Durfee [Vidal y Durfee, 1998] describen un *framework* que puede ser utilizado para modelar y predecir el comportamiento de los sistemas multiagentes con agentes que aprenden. Su modelo tiene ciertos parámetros que capturan la dinámica de una amplia variedad de algoritmos de aprendizaje, por lo tanto capturan las habilidades de aprendizaje de los agentes (tales como su porcentaje de cambio, su porcentaje de aprendizaje y su porcentaje de retención). Validan el algoritmo utilizando agentes de refuerzo de aprendizaje en un sistema de mercado electrónico.

Carmel y Markovitch [Carmel y Markovitch, 1998] han desarrollado una aproximación de aprendizaje que se aproxima al modelo del oponente. Dado un conjunto de movimientos del oponente forman configuraciones específicas de un tablero. En primer lugar presentan un algoritmo para calcular la profundidad de la búsqueda que está siendo usada por el oponente. Si la función asumida es exacta, entones se necesitan pocos ejemplos para inducir la profundidad de la búsqueda. También presentan un algoritmo que aprende de la estrategia de juego del oponente, y adapta la estrategia durante el juego. Las suposiciones realizadas son las siguientes: la función de evaluación del oponente es una combinación lineal de las características conocidas del tablero, y el oponente no cambia su función mientras dura el juego. Experimentalmente demuestran la efectividad de su aproximación de aprendizaje para diferentes estrategias del oponente.

Matos, Sierra y Jennings [Matos et al., 1998] estiman que para tener éxito en entornos multiagentes y abiertos, los agentes autónomos deben ser capaces de adaptar sus estrategias de negociación y sus tácticas a sus circunstancias predominantes. Estos autores presentan un modelo preparado para un proceso de sistema de gestión para un negocio real. Este modelo tiene que operar en un amplio rango de entornos con un alto número de parámetros, por lo tanto estos autores estudian cómo evolucionan las estrategias de negociación de los agentes. La técnica que adoptan para la evolución de las estrategias son los algoritmos genéticos, haciendo evaluaciones en diferentes entornos. Han demostrado la utilidad de los agentes empleando un conjunto de tácticas para diferentes problemas de negociación.

Weiß y Dillenbourg [Weiß y Dillenbourg, 1999] definen un sistema multiagente adaptativo de acuerdo a los sistemas adaptativos. Distinguen tres mecanismos en los sistemas adaptativos multiagentes: el *mecanismo de multiplicación* – cada agente tiene una evolución diferente, por tanto, se han de utilizar diferentes métodos, se realizan diferentes elecciones, y la eficiencia del sistema es mejorada; el *mecanismo de división* – las tareas pueden ser descompuestas en diferentes partes, cada parte se corresponde con un grupo de agentes, y el aprendizaje es sencillo y rápido; y el *mecanismo de interacción* – un agente puede compartir información con otros agentes para mejorar el rendimiento del sistema completo.

Guessoum y otros [Guessoum y Briot, 1999; Guessoum et al., 2001] proponen un modelo genérico de agente adaptativo, donde cada agente tiene un comportamiento y un meta-comportamiento que ofrecen al agente dos tipos de adaptación (estructural y de comportamiento). Ellos validan el modelo con una simulación de un modelo económico.

Corchado [Corchado, 2001] presenta una solución distribuida para los negocios en Red, donde los agentes se incorporan para incrementar la adaptación del sistema y su evolución con

respecto a los cambios del entorno en una compañía de construcción como soporte a las ventas. La adaptación del sistema confía en una pareja de agentes deliberativos caracterizados por una arquitectura BDI (*Believe, Desire and Intention*). Esta arquitectura en conjunción con un sistema de CBR (*Case-Based Reasoning*) configura la adaptación de la solución.

Matos y Sierra en [Matos y Sierra, 1999] dirigen su investigación hacia modelos de negociación. Estudian el papel que la negociación juega entre los agentes de comercio. Presentan dos tipos de arquitectura de agentes: basada en casos y *fuzzy* para modelar una estrategia de negociación basada en agentes. La arquitectura basada en casos determina la combinación de tácticas y de los valores de los parámetros a utilizar en cada momento de la negociación. Una ruta alternativa es la utilización de un conjunto de reglas *fuzzy*, de forma que la adaptación es modelada por un conjunto de reglas de adaptación *fuzzy*. Finalmente, proponen una aproximación de evolución aplicando un algoritmo genético sobre una población de agentes para determinar la estrategia de negociación más exitosa.

Ardissono y Goy [Ardissono y Goy, 2000] desarrollan SETA (*Servizi Telematici Adattativi*), un prototipo para soportar las ventas en las tiendas web, centrándose en la personalización de la interacción con los usuarios. El sistema SETA se basa en una arquitectura multiagente reactiva, desarrollada con **Objectspace Voyager**.

Maes y otros en [Maes et al., 1999] presentan una visión general de la tecnología de agentes en el comercio electrónico, revelando como los agentes se utilizan para manejar el proceso de compra y venta en la web.

M. P. Papazoglou [Papazoglou, 2001] distingue diferentes tipos de agentes en un sistema multiagente de comercio electrónico: agentes de aplicación, agentes personales o de interfaz, agentes para actividades de negocio generales, agentes de localización de información, agentes de negociación, agentes de soporte de nivel de sistema, agentes de planificación, agentes de interoperación, agentes de transacciones de negocio y agentes de seguridad.

M. Weiss [Weiss, 2001] define un lenguaje de patrones para el desarrollo de arquitecturas de comercio electrónico basadas en agentes. Identifica las fuerzas que se necesitan para el diseño de sistemas de comercio electrónico basados en agentes: autonomía, necesidad de interaccionar, sobrecarga de información, múltiples interfaces, aseguramiento de la calidad, adaptabilidad, aspectos de privacidad, búsqueda de precios e identidad.

Los agentes móviles también tienen aplicación en el comercio electrónico, por ejemplo SOMA (*Secure and Open Mobile Agent*) [Corradi et al., 1999], que es un *framework* para construir aplicaciones de comercio electrónico seguras. Otras referencias de agentes móviles en

comercio electrónico son [Papaioannou y Edwards, 1998; Sohn y Yoo, 1998; Vigna, 1998; Yee, 1999; Wong et al., 1999].

Los protocolos y políticas de comunicación en un sistema multiagente para el comercio electrónico es otro aspecto a tener en cuenta. Existen diversas propuestas [Finin et al., 1995; Pitt y Mamdani, 1999; Pitt et al., 2000] basadas en KQML (*Knowledge Query and Manipulation Languague*) [Finin y Wiederhold, 1991] y/o en FIPA (*Foundation for Intelligent Physical Agents*) [FIPA, 2001; García, 2000a].

6.4.1.3 Adaptabilidad en el comercio electrónico

Cuando un sistema hipermedia es utilizado por diferentes usuarios, con diferentes intereses y/o niveles de conocimiento, un aspecto que toma una especial relevancia es la capacidad de adaptación que presenta dicho sistema.

Un entorno de comercio electrónico es un sistema en el que no se debe tratar a todos los usuarios por igual, ofreciéndoles el mismo conjunto de páginas estáticas a todos ellos. En un sistema B2C primarán los aspectos de adaptación relacionados con la interfaz, la navegación, los contenidos y las preferencias del usuario, buscando crear un ambiente amigable donde el cliente se sienta a gusto y reconocido por el sistema. En un entorno B2B o en un contexto de intermediación, en el que concurren aspectos híbridos B2B y B2C, a la adaptación de los parámetros anteriormente comentados habría que añadir los aspectos de adaptación del comportamiento, por ejemplo para establecer políticas de negocio de las empresas proveedoras o para fijar los precios en un entorno cambiante o en el que la negociación estuviera permitida.

F. Paternò y C. Mancini distinguen entre sistemas adaptables y sistemas adaptativos. Los sistemas adaptables son sistemas que permiten únicamente modificar algunos de sus parámetros, y así adaptar su comportamiento en consecuencia. Si el sistema se adapta al usuario de forma automática, es cuando se denomina adaptativo [Paternò y Mancini, 1999].

La hipermedia adaptativa representa una dimensión de trabajo e investigación que intersecciona con la hipermedia y el modelado de usuario. P. Brusilovsky [Brusilovsky, 1996; Brusilovsky, 2001] identifica seis tipos de sistemas hipermedia adaptativos: sistemas educativos, sistemas de información en línea, sistemas de ayuda en línea, sistemas de recuperación de la información, sistemas institucionales y sistemas de gestión de vistas personalizadas en sistemas de información.

Los sistemas de información en línea no forman un grupo homogéneo, y por tanto se dividen en subgrupos, distinguiéndose así [Brusilovsky, 2001]: *enciclopedias electrónicas*, como por ejemplo PEBA-II [Milosavljevic, 1997; Hirashima et al., 1998]; *quioscos de*

información, como por ejemplo AVANTI [Fink et al., 1998]; *museos virtuales*, como por ejemplo Marble Museum [Paternò y Mancini, 1999] o SAGRES [Bertoletti y da Rocha, 1999]; *guías*, como por ejemplo HYPERAUDIO [Not et al., 1998]; *sistemas de comercio electrónico*, como por ejemplo SETA [Ardissono y Goy, 1999] o TELLIM [Jörding, 1999a; Jörding, 1999b; Milosavljevic y Oberlander, 1998]; y *sistemas de soporte al funcionamiento*, como por ejemplo ADAPTS [Brusilovsky y Cooper, 1999] o MMA [Francisco-Revilla y Shipman III, 2000].

Los sistemas de comercio electrónico divergen de los sistemas de información clásicos, porque en este tipo de sistemas la navegación no es la principal actividad, ya que ésta es comprar algún bien, servicio o realizar algún trabajo. De hecho, cuanto mejor trabaja el sistema, menos navegación se requiere [Brusilovsky, 2001].

Un ejemplo digno de mención dentro del caso de los sistemas de comercio electrónico adaptativos es TELLIM (*inTELLIgent Multimedia*) [Jörding, 1999a; Jörding, 1999b; Milosavljevic y Oberlander, 1998]. Es un sistema que genera documentos individuales en tiempo de ejecución de forma que cada cliente consigue la presentación que le satisface y cumple sus intereses. El motor de adaptación se centra en el proceso de adquisición de conocimiento a través del sistema que monitoriza la interacción en el lado del cliente que alimenta un algoritmo incremental que aprende las preferencias del cliente y genera la presentación multimedia en tiempo de ejecución.

Los agentes adaptativos, ya comentados anteriormente, se están utilizando con profusión en el área de hipermedia adaptativa en sistemas de comercio electrónico, además de SETA [Ardissono y Goy, 1999], se pueden citar otros trabajos en este sentido [Pazzani y Billsus, 1997; Menczer y Belew, 1998; Perkowitz y Etzioni, 1998; Pazzani y Billsus, 1999; Ardissono et al., 1999].

6.4.2 Objetivos del proyecto

El objetivo general de este proyecto de investigación es la definición, diseño e implementación de una plataforma de comercio electrónico que conecte diferentes empresas suministradoras, PYMES, con los clientes potenciales de sus productos, a través de Internet. Permitiendo a las PYMES, con un desembolso mínimo y su participación activa en la creación y gestión de sus catálogos de productos, entrar en el mundo del comercio electrónico con unos riesgos mínimos.

Como se va a comprobar a través de los objetivos concretos que se plantean en este mismo apartado, este proyecto de investigación tiene un carácter aplicado que puede enmarcarse dentro de las Tecnologías de la Información. En cuanto a su interés socio-económico, tanto para la

Comunidad Autónoma como para el resto del país, hay que destacar su aplicabilidad a los diferentes tipos de PYMES que puedan ofrecer sus productos y/o servicios a través de Internet, presentando éstos en un catálogo. Esta característica tiene un indudable peso específico en la realidad socio-económica de Castilla y León en particular, y de España en general, al incrementar el mercado potencial de cualquier PYME española.

A continuación se van a enumerar los objetivos concretos que se persiguen en este proyecto. Éstos se concretan en diferentes soluciones en pro de cumplir el objetivo global planteado:

1. Establecer las dimensiones B2B y B2C del modelo de comercio electrónico.

En el campo de los intermediarios para el comercio en Internet se distinguen varios tipos, dependiendo del modelo de negocio que se quiera soportar. Los intermediarios más típicos son los que se centran en la dimensión B2B, sin embargo por las características del modelo que se quiere definir e implementar en este proyecto de investigación, se persigue un modelo mixto o híbrido B2B y B2C.

La dimensión B2B surge de la existencia de relaciones entre un conjunto de empresas proveedoras de productos (PYMES) y otra empresa (el intermediario) que les ofrece los servicios necesarios para que sus productos estén al alcance de los clientes finales.

La dimensión B2C ocurre entre el intermediario y los clientes finales, que ven al intermediario como un escaparate de productos (típicamente relacionados), a los que acceden según la metáfora de funcionamiento del sitio de comercio intermediario.

Así, el objetivo último es **definir los límites y características propias del modelo híbrido B2B/B2C con el que se quiere contar.**

2. Definir las características del producto a compartir y publicar: el catálogo de productos.

Al existir un sitio de comercio electrónico intermediario encargado de gestionar los productos de diferentes empresas proveedoras, es de vital importancia definir cuál va a ser el elemento de información que se comparta y publique en el sitio de comercio electrónico, y que sirva de base semántica para que el cliente final tenga noción de qué compra y tenga posibilidad funcional para poder llegar a encontrarlo, seleccionarlo y, finalmente, comprarlo.

La base elegida es la del catálogo de productos, que se transforma en el sitio de comercio electrónico en un catálogo de productos multi-proveedor; donde el catálogo cumple el modelo "hágalo usted mismo" [Ginsburg et al., 1999], porque es el mediador el que tiene la iniciativa de definir el tipo de catálogo que va a manejar, así como su semántica.

Así, el objetivo último es **definir la semántica de un catálogo de productos multi-proveedor, cuyas restricciones y semántica son establecidas por el intermediario, utilizando para ello XML**.

3. Desarrollar una herramienta de autor para la creación, gestión y publicación de catálogos de productos.

Una de las claves para el éxito del modelo de comercio electrónico que se quiere definir, es que la PYME tenga el control absoluto en la creación del catálogo de productos que quiere publicar en el intermediario.

Para conseguir esto se recurre a la creación de una herramienta de autor para crear, publicar y mantener el catálogo de productos de la PYME en el sitio de comercio electrónico intermediario.

Esta herramienta debe estar especialmente diseñada para facilitar las tareas relacionadas con la creación y mantenimiento del catálogo de productos, teniendo en cuenta que quien va a manipular dicha herramienta no tiene por que tener conocimientos avanzados en informática, sino sólo conocimientos básicos en aplicaciones ofimáticas e Internet. Por ello, la interfaz de esta herramienta debe ser sencilla y amigable, construida sobre la base de un conjunto de metáforas bien conocidas por el usuario y que permanezcan constantes a lo largo de las diferentes partes de la interfaz.

Con el objeto de simplificar y guiar al creador del catálogo se debe utilizar el concepto de vista de trabajo, donde cada una de las vistas contempladas pretende mostrar la información desde un determinado enfoque. Las principales vistas que deben soportarse a través de esta herramienta son: *la vista de definición de plantillas*, *la vista de definición de productos*, *la vista de definición de políticas de negocio* y *la vista de organización de catálogos*. Todas ellas han de agruparse en una vista más general, denominada *vista de repositorio*, donde se muestra toda la información distribuida por el resto de vistas [García et al., 2001b; Hernández et al., 2001].

Además, se debe perseguir que la interfaz de la herramienta de autor soporte características de internacionalización, de forma que soporte un conjunto ilimitado de idiomas diferentes.

Así, el fin último es **implementar una herramienta de autor para que la PYME realice todas las labores de creación, publicación y mantenimiento de los catálogos de productos**.

4. Desarrollar un sitio web intermediario entre las empresas proveedoras y los clientes finales que aloje y sirva de escaparate para los catálogos de productos de las PYMES, y además gestione las peticiones de los clientes.

El sitio web intermediario debe ofrecer a los clientes todos y cada uno de los servicios para navegar a través del catálogo de productos multi-proveedor y realizar sus compras [Adam et al., 1996; Borrego et al., 2001].

Además de los servicios propios de un sitio web comercial, el intermediario debe definir una ontología para el almacenamiento de los productos, así mismo debe permitir la definición de diferentes políticas de negocio para las empresas proveedoras.

La interfaz del sitio web debe cumplir una serie de restricciones muy importantes. Por una parte debe ser amigable y fácil de utilizar a través de las metáforas que el usuario conoce y reconoce debido a su experiencia en la visita a otros sitios de comercio electrónico. Por otro lado, debe existir un soporte multilingüe y de diferentes monedas. Además, se precisa que la interfaz sea adaptativa para los clientes, incorporando un soporte para la gestión de los perfiles de los usuarios, que posibilite atenderlos de una forma más personalizada.

Así, el objetivo último es **desarrollar un sitio web intermediario adaptativo que sirva de escaparate a los productos de las PYMES**.

5. Definir un protocolo de publicación de catálogos entre la herramienta y el sitio web.

Si las PYMES están representadas en la plataforma de comercio electrónico a construir mediante una herramienta de autor para la creación de catálogos, que deben publicar automáticamente en el sitio web intermediario, se necesita un protocolo para facilitar esta comunicación.

Este protocolo de comunicación debe basarse en XML, y permitir tanto la transferencia del catálogo desde la herramienta al sitio web intermediario, como la detección y recuperación de errores en dicho proceso.

Así, el fin último es **definir un protocolo de comunicación entre herramienta y sitio web basado en XML**.

6.4.3 Metodología y plan de trabajo

Para la consecución de los objetivos planteados en el presente proyecto, se ha dividido el mismo en una serie de tareas agrupadas por afinidades y asignadas a los miembros del equipo de investigación (formado por cinco investigadores). La coordinación interna estará a cargo del responsable del proyecto y se llevará a cabo mediante la realización de una sesión quincenal de control de resultados.

Está prevista la incorporación de dos Becarios de Investigación que realicen labores de implementación de las herramientas necesarias al mismo tiempo que se inician en la investigación y realizan sus Tesis Doctorales.

A continuación se describe, por objetivos y tareas, la planificación del proyecto con detalle de resultados esperados y plazos de ejecución.

Modulo 1. Definición del modelo de negocio

Duración

6 meses (desde año 1, mes 1 hasta año 1, mes 9)

Descripción

En este bloque de tareas se pretende definir el modelo de negocio que va a regir a todos los actores involucrados en la plataforma de comercio electrónico que se pretende construir

Resultados esperados

- Definición del dominio de trabajo
- Caracterización de todos los actores involucrados
- Definición de un modelo de negocio mixto B2B/B2C
- Establecimiento de las diferencias y elementos comunes del modelo propuesto con otros modelos existentes
- Definición de la semántica del catálogo de productos

Plan de trabajo (descomposición en tareas)

TAREA: Estudio del papel de los intermediarios en el comercio electrónico

CÓDIGO: T1.1

COMIENZO: Año 1, mes 1

FINAL: Año 1, mes 4

RESPONSABLE: Investigador 1

ENTRADAS: Ninguna

DESCRIPCIÓN: Se repasará la bibliografía especializada en intermediarios, se localizarán ejemplos en la red, y se identificarán los tipos, servicios de éstos

PARTICIPANTES: Investigador 2, Investigador 3

TAREA: Realización del modelo de dominio

CÓDIGO: T1.2

COMIENZO: Año 1, mes 2

FINAL: Año 1, mes 9

RESPONSABLE: Investigador Principal

ENTRADAS: Informes de la tarea T1.1

DESCRIPCIÓN: Se realizará un modelo del dominio de negocio a implementar en la plataforma a desarrollar. Será el documento que establezca los requisitos y restricciones arquitectónicas guíen el resto del proyecto

PARTICIPANTES: Investigador 1, Investigador 2, Investigador 3

TAREA: Definición semántica y estructural del catálogo de productos

CÓDIGO: T1.3

COMIENZO: Año 1, mes 5

FINAL: Año 1, mes 9

RESPONSABLE: Investigador 4

ENTRADAS: Ninguna

DESCRIPCIÓN: Se definirá la semántica que propia del catálogo de productos multi-proveedor

PARTICIPANTES: Becario 1

Modulo 2. Definición e implementación de la herramienta de autor para creación de catálogos

Duración

12 meses (desde año 1, mes 10 hasta año 2, mes 9)

Descripción

En este bloque de tareas se pretende definir e implementar la herramienta de autor, que es la representación de las PYMES en la arquitectura propuesta

Resultados esperados

- Definición de un *workflow* para la creación de catálogos de productos
- Desarrollo de la herramienta de autor para la creación y publicación de catálogos

Plan de trabajo (descomposición en tareas)

TAREA: Definición del *workflow* para la creación de un catálogo de productos

CÓDIGO: T2.1

COMIENZO: Año 1, mes 10

FINAL: Año 1, mes 11

RESPONSABLE: Investigador Principal

ENTRADAS: Ninguna

DESCRIPCIÓN: Se definirá cuál es el camino más adecuado y sencillo para la creación de un catálogo de productos, y éste será el que se implemente en la herramienta. Este *workflow* se traducirá en las vistas de trabajo soportadas por la herramienta

PARTICIPANTES: Investigador 2

TAREA: Modelado de la interacción persona-ordenador en la herramienta

CÓDIGO: T2.2

COMIENZO: Año 1, mes 10

FINAL: Año 2, mes 3

RESPONSABLE: Investigador 3

ENTRADAS: Definición del catálogo

DESCRIPCIÓN: Los aspectos de interacción persona-ordenador son fundamentales para lograr el éxito del modelo de negocio que se pretende definir, así se deben estudiar todos los aspectos de navegación, usabilidad, internacionalización... propios de la herramienta de autor

PARTICIPANTES: Investigador 2, Investigador principal, Becario 1

TAREA: Componentes de negocio en la herramienta de autor

CÓDIGO: T2.3

COMIENZO: Año 1, mes 10

FINAL: Año 2, mes 4

RESPONSABLE: Investigador 4

ENTRADAS: Ninguna

DESCRIPCIÓN: La definición de un catálogo de productos puede verse completada con la definición de políticas de negocio para vender dichos productos. Se necesita por tanto una vista de trabajo que permita establecer estos aspectos y luego comunicárselos al servidor de comercio electrónico intermediario

PARTICIPANTES: Investigador principal, Becario 2

TAREA: Desarrollo de la herramienta de autor

CÓDIGO: T2.4

COMIENZO: Año 1, mes 12

FINAL: Año 2, mes 9

RESPONSABLE: Investigador 1

ENTRADAS: Resultados de las tareas T2.1, T2.2, T2.3 y T1.3

DESCRIPCIÓN: Se desarrollará y probará de forma iterativa e incremental la herramienta de autor

PARTICIPANTES: Todo el equipo.

Modulo 3. Definición e implementación de un sitio de comercio intermediario entre PYMES y clientes finales

Duración

24 meses (desde año 2, mes 1 hasta año 3, mes 12)

Descripción

En este bloque de tareas se pretende definir e implementar el sitio de comercio electrónico intermediario entre las PYMES y los usuarios finales

Resultados esperados

- Definición de una arquitectura de agentes cooperativos y adaptativos para el comercio electrónico
- Desarrollo de un sistema de perfiles para usuarios finales
- Desarrollo de un sitio web intermediario entre PYMES y usuarios finales
- Definición de un protocolo de comunicación entre herramienta de autor y sitio de comercio electrónico

Plan de trabajo (descomposición en tareas)

TAREA: Definición de un sistema multiagente adaptativo para comercio electrónico

CÓDIGO: T3.1

COMIENZO: Año 2, mes 1

FINAL: Año 2, mes 5

RESPONSABLE: Investigador Principal

ENTRADAS: Ninguna

DESCRIPCIÓN: Se definirá una arquitectura basada en agentes inteligentes adaptativos sobre la que se implementará el sitio de comercio electrónico

PARTICIPANTES: Investigador 2, Becario 2

TAREA: Modelado de la interacción persona-ordenador en el sitio de comercio electrónico

CÓDIGO: T3.2

COMIENZO: Año 2, mes 1

FINAL: Año 2, mes 7

RESPONSABLE: Investigador 3

ENTRADAS: Definición del catálogo

DESCRIPCIÓN: Los aspectos de interacción persona-ordenador son fundamentales para lograr el éxito del modelo de negocio que se pretende definir, así se debe estudiar todos los aspectos de navegación, usabilidad, internacionalización, adaptatividad... propios del sitio de comercio electrónico. Cabe destacar la necesidad de definir un sistema de perfiles para adaptar la interfaz a los gustos y necesidades de los usuarios finales

PARTICIPANTES: Investigador 2, Investigador 4, Investigador principal, Becario 1

TAREA: Definición del protocolo de comunicación entre herramienta de autor y el sitio intermediario de comercio electrónico

CÓDIGO: T3.3

COMIENZO: Año 2, mes 10

FINAL: Año 3, mes 1

RESPONSABLE: Investigador 1

ENTRADAS: Definición del catálogo

DESCRIPCIÓN: En esta tarea se abordarán los aspectos de comunicación y coordinación entra la herramienta de autor y el sitio de comercio electrónico, utilizando XML como base para ello

PARTICIPANTES: Investigador 2, Investigador 4, Becario 1, Becario 2

TAREA: Desarrollo del sitio de comercio electrónico intermediario

CÓDIGO: T3.4

COMIENZO: Año 2, mes 4

FINAL: Año 3, mes 12

RESPONSABLE: Investigador 2

ENTRADAS: Resultados de las tareas T3.1, T3.2, T3.3 y T1.3.

DESCRIPCIÓN: Se desarrollará y probará de forma iterativa e incremental el sitio web de comercio electrónico

PARTICIPANTES: Todo el equipo

Modulo 4. Transferencia tecnológica

Duración

12 meses (desde año 3, mes 1 hasta año 3, mes 12)

Descripción

Se realizará la transferencia tecnológica de los resultados de los otros módulos a medida que la madurez de las tecnologías desarrolladas lo permita, con aplicación a situaciones reales

Resultados esperados

- Implantación del modelo de negocio definido en el proyecto

Plan de trabajo (descomposición en tareas)

TAREA: Transferencia tecnológica

CÓDIGO: T4.1

COMIENZO: Año 3, mes 1

FINAL: Año 3, mes 12

RESPONSABLE: Investigador Principal

ENTRADAS: Productos implementados, modelos definidos

DESCRIPCIÓN: Se implantará el modelo básico de negocio en empresas colaboradoras

PARTICIPANTES: Todo el equipo

En la Tabla 6.1 se resume el plan de trabajo del presente proyecto de investigación.

Módulo	Tareas	Investigadores	Primer año	Segundo año	Tercer año
1 Definición del modelo de negocio	T1.1 Estudio del papel de los intermediarios en el comercio electrónico	I1, I2, I3			
	T1.2 Realización del modelo de dominio	IP, I1, I2, I3			
	T1.3 Definición semántica y estructural del catálogo de productos	I4, B1			
2 Definición herramienta de autor para la creación de catálogos	T2.1 Definición del *workflow* para la creación de un catálogo de productos	IP, I2			
	T2.2 Modelado de la interacción persona-ordenador en la herramienta	I3, I2, IP, B1			
	T2.3 Componentes de negocio en la herramienta de autor	I4, IP, B2			
	T2.4 Desarrollo de la herramienta de autor	I1, I2, I3, I4, IP, B1, B2			
3 Definición sitio de comercio electrónico intemediario	T3.1 Definición de un sistema multiagente adaptativo para comercio electrónico	IP, I2, B2			
	T3.2 Modelado de la interacción persona-ordenador en el sitio de comercio electrónico	I3, I2, I4, IP, B1			
	T3.3 Definición del protocolo de comunicación entre herramienta y el sitio intermediario	I1, I2, I4, B1, B2			
	T3.4 Desarrollo del sitio de comercio electrónico intermediario	I1, I2, I3, I4, IP, B1, B2			
4 Trasf. Tecnológica	T4.1 Transferencia tecnológica	I1, I2, I3, I4, IP, B1, B2			

Tabla 6.1. Planificación temporal del proyecto de investigación e-CoUSAL

6.5 Propuesta 2: Iniciación de una línea de productos sobre herramientas software para células CIM

En la industria manufacturera las células CIM (*Computer Integrated Manufacturing*) ocupan un papel fundamental para la realización automática de una tarea compleja que antes era realizada por los operadores humanos para la obtención de un determinado producto. Sin embargo, en la realidad, como señala [Olsson y Piani, 1992], no aparecen aplicaciones CIM genéricas y completas para la automatización global de una planta manufacturera. Esto se achaca a la falta de estructuración por cuanto, frente a otras industrias de procesos como la química, la de manufacturación no dispone aún de estándares para formular la estructura de su producción. En este caso existe una enorme diversificación y es difícil establecer criterios de comportamiento para el control integrado. Solamente existen soluciones limitadas para máquinas de control numérico, robots o células concretas de producción.

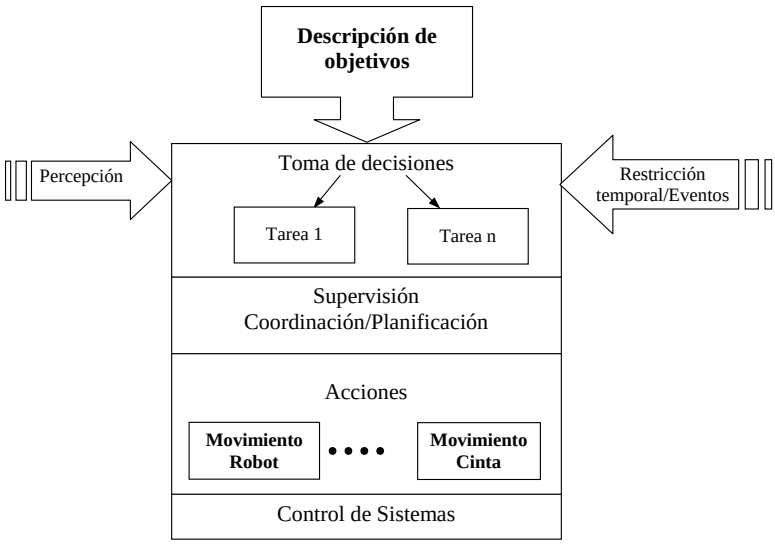


Figura 6.3. Estructura jerárquica de una célula CIM genérica

En cuanto a la consecución de un objetivo CIM global, los trabajos de investigación y desarrollo se centran en proponer diferentes soluciones tecnológicas que pretenden aportar diferentes capacidades de integración, coordinación, optimización, control supervisado, seguridad, reconfigurabilidad, comportamiento inteligente... Para obtener todas o algunas de estas capacidades se está trabajando actualmente en campos como planificación y coordinación de tareas, robótica, sistemas en tiempo real, control de sistemas... La mejora en las condiciones de operación requiere, por tanto, la incorporación de un grado creciente de inteligencia y un comportamiento flexible, encaminados, en definitiva, a lograr una actuación autónoma en la célula y un objetivo general de integración de sistemas. En la Figura 6.3 se muestra un modelo jerárquico donde se observa la estructura de las diferentes tareas a realizar en una célula CIM

genérica. No aparecen todos los niveles de decisión presentes en un modelo global CIM, pero su estructura es aplicable a cualquier caso concreto dentro de una planta de manufacturación.

De forma natural el funcionamiento en una célula CIM tiene un carácter distribuido y heterogéneo, debido a que cada componente tiene asociado un elemento de procesamiento (procesadores dedicados, ordenadores, autómatas programables...). En la práctica, esta característica queda oscurecida dado que el controlador es un autómata industrial que adolece de rigidez. Tradicionalmente, en el entorno industrial, los autómatas programables se han ocupado del control y la sincronización de las secuencias de operaciones de cada elemento individual para realizar el ciclo regular de la célula. Su construcción robusta y su sencillez en la programación han potenciado su utilización durante varias décadas y son en la actualidad la alternativa más empleada. A pesar de las ventajas que presentan los autómatas programables, dada su limitación en el procesamiento de datos, es necesario el uso de computadores en ciertas situaciones donde se necesita una gran versatilidad. De hecho, ambos sistemas pueden aparecer juntos y cooperar mutuamente [Groover et al., 1989].

A la hora de aportar comportamientos más flexibles e inteligentes es necesario disponer de un controlador software que, de forma distribuida, gestione los componentes de la célula, permitiendo contar con nuevas prestaciones. Este controlador software se encargará de la coordinación de todos los elementos de la célula y, por tanto, se requiere de mecanismos de comunicación potentes que aseguren unos requisitos temporales así como de fiabilidad.

La existencia de diferentes tareas computacionales hace asimismo surgir la necesidad de utilizar políticas de diseño de software que permitan su implementación de la forma más independiente posible de las plataformas, así como una integración de diferentes paradigmas de desarrollo, pero cumpliendo unos elevados requisitos de robustez y comportamiento en tiempo real. Adicionalmente, cabe constatar el papel cada vez más importante que tiene el mantenimiento y evolución de las aplicaciones que soportan el funcionamiento de las plantas, que tiene sus implicaciones asimismo en las políticas de diseño de software.

Siempre que la flexibilidad, la facilidad de evolución y la independencia de los medios físicos sean premisas que aparecen en la definición de un determinado sistema de información, la labor de diseño se hace mucho más compleja.

Si a todo lo anterior se añade la importancia de poder adaptar el sistema de información a diferentes contextos con el menor esfuerzo posible, parece apropiado invertir en el esfuerzo de definición de una línea de productos que recoja la arquitectura básica del sistema de información.

Una línea de productos consiste en una arquitectura software básica, un conjunto de elementos reutilizables y un conjunto de productos derivados de la utilización de dichos elementos reutilizables [Bass et al., 1999].

Sin embargo, las líneas de productos son en sí mismas elementos sumamente complejos que requieren un esfuerzo ingente de trabajo tanto en el plano técnico – definición arquitectónica [Jacobson et al., 1997; Bosch, 2000], desarrollo, uso e instanciación – como en el plano organizativo – dimensión de negocio y organización [Bass et al., 2000] – que aleja a muchas organizaciones de esta forma de reutilización, porque no pueden permitirse ni el esfuerzo ni la inversión necesarios para iniciar una línea de productos.

En este proyecto de investigación se pretende iniciar una línea de productos en el área de las células CIM, que permita el desarrollo con reutilización de aplicaciones CIM para la industria manufacturera.

En la iniciación de esta línea de productos se cuenta con dos pilares básicos: la experiencia del Grupo de Robótica del Departamento de Informática y Automática de la Universidad de Salamanca, y un modelo estructural de elemento reutilizable de grano grueso, denominado Mecano [García, 2000b]

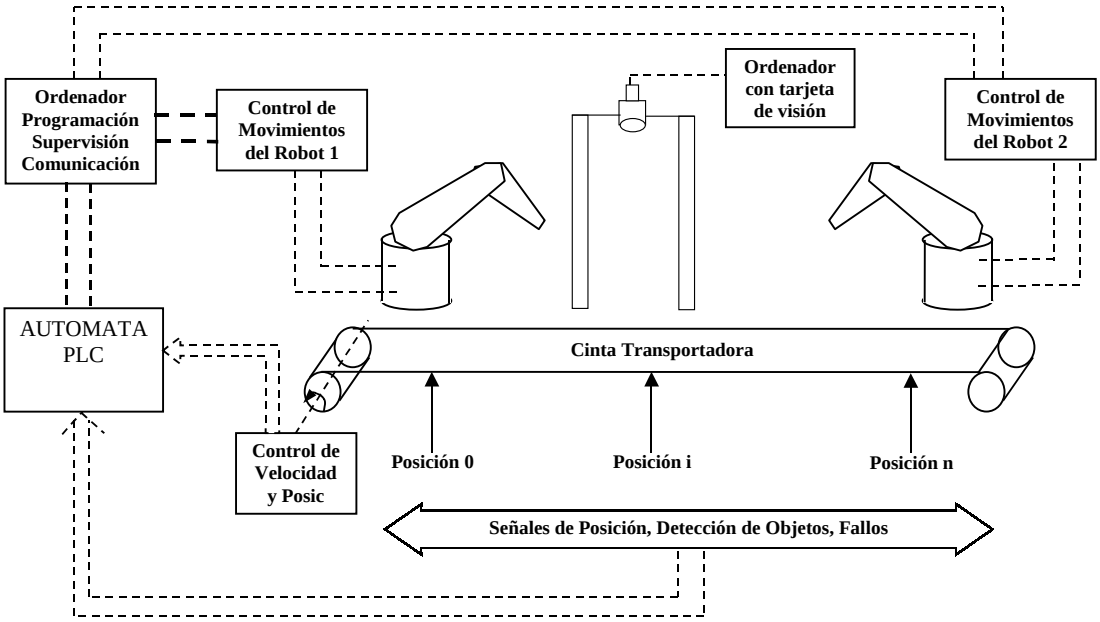


Figura 6.4. Elementos de la célula CIM piloto

El diseño del software base que nutrirá inicialmente la línea de productos se hará para una planta tipo disponible para el Grupo de Robótica. En esta planta tipo existen diversos elementos, como son dos robots tipo PUMA, una cinta transportadora, un autómata programable, una carta

de ejes, varios ordenadores personales, un sistema de adquisición de imágenes con una cámara entre otros, como se puede apreciar en la Figura 6.4.

6.5.1 Estado del arte

En esta sección se realizan algunos comentarios sobre la situación actual en relación con el proyecto de investigación propuesto, trabajos relacionados y mención a la bibliografía más relevante sobre el tema.

Aunque el objetivo del proyecto está en la iniciación de una línea de productos, primeramente, y de forma muy sucinta, se revisa el estado del arte de las materias relacionadas en la realización de este proyecto de investigación.

Así, se comienza con la planificación de tareas que se encuentra en los niveles más altos de la jerarquía y que es necesaria para dotar de flexibilidad a una célula genérica CIM. De forma más particular, y dado que los robots son los elementos que aportan un mayor grado de flexibilidad e inteligencia, se analiza las técnicas involucradas en conseguir un comportamiento inteligente, centrándose en la aplicación de procedimientos de generación automática de trayectorias. Teniendo en cuenta que se deben mantener los requerimientos de fiabilidad necesarios en las aplicaciones industriales, se repasan las bases más importantes del campo de los sistemas en tiempo real, así como los correspondientes a los sistemas de comunicaciones avanzados. Esto se debe a que, de forma natural, en una célula aparecen los sistemas distribuidos heterogéneos. Así, aunque se tengan resueltos los problemas de comunicación, la programación distribuida juega un papel de gran trascendencia atendiendo a las ventajas potenciales que pueden originar los estándares basados en la programación orientada a componentes. Finalmente, se introduce la reutilización sistemática del software, para dar paso a la aproximación de reutilización que aquí interesa, las líneas de productos.

6.5.1.1 Planificación de tareas

En un entorno CIM formado por diferentes subsistemas es fundamental decidir las operaciones que deben ejecutarse y su secuencia de ejecución. Para ello se necesitan unas capacidades de razonamiento y planificación más altas que las que aparecen en el resto de los niveles de la jerarquía. Es una línea de investigación en la que aún faltan resultados generales, pero la Inteligencia Artificial ha producido una gran variedad de métodos que permiten incorporar habilidades de razonamiento para planificar tareas de alto nivel. Estos métodos están disponibles para construir planificadores cuyas salidas son secuencias de operaciones que pueden consistir en la definición de configuraciones objetivo para los robots. Sin embargo, los

planificadores generados con técnicas de Inteligencia Artificial representan tareas utilizando construcciones basadas en la lógica de predicados que permite incorporar poca información sobre aspectos como la geometría o relaciones temporales.

Así, es necesario realizar una labor de modelado de sistemas complejos donde se pueda incorporar toda la información, que con el anterior planteamiento presentaba dificultades. Es en este punto donde las redes de Petri [Peterson, 1977; Peterson, 1981; Silva, 1985] pueden proporcionar una solución a este problema. En este sentido, dada la creciente complejidad de los automatismos y computadores que aparecen como subsistemas que interactúan (sistemas concurrentes), se plantea la utilización de las redes como herramienta de modelado y, así, poder realizar el análisis que puede permitir la validación y optimización de los resultados de la planificación. Aportan una valiosísima capacidad de representación gráfica y soportan una potente teoría de la validación. Algunas variantes de las Redes de Petri, como son las denominadas Redes de Petri Coloreadas [Jensen, 1997a; Jensen, 1997b; Jensen, 1997c] son especialmente adecuadas para el modelado de sistemas distribuidos complejos y adicionalmente pueden ser utilizadas para realizar simulaciones del comportamiento del sistema, que permitirán, a su vez, la validación del diseño del mismo.

6.5.1.2 Robótica

La utilización de robots en los entornos industriales tiene ya una larga tradición, fundamentalmente, en la industria manufacturera, en la que el modo de funcionamiento básico se realiza con movimientos previamente programados. Sin embargo, el uso de robots en una célula de fabricación flexible utilizando este tipo de funcionamiento limita en gran medida el funcionamiento de la planta, puesto que las posibles tareas a realizar están seriamente restringidas. En este sentido, desde los comienzos de la disciplina se plantea la utilización de técnicas del campo de la Inteligencia Artificial, en busca de la capacidad de generación automática de trayectorias, de manera que un robot pueda moverse de forma autónoma por un entorno sin que se produzcan colisiones.

En la Figura 6.5 se muestra una visión detallada de cómo un sistema robótico incorpora el comportamiento inteligente necesario. A partir de la definición de operaciones a realizar especificada por el planificador de tareas, el subsistema robótico debe proporcionar la capacidad de generación automática de movimientos (*path planning*). Como puede observar, la tarea central es la generación de caminos geométricos que representan el camino. Sin embargo, para que sea posible una planificación más sencilla, sin necesidad de tener en cuenta la descripción geométrica del robot, es conveniente hacer uso del espacio de las configuraciones (C-espacio)

[Lozano-Pérez, 1983], concepto procedente de la mecánica y que permite considerar al robot como un simple punto en este espacio.

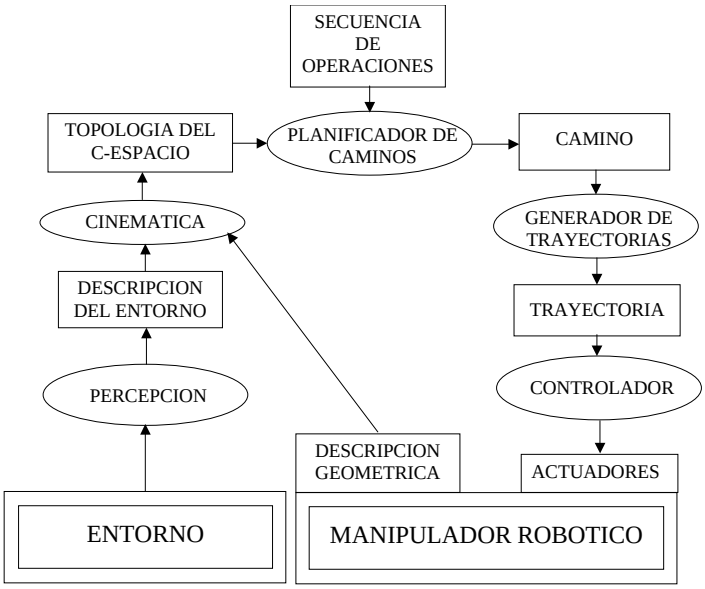


Figura 6.5. Jerarquía de tareas en un sistema robótico inteligente

Como la representación de los obstáculos en el C-espacio requiere de múltiples cálculos cinemáticos, es una tarea con una gran carga computacional. En este sentido, existen diferentes trabajos enfocados en su reducción, entre los que destacan aquéllos en los que se presenta el concepto de convolución entre los obstáculos y el robot [Kavraki 1995; Curto, 1998] utilizando la transformada rápida de Fourier (*Fast Fourier Transform – FFT*) que proporciona una gran optimización. Además, es necesario destacar que en [Curto, 1998] se propone un formalismo general aplicable tanto a robots móviles y articulados.

Por tanto, aunque la obtención del conjunto de configuraciones libres de colisión para los robots articulados es una labor considerablemente complicada, su interés radica en dos aspectos fundamentales como son:

- Los obstáculos se pueden ver como limitaciones en los movimientos del robot. Éstos a su vez se representan como restricciones en las variables de control pues éstas se identifican con los componentes de una configuración.
- Los procedimientos de planificación de trayectorias se ven simplificados porque los procedimientos de generación automática de caminos libres de colisiones utilizan el espacio de las configuraciones como el ambiente en el que se desarrolla el problema de la búsqueda.

En lo que respecta a la tarea principal de generación de caminos, existen un gran número de técnicas utilizadas para este fin [Latombe, 1991]. Estas técnicas abarcan diferentes puntos de vista muy diferenciados:

- Inteligencia Artificial clásica: entendiendo el problema de la búsqueda del camino como un problema de búsqueda de solución, donde cada configuración representa un estado del problema.
- Potenciales: donde se considera una serie de fuerzas ficticias que repelen al robot de los obstáculos y lo acercan hacia su configuración objetivo.
- Computación Evolutiva: también existen trabajos más recientes que utilizan los conceptos nuevos de computación evolutiva, donde destacan en gran medida los algoritmos genéticos dado que se puede considerar como un problema de optimización.

6.5.1.3 Procesamiento paralelo

Dada la complejidad computacional de muchos de los cálculos que se introducen a la hora de resolver algunos de los problemas previamente expuestos, y dadas las limitaciones de tiempo real intrínsecas de la aplicación descrita, en muchos casos la única forma posible de abordar con éxito la resolución del problema es la aplicación de las técnicas de procesamiento paralelo.

De los múltiples modelos de programación paralela [Foster, 1995] que conviven en la actualidad, hay uno que es el más extendido y que mejor encaja en un entorno de procesamiento distribuido: el paso de mensajes. En la actualidad, existen herramientas que permiten desarrollar aplicaciones paralelas utilizando diferentes técnicas para implementar el paso de mensajes en la comunicación entre procesos de computación en una red heterogénea de estaciones de trabajo. Entre ellas cabe destacar MPI (*Message Passing Interface*). Las dos grandes ventajas que presenta la utilización de MPI son: estandarización (MPI se está convirtiendo en el estándar “de facto” para la programación paralela y distribuida con paso de mensajes) y la portabilidad (puede ser ejecutado prácticamente en cualquier plataforma, desde ordenadores personales a supercomputadores, pasando por estaciones de trabajo). Estas dos características posibilitan que el desarrollo de las aplicaciones que se lleven a cabo, no impliquen su utilización restringida a un único y concreto entorno industrial o de investigación, sino que pueda ser fácilmente utilizado en otros entornos.

Algunos de los problemas computacionales susceptibles de ser resueltos mediante las técnicas de procesamiento paralelo mencionadas son: planificación de trayectorias [Henrich, 1997] y cálculos de dinámica y optimización de cálculos de transformadas de Fourier [Tong y

Swartztrauber, 1991]. Debe remarcarse que algunos de estos problemas computacionales ya han sido abordados por nuestro grupo de investigación obteniéndose resultados alentadores [Therón et al., 1999].

6.5.1.4 Sistemas en tiempo real

En una célula CIM los requisitos de tiempo real se presentan en los diferentes niveles que aparecen en la jerarquía de la Figura 6.3. De forma más inmediata, en el nivel inferior la ejecución de los diferentes lazos de control, utilizando microcontroladores, tiene unas fuertes restricciones temporales; en el caso de los robots el control de posición y velocidad de las articulaciones. También en los niveles superiores, la ejecución de las diferentes acciones (trayectorias del robot, movimientos de la cinta...), la coordinación y la supervisión necesitan cumplir unos plazos temporales y reaccionar ante la aparición de determinados eventos. Aparecen por tanto las dos formas de ejecución en tiempo real: periódicas y no periódicas. Esto tiene su influencia en la propuesta y definición de las herramientas o los planificadores con los que llevar a cabo la implementación de las tareas de computación.

Para conseguir este objetivo resulta fundamental que el sistema operativo que se vaya a utilizar cumpla con las exigencias de los requisitos temporales de las citadas tareas. A diferencia de los sistemas operativos de propósito general, cuyo objetivo fundamental es mejorar el comportamiento en el caso medio, los sistemas operativos de tiempo real tienen como objetivo mejorar el comportamiento en el peor caso. Esto es, la corrección del sistema no sólo depende de los resultados lógicos de los algoritmos, sino que también va a depender del momento en el que éstos se producen. Por lo tanto, el sistema operativo de tiempo real [Burns y Wellings, 1997] debe administrar los recursos del ordenador de la forma más adecuada para que todas las tareas puedan cumplir con sus requisitos temporales.

Actualmente existen en el mercado diferentes alternativas que cumplen satisfactoriamente con los requisitos temporales impuestos por la característica de tiempo real. Tanto sistemas comerciales, **VxWorks** (Entorno Tornado, de **WindRiver Systems**), **QNX** (**QNX Software Systems**), como sistemas de libre distribución como **RT-Linux** o **RT-Mach**. La mayoría de éstos proporcionan una arquitectura especialmente diseñada para poder cumplir con las exigencias de las tareas de tiempo real, ofreciendo distintos tipos de planificación de tareas, lo que permite construir sistemas que se ajustan mejor a las necesidades de cada problema. Además, en algunos casos, ofrecen todo un conjunto de herramientas de gran ayuda en el desarrollo de aplicaciones, como son: compiladores cruzados (para diferentes arquitecturas hardware), depuradores, entornos gráficos, adaptación a estándares (extensiones de tiempo real POSIX 1003.1B, ANSI C, entorno de red TCP/IP, CORBA, Java, acceso a Internet). Esto

facilita la integración en sistemas heterogéneos..., de manera que en muchas ocasiones un factor relevante en la elección será, precisamente, la cantidad y la calidad de estas herramientas. Por ejemplo, **VxWorks** ofrece la posibilidad de integración con Java y CORBA (**VisiBroker**), pero de forma totalmente adaptada a las estrictas características de un sistema de tiempo real, de manera que los hilos de Java se corresponden exactamente con tareas de tiempo real, permitiendo así un estricto control de las mismas por parte del planificador.

6.5.1.5 Sistemas de comunicaciones y computación distribuida

Los requisitos que se originan con la incorporación de módulos computacionales hacen que sea totalmente necesaria la generación y el manejo de información, lo que a su vez exige vías de comunicación entre los diferentes dispositivos inteligentes que intervienen en el proceso. Por lo tanto, las comunicaciones constituyen un elemento fundamental en los nuevos entornos de fabricación, constituidos en general por una gran cantidad de dispositivos de control inteligentes, que deben trabajar de forma coordinada. Se ha de garantizar en todo momento la disponibilidad y la accesibilidad de la información.

Dentro de un sistema de comunicaciones existen diversos niveles que se utilizan dependiendo de los requisitos temporales. Así, para la conexión de ordenadores es factible utilizar redes de tipo *Ethernet*, que no garantizan las comunicaciones en tiempo real. Sin embargo, en otro nivel se encuentran los buses de campo que cubren de forma satisfactoria las necesidades temporales requeridas.

La utilización de un sistema basado en bus [Olsson y Piani, 1992] es muy común en aplicaciones de automatización, ya que en él se pueden conectar de forma fácil y eficiente sensores, actuadores, autómatas programables... Existen numerosas alternativas en este sector sin que ninguna parezca liderar completamente el mercado. Los más extendidos son: **Profibus**, **Fieldbus**, **Modbus** y **DeviceNet**. Cada uno de ellos presenta diferentes características en cuanto a tipo de cableado, velocidades, posibilidad de asignar prioridades, protocolos de comunicaciones...

En cuanto a las posibilidades de conexión remota, el mercado ha convertido a Internet en un estándar de facto. Sus protocolos de red se encuentran ampliamente extendidos y la gama de productos (tarjetas, *routers*, *switches*...) es muy extensa, así resulta muy sencillo la contratación de servicios y líneas de comunicación.

El rendimiento de los modernos sistemas de computación distribuida depende en gran medida de los servicios de red que se utilizan para mover información de un nodo a otro. Sin embargo, curiosamente, la evolución de estos servicios está siendo más lenta que la evolución

del resto de los elementos que integran el entorno en el que se construyen los sistemas de computación. Esto hace referencia tanto al hardware implicado (computadores, conmutadores, *routers*...) como al software utilizado (sistemas operativos, protocolos, lenguajes de programación...). Esta lenta evolución no es atribuible ni a la falta de necesidades ni a la ausencia de ideas innovadoras. El problema principal se encuentra en que el cambio o creación de nuevos protocolos de red es una tarea lenta y dificultosa.

Una línea de investigación que trata de paliar estos problemas son las redes activas [Tennenhouse et al., 1997; Tennenhouse y Wetherall, 1996], que proporcionan una interfaz usuario/red programable con la capacidad de soportar modificaciones dinámicas en el comportamiento de la misma. Las redes activas representan un paso fundamental en la evolución de las redes de conmutación de paquetes, desde los dispositivos tradicionales de encaminamiento de paquetes hacia una funcionalidad más general que soporta el control y la modificación dinámica del comportamiento de la red. Las redes activas permiten a las aplicaciones inyectar de forma dinámica programas en los nodos locales y en los del resto de la red de área ancha.

Las redes son *activas* [Tennenhouse et al., 1996] desde dos puntos de vista. Para los *routers* y *switches* de la red, se plantea dotarlos de nuevas funcionalidades para que puedan actuar sobre ella; por ejemplo, realizando cálculos sobre el flujo de datos de usuario que pasan a través de ellos. Desde el punto de vista de los usuarios, éstos pueden programar la red cargando sus propios programas para la realización de tareas específicas; por ejemplo, el usuario puede especificar a un *router* que ejecute un determinado algoritmo de compresión durante el procesado de sus paquetes. No obstante, la idea subyacente y fundamental en las redes activas es estandarizar un modelo de comunicación en lugar de protocolos de comunicación individuales. De esta forma se pretende lograr un consenso en la comunidad de investigación para estandarizar un modelo computacional en los nodos activos (conjunto de instrucciones y recursos disponibles) que proporcione una interfaz de programación de aplicaciones común.

La aplicación de tecnologías de redes activas en el caso concreto de una célula CIM puede permitir el desarrollo de nuevos protocolos específicos que contribuyan a un rendimiento más óptimo de todo el sistema de comunicaciones. Así, por ejemplo, la detección de fallos en las comunicaciones y la reacción a los mismos puede permitir mejorar las necesarias características de fiabilidad en las comunicaciones de una célula.

6.5.1.6 Programación de sistemas distribuidos

El sistema de información asociado a una célula CIM es un claro ejemplo de sistema distribuido, donde cada elemento constituyente tendrá asignada una parte de funcionalidad de dicho sistema de información, y de forma cooperativa interactuará con las demás partes para conseguir la funcionalidad general.

Todavía siendo grandes las ventajas ofrecidas por las aplicaciones distribuidas, éstas no están exentas de problemas, entre los que cabría mencionar su mayor dificultad de definición, desarrollo, entendimiento y mantenimiento. Para superar esta dificultad intrínseca a los sistemas distribuidos se justifica la aplicación de la tecnología de objetos en el desarrollo de los mismos.

Ante esta necesidad nacen diferentes propuestas encaminadas al establecimiento de un modelo estándar de componentes distribuidos basado en el paradigma objetual; siendo algunas de las más relevantes **CORBA** [Mowbray y Zahavi, 1995; Siegel, 1996; Aklecha, 1999; Henning y Vinoski, 1999; Hoque, 1999; Siegel, 2000; OMG, 2001], **DCOM** [Microsoft, 1996] o **RMI** [Orfali y Harkey, 1997; SUN, 2002].

De todas ellas CORBA (*Common Object Request Broker Architecture*) es la que mejor se ajusta a la problemática que presenta una célula CIM porque es un *framework* para objetos distribuidos que especifica los estándares necesarios para el intercambio de mensajes entre objetos residentes en entornos heterogéneos. Por entorno heterogéneo se está haciendo referencia a aquéllos en los que las plataformas que lo configuran aparecen máquinas de muy diversa procedencia (UNIX, Windows, Macintosh...), interconectados por un sistema de red y donde los objetos que colaboran pueden estar implementados en diferentes lenguajes. CORBA solventa toda la problemática de estos entornos definiendo un modelo objeto propio [OMG, 2001], que será independiente de los lenguajes de programación utilizados gracias a la clara separación entre la interfaz y la implementación de los objetos.

6.5.1.7 Reutilización sistemática del software

Para que la reutilización del software ofrezca los beneficios que de ella se esperan, se debe producir un acercamiento sistemático e institucional en su implantación [Griss, 1993; Griss, 1996b] huyendo de una aproximación oportunista, que no dejará más que algunos beneficios muy localizados que no repercutirán en la organización global.

La reutilización sistemática del software es una aproximación de carácter institucional para producir desarrollos en los que los *assets* son intencionadamente creados o adquiridos para ser reutilizables [Griss, 1993].

La primera propuesta de adopción de un proceso de reutilización sistemática del software para mejorar la productividad y la calidad en el desarrollo del software data del año 1968 [McIlroy, 1976]. Sin embargo, y a pesar de los numerosos estudios y trabajos llevados a cabo en este campo, no se han conseguido grandes avances en la adopción sistemática por parte de las organizaciones de la reutilización en el proceso de construcción de software [Biggerstaff, 1992; Mili et al., 1995]. Esta realidad, unida a la necesidad de las organizaciones de mejorar la productividad y la calidad de los cada vez más complejos desarrollos de software, han provocado que en los últimos tiempos la reutilización constituya una de las principales líneas de investigación y desarrollo dentro del campo de la Ingeniería del Software. Una confirmación de este hecho viene dada por los proyectos ESPRIT de la Unión Europea relacionados con la reutilización sistemática del software, entre los que cabe citar los siguientes: ITHACA (*Integrated Toolkit for Highly Advanced Computers Application*) [Ader et al., 1990] (ESPRIT projects 2121, 2705 y 6343); F^3 (*From Fuzzy to Formal*) [Castano y Antonellis, 1994; Antonellis y Pernici, 1995] (ESPRIT III project 6621); REBOOT (*REuse Based on Object-Oriented Techniques*) [Karlsson, 1995] (ESPRIT III 7808) y EUROWARE (*Enabling Users to Reuse Over Wide AREas*) [Sema, 1996] (ESPRIT 8947) proyectos englobados en el proyecto SER ESPRIT project 9809 [SER, 1996].

En el ámbito de la reutilización del software, hay que distinguir dos grandes líneas de investigación: el desarrollo para reutilización y el desarrollo con reutilización [Karlsson, 1995]. El desarrollo para reutilización aborda los problemas derivados de la preparación y calificación de elementos software, de cualquier nivel de abstracción, para que puedan ser reutilizados en proyectos diferentes a aquél en el que se generaron. El desarrollo con reutilización hace referencia a los elementos técnicos y metodológicos relacionados con la construcción de nuevos productos software mediante la utilización de elementos software existentes y reutilizables.

Elementos software reutilizables

La reutilización del software fue inicialmente concebida como la combinación de componentes de código almacenados en una biblioteca mediante herramientas automatizadas. Trabajos posteriores dieron lugar a la aparición de diferentes tecnologías, enfoques y extensiones del concepto de producto software reutilizable, existiendo diferentes clasificaciones de los mismos [Jones, 1984; Krueger, 1992; Mili et al., 1995]. En una primera aproximación, y de forma muy esquemática, todos estos desarrollos relacionados con la reutilización del software se pueden clasificar basándose en el objeto y en el método de reutilización usado.

Atendiendo al método de reutilización, se puede distinguir entre tecnologías de composición y tecnologías de generación. Las primeras tienen un enfoque de bloques de

construcción y las segundas abordan la reutilización desde un enfoque generador o procesador reutilizable. Desde el punto de vista del objeto reutilizable, en la actualidad se considera que todo el conocimiento y productos derivados de la producción del software, son susceptibles de ser reutilizados en la construcción de nuevos sistemas software [Freeman, 1987]. Surge así un nuevo concepto para reflejar el alcance actual de la reutilización, el ***asset***. Un *asset* se define como "*cualquier producto del ciclo de vida del software que pueda ser potencialmente reutilizado. Esto incluye: modelo de dominio, arquitectura de dominio, requisitos, diseño, código, bases de datos, esquemas de bases de datos, documentación, manuales de usuario, casos de prueba...*" [DoD, 1992].

Los beneficios de la reutilización de estos *assets* se ven aumentados si se reutilizan los *assets* subsiguientes derivados de estos *assets* de alto nivel de abstracción [Cybulski, 1996; Cybulski et al., 1997]. Contar con un elemento software reutilizable complejo, definido en diferentes niveles de abstracción simultáneamente, es un factor que potenciará la reutilización del software y en consecuencia los beneficios derivados de ésta. Surgen así los elementos software reutilizables de grano grueso, tales como los mecanos [García, 2000b], los *application frames* [Ader et al., 1990] o los *frameworks* [Wirfs-Brock y Johnson, 1990].

Repositorios para la reutilización del software

Uno de los factores claves del éxito o fracaso del proceso de construcción de software soportado por la reutilización, es la existencia de un entorno que permita una clasificación y recuperación de componentes rápida y efectiva. La función principal de este tipo de entornos es la de servir de enlace entre el desarrollo para la reutilización, donde se producen los componentes, y el desarrollo con reutilización, donde los componentes son reutilizados. Por lo tanto, el entorno debe estar formado no solamente por el repositorio de componentes reutilizables, sino que debe incorporar las estructuras, procesos y herramientas que automaticen total o parcialmente los procesos de reutilización.

En consonancia con esta idea, la mayoría de los repositorios desarrollados a partir de proyectos de investigación o de iniciativas comerciales, incorporan el soporte metodológico junto con sus correspondientes herramientas para el almacenamiento, clasificación y recuperación de *assets*. La utilización de la flexibilidad y el poder que ofrece la tecnología web puede aprovecharse para crear un entorno que dé soporte al desarrollo con reutilización y para la reutilización de forma distribuida a través de Internet [Trump, 1997].

Aunque inicialmente el concepto de repositorio se corresponde con una simple base de datos para el almacenamiento de *assets*, éste ha ido evolucionando hacia el concepto de

biblioteca de reutilización propugnado por el DoD (*Department of Defense*) de EEUU y los estándares de la OTAN para reutilización [NATO, 1992], donde la idea básica es que las bibliotecas de reutilización son modelos más aplicaciones o servicios [Wallnau, 1992].

Uno de los entornos más representativos de esta categoría de repositorios es **ASSET** (*Asset Source for Software Engineering Technology*). ASSET es una división de SAIC (*Science Applications International Corporation*), que ofrece una forma de comercio electrónico basado en web. Dentro de ASSET se encuentra la biblioteca WSRD (*Worldwide Software Resource Discovery*) que es una biblioteca de software libre y comercial, que está en continuo crecimiento, y que está centrada en artefactos del ciclo de vida del software, así como en documentos sobre desarrollo y reutilización.

El proyecto **EEC-ESPRIT II ITHACA** (*Integrated Toolkit for Highly Advanced Computers Applications*) [Ader et al., 1990] se llevó a cabo entre 1989 y 1992 con el objetivo de establecer un entorno de desarrollo software soportado en dos bases fundamentales: la orientación a objeto y la reutilización. El entorno almacena y publica para su reutilización los objetos semánticos, los objetos de diseño y los objetos de implementación dentro del SIB (*Software Information Base*) [Constantopoulos et al., 1992]. El SIB consiste en un conjunto de objetos que representan información del software en diferentes niveles de abstracción (requisitos, diseño y código) organizada en descripciones. La estructura de cada descripción depende del modelo utilizado en cada caso. Un modelo de descripción es un conjunto de metaclases que representan entidades, las relaciones entre dichas entidades y la semántica específica asociada a la forma en que sus instancias se interrelacionan. El SIB establece una red semántica con las descripciones software. Cada descripción es un nodo en la red; y los enlaces representan las dependencias, correspondencias, relaciones semánticas, transformaciones y enlaces hipermedia a documentación [Bellinzona et al., 1993].

Ostertag, Hendler, Prieto-Díaz y Braun [Ostertag et al., 1992] presentan una biblioteca llamada AIRS (*AI-based Reuse System*). Este sistema se basa en un enfoque híbrido, que incluye un enfoque de facetas [Prieto-Diaz, 1989] y de redes semánticas. El método de recuperación se basa en la computación de métricas de similaridad que permiten comparar tanto componentes como paquetes.

El proyecto **REBOOT** (*REuse Based on Object-Oriented Techniques*) [Karlsson, 1995; SER, 1996] es un proyecto europeo ESPRIT III #7808, desarrollado dentro del SER ESPRIT Project #9809. El principal objetivo de este proyecto es crear un marco metodológico y organizador para implantar la reutilización como un método habitual en las organizaciones en las que se desarrolla software. Además del enfoque metodológico, REBOOT aporta un modelo

de componente software reutilizable, que sirve como marco de referencia para la implementación práctica de algunos repositorios. Para hacer factible la reutilización, los *assets* deben estar almacenados en el repositorio junto con la información necesaria que permita su recuperación, la evaluación de su ajuste a los requisitos buscados, y facilite su adaptación e integración en el sistema global si fuera necesario. El modelo de componente software reutilizable tiene como objetivo describir la información que se necesita almacenar junto al *asset*. La clasificación de un *asset* es la información que se guarda junto con el *asset* y que sirve de ayuda para la identificación y recuperación del mismo. Es la información en que se basa el desarrollador con reutilización para buscar un *asset*. La relación entre la clasificación y el componente es de uno a varios. Esto significa que varios componentes pueden tener la misma clasificación.

El proyecto **EUROWARE** (*Enabling Users to Reuse Over Wide AREas*) [Sema, 1996; SER, 1996] es el proyecto ESPRIT #8947, desarrollado dentro del proyecto SER ESPRIT #9809. EUROWARE es un conjunto de aplicaciones, construidas sobre la base de REBOOT, e integradas en un servidor web que permite que clientes remotos, conectados a una red TCP/IP, tengan acceso a los *assets* almacenados en el repositorio para su reutilización.

El repositorio **RIB** (*Repository In a Box*) de NHSE (*National HPCC Software Exchange*) (http://www.nhse.org/RIB), es un conjunto de herramientas para la creación de repositorios software que puedan compartir información a través de Internet. RIB presenta una extensión del BIDM (*Basic Interoperability Data Model*) que es el estándar de IEEE 1420.1 [IEEE, 1995], para la catalogación de software en Internet. El BIDM ha sido desarrollado por el RIG (*Reuse Library Interoperability Group*) [NHSE, 1997].

El proyecto **RBSE** (*Repository Based Software Engineering*) [Eichmann, 1995; Eichmann et al., 1995] es un proyecto de investigación desarrollado en el *Research Institute for Computing and Information Systems* de la Universidad de Houston, y que tiene como objetivo crear un mecanismo de transferencia de tecnología para mejorar las capacidades en ingeniería del software de la NASA, dando soporte a la reutilización del software mediante un repositorio. Este proyecto tiene dos ramas principales: la iniciativa de ingeniería de reutilización y la iniciativa de repositorio. La parte de ingeniería de reutilización se basa en dos pilares que son el ISC (*Information Systems Contract*) y el proyecto ROSE (*Reusable Object Software Engineering*). La parte de repositorio descansa sobre MORE (*Multimedia Oriented Repository Environment*) que ofrece todo un sistema de gestión de repositorio. Como caso concreto o instancia de MORE se ha desarrollado ELSA (*Electronic Library Services and Applications*) que expande la vista tradicional de biblioteca software con la adquisición, clasificación,

almacenamiento y mantenimiento de todo tipo de *assets*, con la potencia añadida que otorga la hipermedia dentro de un entorno web.

La biblioteca de reutilización GIRO (http://www.infor.uva.es) [Hernández et al., 2001] ha sido desarrollada por el grupo GIRO en el proyecto de investigación MENHIR (CICYT - TIC97-0593-C05-05). La base fundamental del modelo de reutilización propuesto en este repositorio está constituida por un elemento reutilizable de grano grueso con soporte simultáneo de varios niveles de abstracción, que recibe el nombre de mecano. El modelo de reutilización y la estructura de componente reutilizable propuestos están exhaustivamente expuestos y defendidos en [García, 2000b].

Para un estudio más pormenorizado de las bibliotecas de reutilización se recomienda el estudio de [Mili et al., 1998].

Ingeniería de dominio

La ingeniería de dominio surge como una evolución de la reutilización sistemática del software basada en modelos donde las arquitecturas software juegan un papel ampliamente destacado.

La reutilización del software dentro de un dominio de aplicación pasa por el descubrimiento de elementos comunes a los sistemas pertenecientes a dicho dominio. Cuando se utiliza este enfoque se está produciendo un cambio de un desarrollo orientado a un único producto software a un desarrollo centrado en varios productos que comparten unas características formando una familia. Esto provoca una reestructuración del proceso software de forma que surgen dos procesos distintos: *la ingeniería de dominio* y *la ingeniería de aplicación*.

La ingeniería de dominio se centra en el desarrollo de elementos reutilizables que formarán la familia de productos, mientras que la ingeniería de aplicación se orienta hacia la construcción o desarrollo de productos individuales, pertenecientes a la familia de productos, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico, reutilizando, adaptando e integrando los elementos reutilizables existentes y producidos en la ingeniería de dominio.

La ingeniería de dominio se puede definir como el proceso clave que se necesita para el diseño sistemático de una arquitectura y de un conjunto de elementos software reutilizables que pueden ser usados en la construcción de una familia de aplicaciones relacionadas o subsistemas. Es el proceso sistemático que incorpora criterios de negocio y produce un soporte racional, modelos y arquitecturas que permiten tomar mejores decisiones, llevar un registro del dominio, obtener nuevas versiones y mejorar el proceso de desarrollo gracias al conocimiento que se tiene del sistema [Griss, 1996a; Griss et al., 1998].

El objetivo fundamental de la ingeniería de dominio es la optimización del proceso de desarrollo del software en un espectro de múltiples aplicaciones que representan un negocio común o problema de dominio [Simos et al., 1996].

La ingeniería de dominio ofrece un conjunto de modelos de referencia que describen el dominio, identificando las arquitecturas y los componentes que permitirán el desarrollo de aplicaciones dentro de él. Además, ofrece la base de conocimiento adecuada para comprender el espacio del problema definido por el software presente en el dominio. Es, por tanto, una parte clave en la consecución de una arquitectura y un conjunto de elementos reutilizables que reúnan la calidad suficiente para confiar en ellos.

La ingeniería de dominio tiene sus orígenes en los trabajos de James M. Neighbors sobre reutilización basada en generación a través del paradigma Draco [Neigbors, 1984] a principios de la década de los ochenta. Desde entonces se han publicado varias aproximaciones que incluyen un amplio rango de métodos y procesos, tanto formales como informales, para llevar a cabo las diferentes actividades en que se descompone la ingeniería de dominio, para capturar y representar información sobre sistemas que comparten un conjunto común de características y datos. Entre las diversas propuestas existentes en la ingeniería de dominios cabe destacar como las más representativas a FODA (*Feature-Oriented Domain Analysis*) [Kang et al., 1990], OODA (*Object Oriented Domain Analysis*) [Cohen y Northrop, 1998], ODM (*Organization Domain Modeling*) [Simos, 1995; Simos et al., 1996], DAGAR (*Domain Architecture-based Generation for Ada Reuse*) [Klingler y Solderitsch, 1996], FORM (*Feature-Oriented Reuse Method*) [Kang, 1998; Kang et al., 1998; Lee et al., 2000], FeatureRSEB (*Feature Reuse-Driven Software Engineering Business*) [Griss et al., 1998] y FODAcom [Vici y Argentieri, 1998].

Actualmente, una de las formas de ingeniería de dominio que más éxito está teniendo es la organización de los dominios en las llamadas líneas de productos.

Líneas de productos

La reutilización del software es uno de los objetivos fundamentales dentro de la Ingeniería del Software. Ya en los últimos años de la década de los sesenta, la idea de construir sistemas mediante la composición de componentes software fue presentada como solución a la afamada crisis del software por M. D. McIlroy [McIlroy, 1976]. Durante la década de los setenta se propugnó la reutilización de los módulos, mientras que en los años ochenta la influencia del paradigma orientado a objetos hizo que la clase se convirtiera en la unidad de reutilización. Sin embargo, todos estas tendencias fallaban en conseguir un enfoque sistemático de reutilización porque daban lugar a iniciativas individuales, frecuentemente realizadas a pequeña escala.

Con estos enfoques se deja de lado la reutilización de elementos software de mayor grano, que pueden corresponderse con partes significativas de los sistemas a construir, y además suelen necesitar de una adaptación de muchos de sus aspectos. Para solucionar este apartado surgen los *frameworks* [Wirfs-Brock y Johnson, 1990] y la programación orientada a componentes [Szyperski, 1998] como aproximaciones a la reutilización del software.

No obstante, aunque con estas construcciones el proceso de reutilización mejora en cuanto a alcance y nivel de abstracción soportado, siguen sin obtenerse los beneficios esperados, entre otras cosas porque el enfoque exclusivamente ascendente que se deriva de la composición de estos elementos falla en la práctica [Bosch, 2000].

Surgen así las líneas de productos como una de las aproximaciones más prometedoras en el campo de la reutilización al conjugar elementos de grano grueso, como son las arquitecturas software y los componentes, con un enfoque descendente y sistemático, en el que los componentes software se integran en una estructura arquitectónica de mayor nivel.

El concepto de línea de productos tiene su origen en las escuelas de negocio en la década de los ochenta, con un claro objetivo económico mediante el desarrollo sinérgico de productos [Knauber y Succi, 2001].

Existen numerosas definiciones de línea de productos en la bibliografía [Sonnemann, 1995; Cohen et al., 1995; Bass et al., 1997; Bosch, 2000; Griss, 2000], pero en este proyecto se va a tomar como referencia al concepto de línea de productos la siguiente: "*Una línea de productos es un conjunto de sistemas software que comparten un conjunto de características común y gestionado, que satisface las necesidades específicas de un segmento de mercado y que son desarrollados a partir de un conjunto central de assets de una forma establecida*" [SEI, 2001].

Un concepto que está muy íntimamente relacionado con las líneas de productos es el de familia de productos. Así, una familia de productos se puede definir como: "el conjunto de productos diferentes que pueden ser producidos desde un diseño común, *assets* compartidos y mediante un proceso de ingeniería de aplicaciones. La pertenencia a este conjunto depende de la abstracción que unifica los *assets* en un sistema que funciona: una arquitectura, las reglas físicas o de negocio, o la plataforma hardware" o como "el conjunto de productos que comparten una plataforma común, pero tiene características y funcionalidad requeridas por el cliente".

Una línea de productos no necesita construirse como una familia de productos, aunque es la forma de obtener los mayores beneficios. Además, una familia de productos no necesita constituir una línea de productos si los productos resultantes tienen poco en común en términos de mercado o de sus características.

Una línea de productos no es un grupo de productos producidos por una única unidad de negocio. Puede esperarse que haya una gran correlación entre las líneas de productos y las unidades de negocio, pero conceptualmente son cosas distintas. Una unidad de negocio se forma por razones de organización o financieras, y puede ser responsable de una o más líneas de productos; mientras que una línea de productos comparte y gestiona un conjunto común de características que satisfacen las necesidades específicas de un mercado seleccionado. Teóricamente, líneas de productos provenientes de diferentes unidades de negocio pueden mezclarse para formar una "super línea de productos".

Siguiendo con las diferencias, una línea de productos tampoco es sinónimo de dominio. Mientras que un dominio es un cuerpo especializado de conocimiento, un área de experiencia o una colección de funcionalidad relacionada, la línea de productos hace referencia a los sistemas software que comparten las características particulares de un sector de mercado. Por ejemplo, en el caso de las telecomunicaciones, el dominio de las telecomunicaciones es el conjunto de problemas relacionados con las telecomunicaciones, que puede estar formado por otros subdominios tales como protocolos, telefonía, redes... Por otro lado, una línea de productos de telecomunicaciones es un conjunto específico de sistemas que tratan algunos de esos problemas.

En [Bosch, 2000] se describen tres dimensiones en las que se pueden descomponer los conceptos involucrados en las líneas de productos, estas dimensiones son: 1) Arquitectura, componente y sistema; 2) Negocio, organización, proceso y tecnología; y 3) Desarrollo, instanciación y evolución.

La primera dimensión divide el dominio de la línea de productos de acuerdo a sus *assets* principales que son parte del desarrollo basado en reutilización, esto es, *arquitectura*, *componentes* y *sistema*. Esta aproximación se presenta también en [Jacobson et al., 1997], recibiendo el nombre de *ingeniería de familia de aplicaciones*, *ingeniería del sistema de componentes* e *ingeniería de aplicación respectivamente.*

La segunda dimensión está relacionada con las diferentes vistas de una organización: *negocio*, *organización*, *proceso* y *tecnología*. En diferentes reuniones de investigación, relacionadas con las líneas de productos, se ha utilizado la descomposición de esta dimensión [Clements, 1997; Bass et al., 1997; Bass et al., 1998; Bass et al., 1999; Bass et al., 2000].

La tercera dimensión se centra en el ciclo de vida de cada uno de los *assets* de la organización, distinguiéndose los siguientes aspectos: *desarrollo, instanciación* y *evolución*.

El inicio de una línea de productos supone una considerable inversión de tiempo, esfuerzo y dinero, de forma que aunque prometa numerosos beneficios, se encuentra con la reticencia natural de las organizaciones, especialmente las más pequeñas, a dicha inversión.

En [Bosch, 2000] se distinguen dos formas de iniciar una línea de producto en relación con el dominio, bien existe una base para la línea de productos formada por una familia de productos previamente desarrollados y una experiencia explícita en el desarrollo de estos productos, o bien se quiere empezar una línea de productos desde cero.

El enfoque que se va a seguir en este proyecto de investigación, es partir de la experiencia de un grupo de investigación en el terreno de las aplicaciones software para células CIM, y sobre este conocimiento, organizar el software desarrollado [González et al., 2000; García et al., 2000; González, 2001; Curto et al., 2001] en una línea de productos, cuya arquitectura base se diseña desde cero.

6.5.2 Objetivos del proyecto

El objetivo general de este proyecto es iniciar una línea de productos en el sector de las células CIM, que canalice y organice la experiencia en este campo del Grupo de Robótica del Departamento de Informática y Automática de la Universidad de Salamanca, de forma que un futuro próximo se puedan realizar nuevas aplicaciones en este dominio utilizando técnicas de ingeniería de aplicación, para obtener así los beneficios de la reutilización del software en estos desarrollos.

Como se va a comprobar a través de los objetivos que se plantean a continuación, este proyecto de investigación tiene un carácter aplicado y puede enmarcarse en el ámbito de las Tecnologías Avanzadas de la Producción, aunque los avances se puedan extender al campo de las Tecnologías de la Información. En cuanto a su interés socio-económico en la Comunidad Autónoma hay que destacar su aplicabilidad a la industria manufacturera y, en especial, a las empresas relacionadas con el sector automovilístico, que tiene un indudable peso específico en la realidad socio-económica de Castilla y León. No obstante, cualquier industria con procesos productivos orientados a eventos discretos podría mejorar las condiciones de operación con la utilización de la línea de productos que se pretende iniciar.

A continuación se van a enumerar los objetivos concretos que se persiguen en este proyecto. Éstos se concretan en diferentes soluciones en pro de cumplir el objetivo global planteado:

1. Hacer un estudio del dominio de las células CIM.

No se puede pretender abordar la puesta en marcha de una línea de productos sin haber delimitado previamente el contexto de la misma.

En el caso concreto de este proyecto, como ya se ha mencionado con anterioridad, el campo de trabajo es el de las células CIM, y aquí el grupo de investigación tiene un amplio bagaje que habrá que organizar en pro de establecer los límites y el alcance de la línea de productos que se quiere iniciar.

El objetivo último será la **organización del conocimiento que se tiene sobre el dominio de aplicación donde se va a circunscribir la línea de productos.**

2. Definición de un lenguaje de patrones software para células CIM.

El diseño de la arquitectura base de la línea de productos a construir es una de las tareas críticas en pro de la consecución de los requisitos no funcionales y restricciones de diseño del sistema, sobre los que recaen las características relacionadas con la distribución, comportamiento en tiempo real, flexibilidad, independencia de plataforma...

Para llevar a cabo esta actividad se recurrirá al empleo de diferentes patrones de diseño, adaptándolos en la mayoría de las ocasiones a las circunstancias del problema aquí tratado, pudiéndose incluso definir nuevos patrones.

El objetivo último será **definir un lenguaje de patrones adaptados y preparados para su utilización en el contexto del diseño de software para células CIM**.

3. Instalación de un repositorio de reutilización.

El repositorio es el elemento de soporte a la reutilización, y por lo tanto imprescindible en un proyecto de estas características.

Motores de repositorios hay muchos tanto comerciales como de libre distribución, pero dado que se van a utilizar mecanos para soportar las líneas de productos, se necesita un repositorio con soporte para ellos, o en último caso para elementos reutilizables de grano grueso. En este caso la mejor opción es utilizar el repositorio GIRO (http://www.infor.uva.es) [Hernández et al., 2001].

Así, el objetivo último será **instalar y configurar un repositorio interno basado en el motor de repositorios GIRO**.

4. Herramientas software de soporte (comportamiento en tiempo real).

El desarrollo de este proyecto requiere de un conjunto de herramientas de software para culminar con el objetivo final. Entre ellas hay que considerar la instalación y puesta en marcha del software necesario para la implementación de los componentes basados en CORBA, donde será necesario realizar las modificaciones pertinentes para un comportamiento en tiempo real sobre las peticiones de servicio que se realicen sobre los objetos distribuidos.

Otro esfuerzo será necesario para que las comunicaciones aporten las condiciones de predecibilidad necesarias para cumplir con los requisitos de tiempo real totales (*end-to-end*). En este sentido se plantea la utilización de técnicas del campo de las redes activas para conseguir este objetivo. Finalmente, la utilización de un soporte de sistema operativo de tiempo real es el que finalmente podrá garantizar la fiabilidad necesaria del producto final, por lo que será la plataforma donde desarrollar los diferentes *drivers*, protocolos, planificadores...

Así, el objetivo último será **establecer el entorno software y hardware de soporte al sistema donde se va a iniciar la línea de productos**.

5. Definición de una línea de productos software en célula de fabricación.

El conocimiento de los problemas relacionados con las células de fabricación, así como de las aportaciones teóricas para abordar sus soluciones, unido al gran esfuerzo para el diseño e implementación de un software flexible, altamente configurable y fácilmente extensible, justifican que el producto final de este trabajo no sea un “simple” sistema software, sino una línea de productos establecida en el dominio de aplicación del software para células de fabricación.

En dicha línea de productos queda embebido todo el conocimiento del problema, con una arquitectura de carácter distribuido donde sus elementos reutilizables, en forma de mecanos, posibilitan la fácil extensión y evolución del sistema software, así como su sencilla adaptación a diferentes configuraciones de células en distintos entornos industriales.

Así, el objetivo último será **iniciar una línea de productos en el dominio de las células CIM**.

6.5.3 Metodología y plan de trabajo

Para la consecución de los objetivos planteados en el presente proyecto, se ha dividido el mismo en una serie de tareas agrupadas por afinidades y asignadas a los miembros del equipo de investigación (formado por seis investigadores). La coordinación interna estará a cargo del

responsable del proyecto y se llevará a cabo mediante la realización de una sesión quincenal de control de resultados.

Está prevista la incorporación de un Becario de Investigación que realice labores de soporte al mismo tiempo que se inicia en la investigación y realiza sus Tesis Doctoral.

A continuación se describe, por objetivos y tareas, con detalle de resultados esperados y plazos de ejecución, la planificación del proyecto.

Modulo 1. Configuración de la planta tipo

Duración

3 meses (desde año 1, mes 1 hasta año 1, mes 3)

Descripción

En este bloque de tareas se pretende configurar el entorno de trabajo donde se va a llevar a cabo el grueso de la investigación y los desarrollos relacionados con las células CIM

Resultados esperados

- Adquisición y configuración de los elementos hardware
- Adquisición y configuración de los elementos software
- Instalación y configuración del motor de repositiorio

Plan de trabajo (descomposición en tareas)

TAREA: Establecimiento de los elementos hardware de la planta tipo

CÓDIGO: T1.1

COMIENZO: Año 1, mes 1

FINAL: Año 1, mes 3

RESPONSABLE: Investigador 1

ENTRADAS: Ninguna

DESCRIPCIÓN: Adquisición de los elementos hardware necesarios y puesta a punto de la planta tipo

PARTICIPANTES: Investigador 2, Investigador 3, Investigador 4, Becario

TAREA: Establecimiento de los elementos software de la planta tipo

CÓDIGO: T1.2

COMIENZO: Año 1, mes 1

FINAL: Año 1, mes 3

RESPONSABLE: Investigador 4

ENTRADAS: Informes de la tarea T1.1

DESCRIPCIÓN: Adquisición de los elementos software necesarios y puesta a punto de los mismos

PARTICIPANTES: Investigador 1, Investigador 2, Investigador 3, Becario

TAREA: Instalación y configuración del repositorio GIRO

CÓDIGO: T1.3

COMIENZO: Año 1, mes 1

FINAL: Año 1, mes 3

RESPONSABLE: Investigador Principal

ENTRADAS: Ninguna

DESCRIPCIÓN: Se instalará y configurará el motor de repositorios GIRO para su posterior utilización en el Módulo 2

PARTICIPANTES: Investigador 5, Becario

Modulo 2. Iniciación de la línea de productos para las células CIM

Duración

15 meses (desde año 1, mes 4 hasta año 2, mes 6)

Descripción

En este bloque de tareas se inicia la línea de productos. Como la organización donde se pone en marcha es un grupo de investigación pequeño, y el esfuerzo de iniciar una línea de productos es grande, se va a seguir un proceso ligero, adaptado a las necesidades y objetivos del grupo, y donde se van a relajar ciertos aspectos organizativos. Aunque este módulo se va expresar de forma secuencial, tendrá lugar de forma iterativa e incremental

Resultados esperados

- Definición de un proceso ligero para la iniciación de líneas de productos
- Definición, diseño e implementación de una línea de productos básica, compuesta por una arquitectura base y un conjunto de componentes, en el dominio de las células CIM
- Almacenamiento de los *assets* que forman la línea de productos en el repositorio en forma de mecanos

Plan de trabajo (descomposición en tareas)

TAREA: Definición del dominio

CÓDIGO: T2.1

COMIENZO: Año 1, mes 4

FINAL: Año 1, mes 6

RESPONSABLE: Investigador Principal

ENTRADAS: Aplicaciones, bibliografía, artículos...

DESCRIPCIÓN: El propósito básico de esta fase es seleccionar el dominio propiamente dicho, apropiadamente enfocado y alineado con una estrategia más amplia dentro de los intereses de la organización

PARTICIPANTES: Investigador 1, Investigador 2, Investigador 4

TAREA: Ingeniería de requisitos para la línea de productos

CÓDIGO: T2.2

COMIENZO: Año 1, mes 6

FINAL: Año 1 mes 11

RESPONSABLE: Investigador Principal

ENTRADAS: Informes T2.1

DESCRIPCIÓN: En los requisitos de una línea de productos se ven reflejados requisitos de todos los productos que forman esa línea de productos, incluso de aquéllos que todavía no han sido desarrollados. Por tanto, además de la información que representa un requisito, interesa conocer la variabilidad de éstos y las dependencias de obligatoriedad o exclusión entre ellos. Para representar esta información se recurre a representaciones jerárquicas de los requisitos

PARTICIPANTES: Investigador 1, Investigador 2, Investigador 4

TAREA: Definición de la arquitectura de referencia

CÓDIGO: T2.3

COMIENZO: Año 1, mes 9

FINAL: Año 2, mes 2

RESPONSABLE: Investigador Principal

ENTRADAS: Informes T2.1, T2.2

DESCRIPCIÓN: Una vez que los requisitos de la línea de productos han quedado establecidos, éste es el paso más crítico en la iniciación de la línea de productos, pues esta arquitectura base o de referencia será reutilizada en todos y cado uno de los productos que nutrirán la línea de productos. No obstante, el diseño de la arquitectura de referencia es probablemente el aspecto más creativo del proceso, y por tanto el menos sujeto a normalización. Su definición está condicionada por la experiencia del arquitecto y el tipo de producto que se construye. En dominios bien estudiados, el uso de arquitecturas clásicas como cliente-servidor, por ejemplo, será suficiente, pero en otras situaciones será necesaria toda la capacidad inventiva del

arquitecto. En cualquier caso, es preferible diseñar desde cero pensando en su reutilización, pues el resultado es más óptimo, además de ser normalmente un proceso más rápido, que si se intenta evolucionar la arquitectura de alguno de los productos ya existentes

PARTICIPANTES: Investigador l, Investigador 3, Investigador 5, Becario

TAREA: Desarrollo de los componentes

CÓDIGO: T2.4

COMIENZO: Año 1, mes 12

FINAL: Año 2, mes 4

RESPONSABLE: Investigador 1

ENTRADAS: Resultados de las tareas T2.1, T2.2, T2.3

DESCRIPCIÓN: El diseño e implementación de la línea de productos, o más propiamente dicho de la arquitectura de la misma, continua con el diseño de un conjunto de componentes que completen esa arquitectura de referencia. Estos componentes pueden diseñarse desde cero, aunque también pueden incorporar el código legado de los productos ya existentes, bien aplicando diversas técnicas de diseño para reutilización o bien realizando envoltorios que ofrezcan interfaces acordes con el diseño arquitectónico, pero que hagan uso de elementos ya existentes

PARTICIPANTES: Investigador 2, Investigador 4, Investigador 5, Becario

TAREA: Población del repositorio

CÓDIGO: T2.5

COMIENZO: Año 2, mes 2

FINAL: Año 2, mes 6

RESPONSABLE: Investigador 5

ENTRADAS: Resultados de las tareas T2.1, T2.2, T2.3, T2.4

DESCRIPCIÓN: El repositorio o biblioteca de reutilización ofrece el soporte operativo para el almacenamiento y gestión de los elementos de la línea de productos, mecanos en esta propuesta

PARTICIPANTES: Investigador 2, Becario

Modulo 3. Transferencia tecnológica

Duración

6 meses (desde año 2, mes 7 hasta año 2, mes 12)

Descripción

Se realizará la transferencia tecnológica de los resultados de los otros módulos a medida que la madurez de las tecnologías desarrolladas lo permita, con aplicación a situaciones reales

Resultados esperados

- Realización de productos que nutran la línea de productos

Plan de trabajo (descomposición en tareas)

TAREA: Transferencia tecnológica

CÓDIGO: T3.1

COMIENZO: Año 2, mes 7

FINAL: Año 2, mes 12

RESPONSABLE: Investigador 2

ENTRADAS: Línea de productos base

DESCRIPCIÓN: Se realizarán desarrollos de células CIM basadas en la línea de productos iniciada para empresas colaboradoras

PARTICIPANTES: Todo el equipo

En la Tabla 6.2 se resume el plan de trabajo del presente proyecto de investigación.

Módulo	Tareas	Investigadores	Primer año	Segundo año
1 Configuración de la planta tipo	T1.1 Establecimiento de los elementos hardware de la planta tipo	I1, I2, I3, I4, B		
	T1.2 Establecimiento de los elementos software de la planta tipo	I1, I2, I3, I4, B		
	T1.3 Instalación y configuración del repositorio GIRO	I4, B1		
2 Iniciación de la línea de productos paras las células CIM	T2.1 Definición del dominio	IP, I1, I2, I4		
	T2.2 Ingeniería de requisitos para la línea de productos	IP, 11, I2, I4		
	T2.3 Definición de la arquitectura de referencia	IP, I1, I3, I5, B		
	T2.4 Desarrollo de los componentes	I1, I2, I4, I5, B		
	T2.5 Población del repositorio	I2, I5, B		
3 Trasf. Tecnológica	T3.1 Transferencia tecnológica	I1, I2, I3, I4, I5. IP, B		

Tabla 6.2. Planificación temporal del proyecto de investigación sobre la iniciación de la línea de productos en células CIM

6.6 Referencias

[Adam et al., 1996] Adam, N., Yesha, Y., Awerbuch, B., Bennet, K., Blaustein, B., Brodsky, A., Chen, R., Dogramaci, O., Grossman, B., Holowczak, R., Johnson, J., Kalpakis, K., McCollum, C., Neches, A.-L., Neches, B., Rosenthal, A., Slonim, J., Wactlar, H., Wolfson, O., Yesha, Y. "*Strategic Directions in Electronic Commerce and Digital Libraries: Towards a Digital Agora*". ACM Computing Surveys 28(4):818-835. December 1996.

[Ader et al., 1990] Ader, M., Nierstrasz, O., McMahon, S., Muller, G., Pröfrock, A.-K. "*The ITHACA Technology: A Landscape for Object-Oriented Application Development*". In proceedings of ESPRIT'90 Conference. Kluwer Academic Publisher, 1990.

[Aklecha, 1999] Aklecha, V. "*Object-Oriented Frameworks Using C++ and CORBA. Gold Book*". Coriolis Technology Press, 1999.

[Antonellis y Pernici, 1995] Antonellis, V. de, Pernici, B. "*Reusing Specifications through Refinement Levels*". Data and Knowledge Engineering, 15(2):109-133. April 1995.

[Ardissono et al., 1999] Ardissono, L., Barbero, C., Goy, A., Petrone, G. "*An Agent Architecture for Personalized Web Stores*". In Proceedings of the Third International Conference on Autonomous Agents – AGENTS'99. (Seattle, WA, USA, 1999). Pages 182-189. ACM Press, 1999.

[Ardissono y Goy, 2000] Ardissono, L., Goy, A. "*Tailoring the Interaction With Users in Web Stores*". User Modeling and User-Adapted Interaction, 10(4):251-303. Kluwer Academic Publishers. 2000.

[Armstrong y Durfee, 1998] Armstrong, A., Durfee, E. "*Mixing and Memory: Emergent Cooperation in an Information Marketplace*". In Proceedings of the Third International Conference on Multi-Agent Systems ICMA'98. Pages 34-41. 1998.

[Baron et al., 2000] Baron, J. P., Shaw, M. J., Bailey, A. D. Jr. "*Web-based E-catalog Systems in B2B Procurement*". Communications of the ACM, 43(5):93-100. May 2000.

[Bailey y Bakos, 1997] Bailey, J., Bakos, Y. "*An Exploratory Study of Emerging Role of Electronic Intermediaries*". International Journal of Electronic Commerce, 1(3):7-20. Spring 1997.

[Bakos, 1991] Bakos Y. "*A Strategic Analysis of Electronic Marketplace*". MIS Quarterly, 15(4):295-310. December 1991.

[Bakos, 1998] Bakos, Y. "*The Emerging Role of Electronic Marketplaces on the Internet*". Communications of the ACM, 41(8):35-42. August 1998.

[Bass et al., 1997] Bass, L., Clements, P., Cohen, S., Northrop, L. M., Withey, J. "*Product Line Practice Workshop Report*". Technical Report CMU/SEI-97-TR-003 (ESC-TR-97-003), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 1997.

[Bass et al., 1998] Bass, L., Chastek, G., Clements, P., Northrop, L. M., Smith, D., Withey, J. "*Second Product Line Practice Workshop Report*". Technical Report CMU/SEI-98-TR-015 (ESC-TR-98-015), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 1998.

[Bass et al., 1999] Bass, L., Campbell, G., Clements, P. C., Northrop, L., Smith, D. "*Third Product Line Practice Workshop Report*". Technical Report CMU/SEI-99-TR-03 (ESC-TR-99-003), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213 (USA).

[Bass et al., 2000] Bass, L., Clements, P., Donohoe, P., McGregor, J., Northrop, L. "*Fourth Product Line Practice Workshop Report*". Technical Report CMU/SEI-2000-TR-002 (ESC-TR-2000-002), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 2000.

[Bellinzona et al., 1993] Bellinzona, R., Fugini, M. G., de Mey, V. "*Reuse of Specifications and Designs in a Development Information System*". In Prakash, N., Rolland, C., Percini, B. (Editors), *Information System Development Process*. Pages 79-96. Amsterdam. North-Holland, 1993.

[Benjamin y Wigand, 1995] Benjamin, R., Wigand, R. "*Electronic Markets and Virtual Value Chains on the Information Highway*". Sloan Management Review, 36(2):62-72. Winter 1995.

[Bertoletti y da Rocha, 1999] Bertoletti, A. C., da Rocha Costa, C. "*SAGRES – A Virtual Museum*". In Proceedings of Museums and the Web. (New Orleans, LA, USA). 1999.

[Bichler, et al., 1998] Bichler, M., Segev, A., Beam, C. "*An Electronic Broker for Business-To-Business Electronic Commerce on the Internet*". International Journal of Cooperative Information Systems, 7(4):315-330. 1998.

[Biggerstaff, 1992] Biggerstaff, T. J. "*An Assessment and Analysis of Software Reuse*". In M. C. Yovits (Editor). *Advances in Computers*, 34:1-57. Academic Press, Inc.1992.

[BOCG, 2001] Boletín Oficial Cortes Generales. "*Ley Orgánica de Universidades*". Congreso de los Diputados. Serie A. Núm. 45-13. Páginas 463-495. 26 de diciembre de 2001.

[BOE, 1983] *Boletín Oficial del Estado de 1 de Septiembre de 1983;* Ley Orgánica 11/1983, de 25 de agosto, de reforma universitaria.

[Borrego et al., 2001] Borrego, I., Hernández, M. J., García, F. J., Curto, B., Moreno, V., Hernández, J. A. "*Arquitectura Automatizada de Comercio Electrónico*". Actas del 2º Congreso Internacional de Interacción Persona-Ordenador – Interacción 2001. J. Abascal, F. J. García, A. B. Gil (Eds.). (16-18 de mayo de 2001, Salamanca). Páginas 423-427. Ediciones Universidad de Salamanca. Colección Aquilafuente, Nº 19. 2001.

[Bosch, 2000] Bosch, J. "*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*". Addison-Wesley, 2000.

[Bray et al., 2000] Bray, T., Paoli, J,. Sperberg-MacQueen, C. M. "*Extensible Markup Language (XML) 1.0*" (Second Edition). World Wide Web Consortium Recommendation October 2000. http://www.w3c.org/TR/2000/REC-xml-20001006. 2000.

[Brusilovsky, 1996] Brusilovsky, P. "*Methods and Techniques of Adaptive Hypermedia*". User Modeling and User Adapted Interaction, 6(2/3):87-129. 1996.

[Brusilovsky, 2001] Brusilovsky, P. "*Adaptive Hypermedia*". User Modeling and User Adapted Interaction, 11(1/2):87-110. 2001.

[Brusilovsky y Cooper, 1999] Brusilovsky, P., Cooper, D. W. "*ADAPTS: Adaptive Hypermedia for a Web-based Performance Support System*". In Proceedings of Second Workshop on Adaptive Systems and User Modeling on WWW at 8th International World Wide Web Conference and 7th International Conference on User Modeling. (Toronto, Canada, 1999). Computer Science Report 99-07, Eindhoven University of Technology. Pages 41-47. 1999.

[Burns y Wellings, 1997] Burns, A., Wellings, A. "*Real-Time Systems and Programming Languages*". Addison-Wesley, 1997.

[Buxmann y Gebauer, 1998] Buxmann, P., Gebauer, J. "*Internet-based Intermediaries – The Case of the Real Estate Market*". In Proceedings of the 6th European Conference on Information Systems – ECIS'98. (Aix-en-Provence, France, June 4-6, 1998). 1998.

[Carmel y Markovitch, 1998] Carmel, D., Markovitch, S. "*How to Explore Your Opponent's Strategy (almost) Optimally*". In Proceedings of the Third International Conference on Multi-Agent Systems ICMA'98. Pages 64-71. 1998.

[Casati y Shan, 2001] Casati, F., Shan, M.-C. "*Dynamic and Adaptive Composition of E-Services*". Information Systems, 26(3):143-163. May 2001.

[Castano y Antonellis, 1994] Castano, S., Antonellis, V. de. "*The F3 Reuse Environment for Requirements Engineering*". ACM Software Engineering Notes, 19(3). 1994.

[Clements, 1997] Clements, P. "*Report of the Reuse and Product Lines Working Group of WISR8*". Special Report CMU/SEI-97-SR-010, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 1997.

[Cohen et al., 1995] Cohen, S. G., Friedman, S., Martin, L., Solderitsch, N., Webster, R. "*Product Line Identification for ESC-Hanscom*". Special Report CMU/SEI-95-SR-024, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 1995.

[Cohen y Northrop, 1998] Cohen, S. G., Northrop, L. M. "*Object-Oriented Technology and Domain Analysis*". In Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, (June 2-5, 1998, Victoria, B.C., Canada). Pages 86-93. IEEE-CS, 1998.

[Constantopoulos et al., 1992] Constantopoulos, P., Jarke, M., Mylopoulos, J., Vassiliou, Y. "*The Software Information Base: A Server for Reuse*". ITHACA.FORTH.92.E2#1, FORTH Computer Science Institute, Iraklion (Greece). 1992.

[Corchado, 2001] Corchado, J. M. "*CBR-BDI Agents for an E-commerce Environment*". In Proceedings of the Workshop On Object-Oriented Business Solutions – WOOBS'01. Celebrated under the auspices of the 15th European Conference on Object-Oriented Programming ECOOP'01. R. Corchuelo, A. Ruiz, J. Mühlbacher, J. D. García-Consuegra editors. (Budapest, Hungary, June 18-19, 2001). Pages 13-22. 2001.

[Corradi et al., 1999] Corradi, A., Cremonini, M., Montanari, R., Stefanelli, C. "*Mobile Agents Integrity for Electronic Commerce Applications*". Information Systems, 24(6):519-533. 1999.

[Curto, 1998] Curto, B. "*Formalismo Matemático para la Representación de Obstáculos en el Espacio de las Configuraciones de un Robot*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Septiembre, 1998.

[Curto et al., 2001] Curto, B., García, F. J., Moreno, V., González, J., Moreno, Á. Mª. "*An Experience of a CORBA Based Architecture for Computer Integrated Manufacturing*". In Proceedings of 8th IEEE International Conference on Emerging Technologies and Factory Automation – ETFA 2001. (Antibes – Juan les Pins, France, October 15-18, 2001). Pages 765-769. IEEE Press, 2001.

[Cybulski, 1996] Cybulski, J. L. "*Introduction to Software Reuse*". Technical Report 96/4. Department of Information Systems. University of Melbourne, 1996.

[Cybulski et al., 1997] Cybulski, J. L., Neal, R. D., Kram, A., Allen, J. C. "*Report on the Reuse of Early Life-Cycle Artefacts*". WISR-8; 8th Annual Workshop on Software Reuse/Working Group #5. 1997.

[Chavez y Maes, 1996] Chavez, A., Maes, P. "*Kasbah: An Agent Marketplace for Buying and Selling Goods*". In Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology. (London, United Kingdom, 1996). Pages 75-90. IEE, 1996.

[Chircu y Kauffman, 1999a] Chircu, A. M., Kauffman, R. J. "*Analyzing Firm-level Strategy for Internet-focused Reintermediation*". In Proceedings of the 32nd Hawaii International Conference on System Science. Sprague, R. H. (editor). IEEE-CS Press. 1999.

[Chircu y Kauffman, 1999b] Chircu, A. M., Kauffman, R. J. "*Strategies for Internet Middlemen in the Intermediation / Disintermediation / Reintermediation Cycle*". Electronic Markets – The International Journal of Electronic Commerce and Business Media, 9(2):109-117. May 1999.

[Chircu y Kauffman, 2000] Chircu, A. M., Kauffman, R. J. "*Reintermediation Strategies in Business-To-Business Electronic Commerce*". International Journal of Electronic Commerce, 4(4):7-42. Fall 2000.

[DoD, 1992] Department of Defense. "*DoD Software Reuse Initiative Vision and Strategy*". Technical Report 1222-04-10/40, Department of Defense (DoD Software Reuse Initiative), Falls Church, Va, 1992.

[Dogac y Cingil, 2001] Dogac, A., Cingil, I. "*A Survey and Comparison of Business-to-Business E-Commerce Frameworks*". SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-commerce, 2(2):16-27. Spring 2001.

[Doorenbos et al., 1997] Doorenbos, B., Etzioni, O., Weld, D. "*A Scalable Comparison-Shopping Agent for the World-Wide Web*". In Proceedings of the ACM Autonomous Agents'97. (Marina del Rey, USA, 1997). Pages 39-48. ACM Press, 1997.

[DPTOIA, 2001] Departamento de Informática y Automática. "*Memoria de Actividades*". Universidad de Salamanca. 2001.

[Eichmann, 1995] Eichmann, D. "*The Repository Based Software Engineering Program*". In Proceedings of the Fifth Systems Reengineering Technology Workshop, Monterrey, CA.February 7-9, 1995.

[Eichmann et al., 1995] Eichmann, D., Price, M., Terry, R. H., Welton, L. L. "*ELSA and MORE: A Library and an Environment for the Web*". http://rbse.mountain.net/MOREplus/ELSAandMORE. 1995.

[Fingar, 2000] Fingar, P. "*Component-based Frameworks for E-Commerce*". Communications of the ACM, 43(10):61-66. October 2000.

[Finin et al., 1995] Finin, T., Labrou, Y., Mayfield, J. "*KQML as an Agent Communication Language*" In *Software Agents*, Bradshaw, J. (Editor). MIT Press, 1995.

[Finin y Wiederhold, 1991] Finin, T., Wiederhold, G. "*An Overview of KQML: A Knowledge Query and Manipulation Language*". Stanford University Computer Science Department, 1991.

[Fink et al., 1998] Fink, J., Kobsa, A., Nill, A. "*Adaptable and Adaptive Information Provision for All Users, Including Disabled and Elderly People*". The New Review of Hypermedia and Multimedia, 4, 163-188. 1998.

[FIPA, 2001] FIPA. "*FIPA'2001 Specification 2: Agent Communication Language*". FIPA. http://www.fipa.org/. 2001.

[Fourie, 1991] Fourie, F. C. v. N. "*The Nature of the Market: A Structural Analysis*". In *Rethinking Economics – Markets, Technology and Economic Evolution*. Hodgson, G. M., Screpanti, E. (editors). Pages 40-57. Aldershot, 1991.

[Foisel et al., 1998] Foisel, R., Chevrier, V., Haton, J. "*Modeling Adaptative Organizations*". In Proceedings of the Third International Conference on Multi-Agent Systems ICMA'98. Pages 427-428. 1998.

[Foster, 1995] Foster, I. "*Designing and Building Parallel Programs. Concepts and Tools for Parallel Software Engineering*". Addison-Wesley, 1995.

[Francisco-Revilla y Shipman, 2000] Francisco-Revilla, L., Shipman, F. M. III. "*Adaptive Medical Information Delivery: Combining User, Task, and Situation Models*". In Proceedings of 2000 International Conference on Intelligent User Interfaces. (New Orleans, LA, USA, 2000). Pages 94-97. 2000.

[Freeman, 1987] Freeman, P. "*Reusable Software Engineering: Concepts and Research Directions*". In P. Freeman (Editor), *Tutorial: Software Reusability*. Pages 10-23. IEEE CS Press, 1987.

[Froehlich et al., 1999] Froehlich, G., Hoover, H. J., Liew, W., Sorenson, P. G. "*Application Framework Issues when Evolving Business Applications for Electronic Commerce*". Information Systems, 24(6):457-473. 1999.

[García, 2000a] García Alonso, D. "*Introducción al estándar FIPA*". Informe Técnico UCM-DSIP 98-00. Versión 1.0. Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid. Febrero de 2000.

[García, 2000b] García Peñalvo, F. J. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[García, 2001] García Peñalvo, F. J. "*Docencia Práctica en los Laboratorios de las Ingenierías en Informática Apoyada en Herramientas CASE – Memoria de Resultados*". Departamento de Informática y Automática, Universidad de Salamanca. Consejería de Educación y Cultura de la Junta de Castilla y León. Noviembre de 2001.

[García, 2002] García Peñalvo, F. J. "*Herramienta de Autor para el Desarrollo de Material Didáctico Multimedia. Memoria de Resultados del Proyecto SA002/01*". Departamento de Informática y Automática. Universidad de Salamanca. Enero, 2002.

[García et al., 2000] García, F. J., Moreno, V., Curto, B., González, J., Moreno, A., Blanco, J. "*Propuesta de una Arquitectura Software Basada en Componentes Distribuidos para una Célula CIM*". Actas del Primer Taller de Trabajo en Ingeniería del Software basada en Componentes Distribuidos - IScDIS'00, desarrollado dentro de las V Jornadas de Ingeniería del Software y Bases de Datos, JISBD'2000 (Valladolid, 8-10 de noviembre de 2000). J. García-Molina, J. Hernández, F. Sánchez, D. Sevilla, A. Vallecillo (Editores). Informe Técnico nº TR-12/2000. Dpto. de Informática. Universidad de Extremadura. (http://webepcc.unex.es/juan/iscdis00/). Páginas 69-75. Diciembre, 2000.

[García et al., 2001a] García, F. J., Borrego, I., Hernández, M. J. "*Publicación de Catálogos en una Arquitectura de Comercio Electrónico sobre la base de XML*". Actas del Simposio en Informática y Telecomunicación, SIT'2001 (A Coruña, 12-14 de septiembre de 2001). S. Barro Ameneiro, J. L. Freire Nistal, J. Rivero Laguna (Editores). Páginas 183-195. Serie Actas de Congresos y Reuniones Técnicas. Colección SIT-Simposio en Informática y Telecomunicación. Edita Fundación Dintel. 2001.

[García et al., 2001b] García, F. J., Moreno, Mª N., Hernández, J. A. "*An XML-Based E-Commerce Architecture Proposal*". In Proceedings of the Workshop On Object-Oriented Business Solutions – WOOBS'01. Celebrated under the auspices of the 15th European Conference on Object-Oriented Programming ECOOP'01. R. Corchuelo, A. Ruiz, J. Mühlbacher, J. D. García-Consuegra editors. (Budapest, Hungary, June 18-19, 2001). Pages 109-118. 2001.

[Genesereth y Ketchpel, 1994] Genesereth, M. R., Ketchpel, S. P. "*Software Agents*". Communications of the ACM, 37(7):48-53,147. July 1994.

[Gil et al., 2001] Gil, A. B., García, F. J., Guessoum, Z. "*Adaptive Agents for E-Commerce Applications*". Actas de la Reunión de trabajo sobre métodos y herramientas para el desarrollo de aplicaciones de comercio electrónico, ZOCO (Almagro – Ciudad Real, 20 de noviembre de 2001). Páginas 43-52. 2001.

[Ginige y Murugesan, 2001] Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. January-March 2001.

[Ginsburg et al., 1999] Ginsburg, M., Gebauer, J., Segev, A. "*Multi-Vendor Electronic Catalogs to Support Procurement: Current Practice and Future Directions*". In Proceedings of the Twelfth International Bled Electronic Commerce Conference. (Bled, Slovenia, June 7-9, 1999). 1999.

[González, 2001] González, J. "*Diseño e implementación de componentes distribuidos para una célula CIM*". Proyecto de Final de Carrera. Ingeniería Técnica en Informática de Sistemas. Facultad de Ciencias. Universidad de Salamanca. Marzo, 2001.

[González et al., 2001] González, J., Curto, B., García, F. J., Moreno, V., Blanco, J. "*Diseño de un Sistema Distribuido para la Gestión de una Célula CIM*". Actas del Simposio Español de Informática Distribuida, SEID 2000. Editores S. Barro, J. Mª Busta, J. M. Corchado, P. Cuesta (Ourense, 25-27 de septiembre de 2000). Páginas 127-134. 2000.

[Glushko et al., 1999] Glushko, R. J., Tenenbaum, J. M., Meltzer, B. "*An XML Framework for Agent-based E-commerce*". Communications of the ACM, 42(3):106-114. March 1999.

[Griss, 1993] Griss, M. L. "*Software Reuse: From Library to Factory*". IBM System Journal, 32(4):1-23. November 1993.

[Griss, 1996a] Griss, M. L. "*Domain Engineering and Variability in the Reuse-Driven Software Engineering Business*". Object Magazine, 6(7). 1996.

[Griss, 1996b] Griss, M. L. "*Systematic Software Reuse: Architecture, Process and Organization are Crucial*". Fusion Newsletter. http://hpl.hp.com/reuse/papers/fusion1.htm. 1996.

[Griss, 2000] Griss, M. L. "*Implementing Product-Line Features by Composing Component Aspects*". In Proceedings of First International Software Product Line Conference. Denver, CO (USA). August 2000.

[Griss et al., 1998] Griss, M. L., Favaro, J., d'Alessandro, M. "*Integrating Feature Modeling with RSEB*". In Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, (June 2-5, 1998, Victoria, B.C., Canada). Pages 76-85. IEEE-CS. 1998.

[Groover et al., 1989] Groover, M., Weiss, M., Nagel, R., Odrey, N. "*Industrial Robotics*". McGraw Hill, 1989.

[Guessoum et al., 2001] Guessoum, Z., Quenault, M., Durand, R. "*An Adaptive Agent Model*". In proceedings of. AIB'S, York, March 2001.

[Guessoum y Briot, 1999] Guessoum, Z., Briot, J.-P. "*From Active Objects to Autonomous Agents*". IEEE Concurrency, 7(3):68-76. 1999.

[Hagel y Singer, 1999] Hgel, J., Singer, M. "*Net Worth: Shaping Markets whwn Customers Make the Rules*". Harvard Business School Press, 1999.

[Harting, 2000] Harting, M. C. "*Business-To-Business E-Marketplaces: A Primer*". KPMG Consulting, 2000.

[Henning y Vinoski, 1999] Henning, M., Vinoski, S. "*Advanced CORBA Programming with C++*". Addison Wesley, 1999.

[Henrich, 1997] Henrich, D. "*Fast Motion Planning by Parallel Processing - A review*". Journal of Intelligent and Robotic System, 20(1):45-69, 1997.

[Hernández et al., 2001] Hernández, C., García, F. J., Laguna, M. Á. "*La Biblioteca de Reutilización GIRO*". Actas de las I Jornadas de Trabajo DOLMEN (Sevilla, 12 y 13 de junio de 2001). Páginas 103-112. Junio, 2001.

[Hernádez et al., 2001] Hernández, M. J., Borrego, I., García, F. J., Curto, B., Moreno, Mª N., Hernández, J. A. "*Herramienta Automatiza para la Generación de Catálogos de Venta en Internet*". Actas del 2º Congreso Internacional de Interacción Persona-Ordenador – Interacción 2001. J. Abascal, F. J. García, A. B. Gil (Eds.). (16-18 de mayo de 2001, Salamanca). Páginas 429-434. Ediciones Universidad de Salamanca. Colección Aquilafuente, Nº 19. 2001.

[Hernádez y García, 2001] Hernádez Gajate, M. J., García Peñalvo, F. J. "*XML y Comercio Electrónico*". Informe Técnico (DPTOIA-IT-2001-002), Universidad de Salamanca (España). http://tejo.usal.es/inftec/2001/DPTOIA-IT-2001-003.pdf. Diciembre, 2001.

[Hirashima et al., 1998] Hirashima, T., Matsuda, N., Nomoto, T., Toyodda, J. "*Context-Sensitive Filtering for Browsing in Hypertext*". In Proceedings of International Conference on Intelligent User Interfaces, IUI'98. (San Francisco, CA, USA). Pages 21-28. 1998.

[Hoque, 1999] Hoque, R. "*CORBA for Real Programmers*". Morgan Kaufmann, 1999.

[IEEE, 1995] IEEE. "*IEEE Standard for Information Technology – Software Reuse – Data Model for Reuse Library Interoperability Data Model (BIDM)*". IEEE Std 1420.1, 1995.

[ISO, 1996] ISO. "*The Open-EDI Reference Model*". IS 14662, ISO/IEC JTC1/SC30, International Standards Organization, 1996.

[Jacobson et al., 1997] Jacobson I., Griss M., Jonsson P. "*Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison-Wesley Longman, 1997.

[Jensen, 1997a] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jensen, 1997b] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jensen, 1997c] Jensen, K. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3*". 2nd edition. Springer Verlag, 1996. Second corrected printing 1997.

[Jones, 1984] Jones, T. C. "*Reusability in Programming: A Survey of the State of the Art*". IEEE Transactions on Software Engineering, 10(5):488-494. September 1984.

[Jörding, 1999a] Jörding, T. "*Adaptive Shopping in the Web: Individual Product Presentations for Every Customer*". In Proceedings of HCI International '99 (Munich, Germany, 1999). 1999.

[Jörding, 1999b] Jörding, T. "*Temporary User Modeling for Adaptive Product Presentations in the Web*". In Proceedings of the Seventh International Conference on User Modeling (Banff, Canada. 1999). 1999.

[Kang, 1998] Kang, K. C. "*Feature-Oriented Development of Applications for a Domain*". In Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, (June 2-5, 1998, Victoria, B.C., Canada). Pages 354-355. IEEE-CS, 1998.

[Kang et al., 1990] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S. "*Feature-Oriented Domain Analysis (FODA). Feasibility Study*". Technical Report CMU/SEI-90-TR21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213. 1990.

[Kang et al., 1998] Kang, K. C., Kim, S., Lee, J., Kim, K. "*FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*". Annals of Software Engineering, 5:143-168. 1998.

[Karlsson, 1995] E.-A. Karlsson (Editor). "*Software Reuse. A Holistic Approach*". John Wiley & Sons Ltd., 1995.

[Kavraki, 1995] Kavraki, L. E. "*Computation of Configuration-Space Obstacles Using the Fast Fourier Transform*". IEEE Transactions on Robotics and Automation, 11(3):408-413, 1995.

[Kienan, 2000] Kienan, B. "*Small Business Solutions E-Commerce*". McGraw-Hill, 2000.

[Klingler y Solderitsch, 1996] Klingler, C. D., Solderitsch, J. "*DAGAR: A Process for Domain Architecture Definition and Asset Implementation*". In Proceedings of the Annual International Conference on ADA (TriAda'96), (December 3-7, 1996, Philadelphia, PA, USA). Pages 231-245. ACM Press, 1996.

[Knauber y Succi, 2001] Knauber, P., Succi, G. "*Perspectives on Software Product Lines*". ACM Software Engineering Notes, 26(2):29-33. March 2001.

[Krueger, 1992] Krueger, C. W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. 1992

[Lang, 1995] Lang, K. "*Newsweeder: Learning to Filter Netnews*". In proceedings of the Machine Learning Conference. Morgan Kaufman, San Francisco, 1995.

[Laplante, 1998] Laplante, M. "*Making EDI Accessible with XML*". EC.COM 4(2):23-26. March 1998.

[Latombe, 1991] Latombe, J. C. "*Robot Motion Planning*". Kluwer Academic Publishers, Boston, MA, 1991.

[Lee et al., 2000] Lee, K., Kang, K. C., Chae, W., Choi, B. W. "*Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse*". Software: Practice and Experience, 30(9):1025-1046. 2000.

[Leebaert, 1998] Leebaert, D. (Editor). "*The Future of the Electronic Marketplace*". The MIT Press, 1998.

[Lores, 2001] Lores, J. (Editor). "*La Interacción Persona-Ordenador*". . http://griho.udl.es/ipo/libroe.html. Diciembre, 2001.

[Lozano-Pérez, 1983] Lozano-Pérez, T. "*Spatial Planning: A Configuration Space Approach*". IEEE Transactions on Computers, 32:108-120, 1983.

[Ma, 1999] Ma, M. (Guest Editor). "*Agents in E-Commerce*". Communications of the ACM, 42(3):79-80. March 1999.

[Maes, 1994] Maes, P. "*Agents that Reduce Work and Information Overload*". Communications of the ACM, 37(7):31-40,146. July 1994.

[Maes et al., 1999] Maes, P., Guttman, R. H., Moukas, A. "*Agents that Buy and Sell*". Communications of the ACM, 42(3):81-91. March 1999.

[Malone et al., 1987] Malone, T. W., Yates, J., Benjamin, R. I. "*Electronic Markets and Electronic Hierarchies*". Communications of the ACM, 30(6):484-497. June 1987.

[Marathe y Diwakar, 2001] Marathe, M., Diwakar, H. "*The Architecture of a One-Stop Web-Window Shop*". SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-commerce, 2(1):11-18. Winter 2001.

[MarketScience, 2000] MarketScience. "*Integration Is the Key to Marketplace Success*". White Paper. November 2000.

[Matos et al., 1998] Matos, N., Sierra, C., Jennings, N. "*Determining Successful Negotiation Strategies: An Evolutionary Approach*". In Proceedings of the Third International Conference on Multi-Agent Systems ICMA'98. Pages 182-189. 1998.

[Matos y Sierra, 1999] Matos, N., Sierra, C. "*Evolutionary Computing and Negotiation Agents*". In *Agent Mediated Electronic Commerce*. Noriega, P., Sierra, C. (Editors). Lecture Notes in Computer Science. VOL. 1571. Pages 126-150. Springer-Verlag, 1999.

[McConnell, 1997] McConnell, S. (Editor). "*The OMG/CommerceNet Joint Electronic Commerce*". Whitepaper, EC/97-06-09. Object Management Group, 1997.

[McIlroy, 1976] McIlroy, M. D. "*Mass-produced Software Components*". In J. M. Buxton, P. Naur, B. Randell (eds.) *Software Engineering Concepts and Techniques* (1968 NATO Conference on Software Engineering). Pages 88-98. Van Nostrand Reinhold, 1976.

[Menczer y Belew, 1998] Menczer, F., Belew, R. K. "*Adaptive Information Agents in Distributed Textual Environments*". In Proceedings of the Second International Conference on Autonomous Agents – AGENTS'98. (Minneapolis, MN, USA, 1998). Pages 157-164. ACM Press, 1998.

[MENHIR, 1998] Grupo MENHIR. "*MENHIR (Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos)*". En las actas de las III Jornadas de Investigación y Docencia en Bases de Datos. Editores J. Carlos Casamayor, M. Celma, L. Mota, Mª Á. Pastor (Valencia, 23 de marzo de 1998). Páginas 11-41. 1998.

[Microsoft, 1996] Microsoft Corporation. "*DCOM Technical Overview*". White Paper, Microsoft Developer Network, 1996.

[Mili et al., 1995] Mili, H., Mili, F., Mili, A. "*Reusing Software: Issues and Research Directions*". IEEE Transactions on Software Engineering. 21(6):528-562. June 1995.

[Mili et al., 1998] Mili, A., Mili, R., Mittermeir, R. T. "*A Survey of Software Reuse Libraries*". Annals of Software Engineering, 5:349-414. 1998.

[Milosavljevic, 1997] Milosavljevic, M. "*Augmenting the User's Knowledge Via Comparison*". In Proceedings of 6th International Conference on User Modeling – UM97. Jameson, A., Paris, C., Tasso, C. (Editors). Pages 119-130. Springer, 1997.

[Milosavljevic y Oberlander, 1998] Milosavljevic, M., Oberlander, J. "*Dynamic Hypertext Catalogues: Helping Users to Help Themselves*". In Proceedings of Ninth ACM International Hypertext Conference – Hypertext'98. (Pittsburgh, USA, 1998). Pages 123-131. 1998.

[Mowbray y Zahavi, 1995] Mowbray, T. J., Zahavi, R. "*The Essential CORBA. Systems Integration Using Distributed Objects*". John Wiley & Sons, 1995.

[NATO, 1992] NATO. "*NATO Standard for Management of a Reusable Software Component Library*". Volume 2 (of 3 Documents). NATO Communications and Information Systems Agency (NACISA). 1992.

[Neches, 1996] Neches, A.-L. "*The Future of Electronic Commerce: A Pragmatic View*". ACM Computing Surveys, 28(4es). December 1996.

[NHSE, 1997] NHSE. "*The Internal Data Format of RIB*". NHSE. October 24, 1997.

[Neighbors, 1984] Neighbors, J. M. "*The Draco Approach to Constructing Software from Reusable Components*". IEEE Transactions on Software Engineering, SE-10(5):564-574. 1984.

[Not et al., 1998] Not, E., Petrelli, D., Sarini, M., Stock, O., Strapparava, C., Zancanaro, M. "*Hypernavigation in the Physical Space: Adapting Presentation to the User and to the Situational Context*". New Review of Multimedia and Hypermedia 4, 33-45. 1998.

[Olsson y Piani, 1992] Olsson, G., Piani, G. "*Computer Systems for Automation and Control*". Herts. Prentice Hall, 1992.

[Omelayenko, 2001] Omelayenko, B. "*Integration of Product Ontologies for B2B Marketplaces: A Preview*". SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-commerce 2(1):19-25. Winter 2001.

[OMG, 2001] Object Management Group. "*CORBA 2.6 Specification*". Document formal/01-02-35. 2001.

[Orfali y Harkey, 1997] Orfali, R., Harkey, D. "*Client/Server Programming with Java and CORBA*". John Wiley & Sons, 1997.

[Ostertag et al., 1992] Ostertag, E., Hendler, J., Prieto-Díaz, R., Braun, C. "*Computing Similarity in a Reuse Library System: An AI-based Approach*". ACM Transactions on Software Engineering and Methodology, 1(3):205-228. July 1992.

[Papaioannou y Edwards, 1998] Papaioannou, T., Edwards, J. "*Mobile Agent Technology in Support of Sales Order Processing in the Virtual Enterprise*". In Proceedings of the 3rd IEEE/IFIP Int'l Conference on Information Technology for Balanced Automation Systems in Manufacturing. (Prague, Czech Republic, 1998). Pages 275-288. Kluwer Academic Publishers, 1998.

[Papazoglou, 2001] Papazoglou, M. P. "*Agent-Oriented Technology in Support of E-Business*". Communications of the ACM, 44(4):71-77. April 2001.

[Paternò y Mancini, 1999] Paternò, F., Mancini, C. "*Designing Web User Interfaces Adaptable to Different Types of Use*". In Proceedings of Museums and the Web. (New Orleans, LA, USA). 1999.

[Pazzani y Billsus, 1997] Pazzani, M. J., Billsus, D. "*Learning and Revising User Profiles: The Identification of Interesting Web Sites*". Machine Learning, 27:313-331. 1997.

[Pazzani y Billsus, 1999] Pazzani, M. J., Billsus, D. "*Adaptive Web Site Agents*". In proceedings of the Third International Conference on Autonomous Agents – AGENTS'99. (Seattle, WA, USA, 1999). Pages 394-395. ACM Press, 1999.

[Perkowitz y Etzioni, 1998] Perkowitz, M., Etzioni, O. "*Adaptive Web Sites: Automatically Synthesizing Web Pages*". In Proceedings of AAAI'98. 1998.

[Peterson, 1977] Peterson, J. "*Petri Nets*". ACM Computing Surveys 9(3):223-252, 1977.

[Peterson, 1981] Peterson, J. "*Petri Net Theory and the Modeling of Systems*". Prentice-Hall, 1981.

[Pitt y Mamdani, 1999a] Pitt, J., Mamdani, A. "*Designing Agent Communication Languages for Multi-Agent Systems*". In *Multi-Agent System Engineering MAAMAW'99*, Garijo, F., Boman, M. (Editor). Volume LNAI1647. Pages 102-114. Springer-Verlag, 1999.

[Pitt y Mamdani, 1999b] Pitt, J., Mamdani, A. "*A Protocol-Based Semantics for an Agent Communication Language*". In Proceedings 16th IJCAI'99. Pages 485-491. Morgan-Kaufmann, 1999.

[Pitt et al., 2000] Pitt, J., Guerin, F., Stergiou, C. "*Protocols and Intentional Specifications of Multi-Party Agent Conversations for Brokerage and Auctions*". In Proceedings of Agents 2000. (Barcelona, Spain, 2000). Pages 269-276. ACM Press, 2000.

[Prieto-Díaz, 1989] Prieto-Díaz, R. "*Classification of Reusables Modules*". In Biggerstaff, T. J., Perlis, A. J. (Editors), *Software Reusability. Concepts and Models.* Volume I of Frontier Series. ACM Press, Addison-Wesley, New York, 1989.

[Rayport y Sviokla, 1995] Rayport, J. F., Sviokla, J. J. "*Exploiting the Virtual Value Chain*". Harvard Business Review, 73(6):75-85. November-December 1995.

[Sarkar et al., 1995] Sarkar, M. B., Butler, B., Steinfield, C. "*Intermediaries and Cybermediaries: A Continuing Role for Mediating Players in the Electronic Marketplace*". Journal of Computer-Mediated Communications, 1(3). December 1995.

[Segev et al., 1995] Segev, A., Wan, D., Beam, C. "*Designing Electronic Catalogs for Business Value: Results from the CommerceNet Pilot*". CITM Working Paper WP-95-1005, Haas School of Business, University of California, Berkeley, 1995.

[Segev et al., 1999] Segev, A., Gebauer, J., Färber, F. "*Internet-based Electronic Markets*". EM – International Journal of Electronic Markets, 9(3). September 1999.

[SEI, 2001] Software Engineering Institute. "*A Framework for Software Product Line – Version 3.0*". Product Line Systems Program, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). http://www.sei.cmu.edu/plp/framework.html. 2001.

[Sema, 1996] Sema Group. "*EUROWARE User's Manual*". Sema Group, 1996.

[Sen y Weiss, 1999] Sen, S., Weiss, G. "*Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*"**.** In *Multiagent Systems.* Weiss, G. (Editor). Pages. 259-298. The MIT Press, 1999.

[SER, 1996] SER Consortium. "*Solutions for Software Evolution and Reuse*". SER ESPRIT Project 9809. January 1996.

[Sheth, 1994] Sheth, B. "*A Learning Approach to Personalized Information Filtering*". SM Thesis. Department of Electrical Engineering and Computer Science. MIT. February 1994.

[Sheth y Maes, 1993] Sheth, B., Maes, P. "*Evolving Agents for Personalized Information Filtering*". In Proceedings of the Ninth Conference on Artificial Intelligence for Applications. IEEE-CS, 1993.

[Siegel, 1996] Siegel, J. "*CORBA. Fundamentals and Programming*". John Wiley & Sons, 1996.

[Siegel, 2000] Siegel, J. (Editor). "*CORBA 3 Fundamentals and Programming*". 2nd edition. Jonh Wiley & Sons, 2000.

[Silva, 1985] Silva, M. "*Las Redes de Petri: En la Automática y la Informática*". Editorial AC, libros científicos y técnicos. Madrid. 1985.

[Simos, 1995] Simos, M. "*Organization Domain Modeling (ODM): Formalizating the Core Domain Modeling Life Cycle*". Symposium on Software Reusability. April 1995.

[Simos et al., 1996] Simos, M., Creps, D., Klingler, C., Levine, L., Allemang, D. "*Organization Domain Modeling (ODM) Guidebook – Version 2.0*". Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121. 1996.

[Slonim y Bennet, 1996] Slonim, J., Bennet, K. "*The Electronic Commerce Position Paper*". ACM Computing Surveys, 28(4es). December 1996.

[Smith y Poulter, 1999] Smith, H., Poulter, K. "*Share the Ontology in XML-based Trading Architectures*". Communications of the ACM, 42(3):110-111. March 1999.

[Sohn y Yoo, 1998] Sohn, S., Yoo, K. J. "*An Architecture of Electronic Market Applying Mobile Agent Technology*". In Proceedings of the 4th IEEE Symposium on Computers and Communications. (Athens, Greece, 1998). Pages 359-364. IEEE-CS Press, 1998.

[Sonnemann, 1995] Sonnemann, R. "*Exploratory Study of Software Reuse Factors*". PhD Thesis, George Mason University, Fairfax, Virginia, Spring 1995.

[SUN, 2002] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 12/04/2002]. March 2002.

[Sycara et al., 1996] Sycara, K., Pannu, A., Williamson, M., Zeng, D., Decker, K. "*Distributed Intelligent Agents*". IEEE Expert, 11(6):36-46. December 1996.

[Szyperski, 1998] Szyperski, C. "*Component Software – Beyond Object-Oriented Programming*". Addison-Wesley, 1998.

[Tennenhouse et al., 1996] Tennenhouse, D. L., Garland, S. L., Shrira, L., Kaahoek, M. F. "*From Internet to ActiveNet*". RFC. http://www.tns.lcs.mit.edu. January 1996.

[Tennenhouse et al., 1997] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., Minden, G. J. "*A Survey of Active Network Research*". IEEE Communications Magazine, 35(1):80-86. January 1997.

[Tennenhouse y Wetherall, 1996] Tennenhouse, D. L., Wetherall, D. J. "*Towards an Active Network Architecture*". Computer Communication Review, 26(2), April 1996.

[Textor, 1999] Textor Webmasters Ltd. "*An Electronic Commerce Primer*". Version 2.2. August 1999.

[Therón et al., 1999] Therón, R., Blanco, F. J., Curto, B., Moreno, V. "*Implementación Paralela de Algoritmos para la Evaluación Rápida del Espacio de las Configuraciones de un Robot*". Actas de las XX Jornadas de Automática, 1999.

[Tong y Swartztrauber, 1991] Tong, C., Swartztrauber, P. "*Orderer FFTs on a Massively Parallel Hypercube Multiprocessor*". Parallel Computing, 50-59, 1991.

[Trump, 1997] Trump, D. "*Using the WWW and the Internet to Support Corporate Reuse*". In Proceedings of the 8th Annual Workshop on Software Reuse, WISR-8. (March 23-26, 1997. The Ohio State University, Columbus, Ohio, USA). 1997.

[Tsvetovat et al., 2000] Tsvetovat, M., Sycara, K., Chen, Y., Ying, J. "*Customer Coalitions in the Electronic Marketplace*". In Proceedings of the 3rd Workshop on Agent Mediated Electronic Commerce – AMEC-2000. 2000.

[Tucker et al., 1990] Tucker, A. B. (Editor and Co-chair), Barnes, B. H. (Co-chair), Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*Computing Curricula 1991. Report of the ACM/IEEE-CS Joint Curriculum Task Force*". ACM. http://www.computer.org/education/cc1991/. [Última vez visitado, 27-12-2001]. December 1990.

[USAL, 1997] Universidad de Salamanca. "*Normativa Universitaria 1997*". Secretaría General de la Universidad de Salamanca. Enero, 1997.

[Vázquez, 1999] Vázquez, E. "*Estudio de Situación del Comercio Electrónico en España*" http://www.dit.upm.es/doc/comercioe/index.html. [Última vez visitado, 31-1-2002]. 1999.

[Vici y Argentieri, 1998] Vici, A. D., Argentieri, N. "FODAcom: An Experience with Domain Analysis in Italian Telecom Industry". In Proceedings of the Fifth International Conference on Software Reuse, ICSR-5, (June 2-5, 1998, Victoria, B.C., Canada). Pages 166-175. IEEE-CS, 1998.

[Vidal y Durfee, 1998] Vidal, J., Durfee, E. "*The Moving Target Function Problem in Multi-Agent Learning*". In proceedings of the Third International Conference on Multi-Agent Systems ICMA'98. Pages317-324, 1998.

[Vigna, 1998] Vigna, G. (Editor). "*Mobile Agents and Security*". Lecture Notes of Computer Science 1419(12). Springer Verlag, 1998.

[Wallnau, 1992] Wallnau, K. C. "*Towards an Extended View of Reuse Libraries*". In Proceedings of 5th Workshop on Institutionalizing Software Reuse (WISR-5), Palo Alto, California (USA). 1992.

[Weiß y Dillenbourg, 1999] Weiß, G., Dillenbourg, P. "*What Is 'Multi' in Multi-Agent Learning?*". Chapter 4 in *Collaborative Learning. Cognitive and Computational Approaches*. P. Dillenbourg (Editor). Pages 64-80. Pergamon Press, 1999.

[Weiss, 2001] Weiss, M. "*Patterns for e-Commerce Agent Architectures: Using Agents as Delegates*". In proceedings of Pattern Languages of Programs 2001 - PLoP 2001. 2001.

[Whinston et al., 1997] Whinston, A. B., Stahl, D. O., Choi, S. Y. "*The Economics of Electronic Commerce*". Macmillan, 1997.

[Williamson, 1985] Williamson, O. E. "*The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting*". New York, 1985.

[Wirfs-Brock y Johnson, 1990] Wirfs-Brock, R. J., Johnson, R. E. "*Surveying Current Research in Object-Oriented Design*". Communications of the ACM, 33(9):105-124. 1990.

[Wong et al., 1999] Wong, D., Paciorek, N., Moore, D. "*Java-based Mobile Agents*". Communications of the ACM, 42(3):92-102. March 1999.

[Wooldridge y Jennings, 1995] Wooldridge, M., Jennings, N. R. "*Intelligent Agents: Theory and Practice*". The Knowledge Engineering Review, 10(2):115-152. 1995.

[Yamamoto y Sycara, 2001] Yamamoto, J., Sycara, K. "*A Stable and Efficient Buyer Coalition Formation Scheme for E-Marketplaces*". In Proceedings of AGENTS'01 (Montréal, Quebec, Canada, May 28-June 1, 2001). Pages 576-583. ACM Press, 2001.

[Yee, 1999] Yee, B. "*A Sanctuary for Mobile Agents, Secure Internet Programming*". Lecture Notes of Computer Science, 1603(10):261-274. 1999.

Apéndice A:
Plan de Calidad de la Unidad Docente de IS y OO

Se incluye íntegramente la versión 1.1 del Plan de Calidad de la Unidad Docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca para el bienio 1999-2000, cuya última modificación data del 20 de marzo de 2000 (por lo tanto existen algunas diferencias con respecto a las asignaturas presentadas en el presente Proyecto Docente e Investigador). Este plan de calidad ha sido elaborado por el Dr. Francisco José García Peñalvo, la Dra. María N. Moreno García, el Dr. José Rafael García-Bermejo Giner y Ana de Luis Reboredo.

A.1 Introducción

Hay muchas definiciones de calidad en la bibliografía, según la norma ISO 8402 (admitida como norma española UNE 66-001-92), la calidad se define como: "*Totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas*".

El concepto de calidad es un objetivo fundamental para los directivos de una empresa junto a los dos parámetros clásicos de su gestión: el dinero y el tiempo. El mercado actual es

sumamente competitivo, donde no basta con producir masivamente los productos o servicios, vender es lo importante y sólo se produce un producto cuando se tiene la seguridad de aceptación por parte del cliente. Así, conseguir la satisfacción del cliente y conocer sus necesidades, para luego definirlas en forma de requisitos a cumplir, se convierte en una necesidad imperiosa para las empresas que pretendan hacerse con un hueco en un determinado mercado.

En la vida cotidiana, la calidad representa las propiedades inherentes a un objeto, de forma que pueda ser comparado con otros objetos de su especie para determinar si es mejor, igual o peor. Calidad es sinónimo de bondad, excelencia o superioridad.

Para la comunidad universitaria, la calidad debe ser un objetivo tan importante como para la empresa. El profesor de Universidad debe ver en la calidad un camino hacia la excelencia en todas sus actividades: docencia, investigación y gestión.

En lo referente a la actividad investigadora, la calidad debe ser la guía para aumentar el conocimiento del investigador, y debe estar presente en todos los productos resultantes de la investigación.

La actividad docente por su parte puede asociarse más a una actividad contractual donde la Universidad es la empresa y los clientes son varios. El cliente más directo es el alumno, el producto que se le ofrece debe ser un currículo que cumpla unos requisitos acordes a los objetivos marcados por la titulación elegida. La sociedad es el cliente indirecto de la Universidad, donde el producto que se le ofrece son los alumnos formados y preparados para introducirse en el mundo real y rendir acorde a las necesidades de esa sociedad. Por último, el propio docente será a la vez mecanismo y cliente de la Universidad, al buscar por un lado la excelencia de conocimiento en una determinada materia y por otro lado la satisfacción personal conseguida cuando los alumnos salen cumpliendo los requisitos de conocimientos que la Universidad marca y la adecuación que la sociedad demanda.

Entre los cursos académicos 96-97 y 98-99 se han desarrollado una serie de actividades destinadas a la mejora continua en el ámbito de la docencia en los campos de la Ingeniería del Software y de la Orientación a Objetos, las cuales se han visto plasmadas en un conjunto de planes de calidad que se centraban en recoger los objetivos y los criterios de evaluación de los mismos para asignaturas concretas, cayendo la responsabilidad de poner en marcha estos planes en la persona que acometía esta tarea [García et al., 1999a].

Como complemento a esta actividad de carácter meramente individual, se estableció un mecanismo de evaluación externa, que permitiese al responsable de la misma obtener una mayor

información contrastada con la que poder realimentar el proceso de mejora continua [García et al., 1999b].

Sin embargo, estas iniciativas individuales no han influido de una forma clara en la obtención de una mejora global dentro de un colectivo (área, departamento, titulación...) al estar situadas en los límites de una sola asignatura.

Por este motivo, se va acometer la tarea de la realización de un plan de calidad para el bienio 1999-2001, pero no centrado en una única asignatura, sino que abarque una unidad docente completa, la unidad docente de Ingeniería del Software y Orientación a Objetos, que tiene competencias en dos titulaciones universitarias diferentes, aunque relacionadas: la *Ingeniería Técnica en Informática de Sistemas (Plan de 1997)* y la *Ingeniería Informática*, ambas impartidas en la Universidad de Salamanca.

El plan de calidad se estructura como sigue: en la sección dos se presenta y se justifica la unidad docente de Ingeniería del Software y Orientación a Objetos, especificando los objetivos perseguidos por esta unidad docente, las asignaturas que la forman, indicando sus interrelaciones, y estableciendo a groso modo como se reparten los objetivos de la unidad docente en las asignaturas. La tercera sección se centra en detallar de forma concreta los objetivos y las aportaciones de cada una de las asignaturas, con la misión específica, en esta primera aproximación para el bienio 1999-2001, de minimizar solapamientos de contenidos y de conocer explícitamente los contenidos y limites del resto de las asignaturas que conforman la unidad docente.

A.2 Unidad docente de Ingeniería del Software y Orientación a Objetos

La Ingeniería del Software es la parte de la disciplina informática que se ocupa de ofrecer una aproximación sistemática a la creación y mantenimiento de los sistemas software, ofreciendo un enfoque que rehuye las concepciones tradicionales, por las que el software se concibe como un producto artesano inmerso en un ambiente *mitológico*, casi *místico*, donde los sistemas software distan mucho de las necesidades reales del tejido social y empresarial actual.

En la aproximación buscada por la Ingeniería del Software se concibe al *ingeniero informático* como aquella persona que es capaz de aplicar sus conocimientos para la resolución de los problemas reales derivados de la concepción, explotación y mantenimiento de sistemas informáticos, donde la perspectiva no es la de pequeños sistemas, que pueden ser

responsabilidad de un único individuo, sino de los sistemas que requieren para su realización el trabajo conjunto y coordinado de diferentes personas.

En España, actualmente, los estudios universitarios de informática se organizan en ingenierías técnicas e ingenierías superiores, siendo una política acorde con los que defienden que la informática es una ingeniería [Buxton et al., 1976; Shaw y Tomayko, 1991; Leveson, 1997; Parnas, 1997]. Así, el cuerpo de conocimientos de un ingeniero informático debe estar formado por una sólida base en Lógica, Matemáticas y Ciencia de la Computación, aspectos propios de Ingeniería del Software y un conjunto de temas de carácter universal que completarán su currículo (economía, idiomas...).

La unidad docente de Ingeniería del Software y Orientación a Objetos del Departamento de Informática y Automática de la Universidad de Salamanca, se encarga de impartir la docencia relacionada con la Ingeniería del Software en las titulaciones que se discuten.

La inclusión de la orientación a objetos como parte de la unidad docente se debe a la gran repercusión de los métodos orientados a objetos en el desarrollo de sistemas software, lo cual lleva a hablar de una Ingeniería del Software Orientada a Objetos que, por las propias características intrínsecas de la tecnología de objetos, abarca con un mayor énfasis todas las fases del ciclo de vida del software.

A.2.1 Objetivos de la unidad docente

Los objetivos de la unidad docente se pueden clasificar en tres grandes apartados:

- Conceptos teóricos
- Aspectos prácticos
- Habilidades personales

A.2.1.1 Conceptos teóricos

T1 Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.

T2 Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

T5 Especificaciones formales de requisitos.

T6 Método de análisis/diseño estructurado.

T7 Método de análisis/diseño orientado a objetos.

T8 Diseño de la interfaz gráfica de usuario.

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

T10 Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

T11 Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

T13 Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

A.2.1.2 Aspectos prácticos

P1 Aplicación de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicación de forma práctica los conceptos teóricos de Orientación a Objetos.

P3 Aplicación de forma práctica los conceptos teóricos sobre gestión de proyectos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

P5 Programación orientada a objetos.

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

P7 Aprendizaje y manejo de forma práctica de plataformas, entornos de desarrollo, lenguajes de programación... de alta repercusión en el desarrollo de sistemas software en la actualidad.

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

P9 Construcción de sistemas software de entidad superior a una práctica de laboratorio, a ser posible partiendo de unas especificaciones reales elicitadas a *clientes* y/o *usuarios* reales.

A.2.1.3 Habilidades personales

H1 Mejora de la expresión oral.

H2 Mejora en la redacción de documentos técnicos.

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

H4 Capacitación de los alumnos para el trabajo en grupo.

A.2.2 Asignaturas de la unidad docente

La unidad docente de Ingeniería del Software y Orientación a Objetos se ha creado para dar cobertura a las titulaciones de Ingeniería Técnica en Informática de Sistemas (I.T.I.S) (Plan de 1997) e Ingeniero en Informática (I.I) (Plan de 1998) impartidas en la Universidad de Salamanca.

La primera de ellas es una titulación de primer ciclo (de tres cursos de duración), mientras que la segunda es una titulación de segundo ciclo (de dos cursos de duración), que tiene como condición de acceso la posesión del título de Ingeniero Técnico en Informática de Gestión o Ingeniero Técnico en Informática de Sistemas.

En estas titulaciones las asignaturas que mejor se ajustan a los objetivos marcados en la unidad docente se recogen en la Tabla A.1.

Asignatura	Titulación	Curso	Carácter	Créditos
Interfaces Gráficas	I.T.I.S	2º	Optativa	6 (3T + 3P)
Ingeniería del Software	I.T.I.S	3º	Obligatoria	6 (4,5T + 1,5P)
Programación Orientada a Objetos	I.T.I.S	3º	Optativa	6 (3T + 3P)
Proyecto	I.T.I.S	3º	Obligatoria	9 (9P)
Análisis de Sistemas	I.I	1º	Troncal	9 (6T + 3P)
Administración de Proyectos Informáticos	I.I	2º	Troncal	9 (6T + 3P)
Sistemas de Información	I.I	2º	Troncal	9 (9P)
Proyecto	I.I	2º	Troncal	6 (6P)

Tabla A.1. Asignaturas que componen la unidad docente

Las dependencias e interrelaciones entre estas asignaturas se muestran en la Figura A.1. En el establecimiento de estas dependencias se ha tenido en cuenta tanto el factor tiempo, que claramente establece el orden lógico en el que se van a cursar las asignaturas, como las aportaciones a la unidad docente de las mismas en forma de contenidos y objetivos a conseguir.

Además de las asignaturas propias de la unidad docente, se incluye la asignatura de **Diseño de Bases de Datos**. Esto se debe a que es la asignatura en la que se introducen los modelos de datos conceptuales y lógicos (diagramas entidad-interrelación y modelos relacionales típicamente), lo que supone una importante base, a la vez que una descarga, para la asignatura de Ingeniería del Software donde estos modelos serán utilizados de forma práctica sin necesidad de tener que incluirlos en la parte teórica de la asignatura.

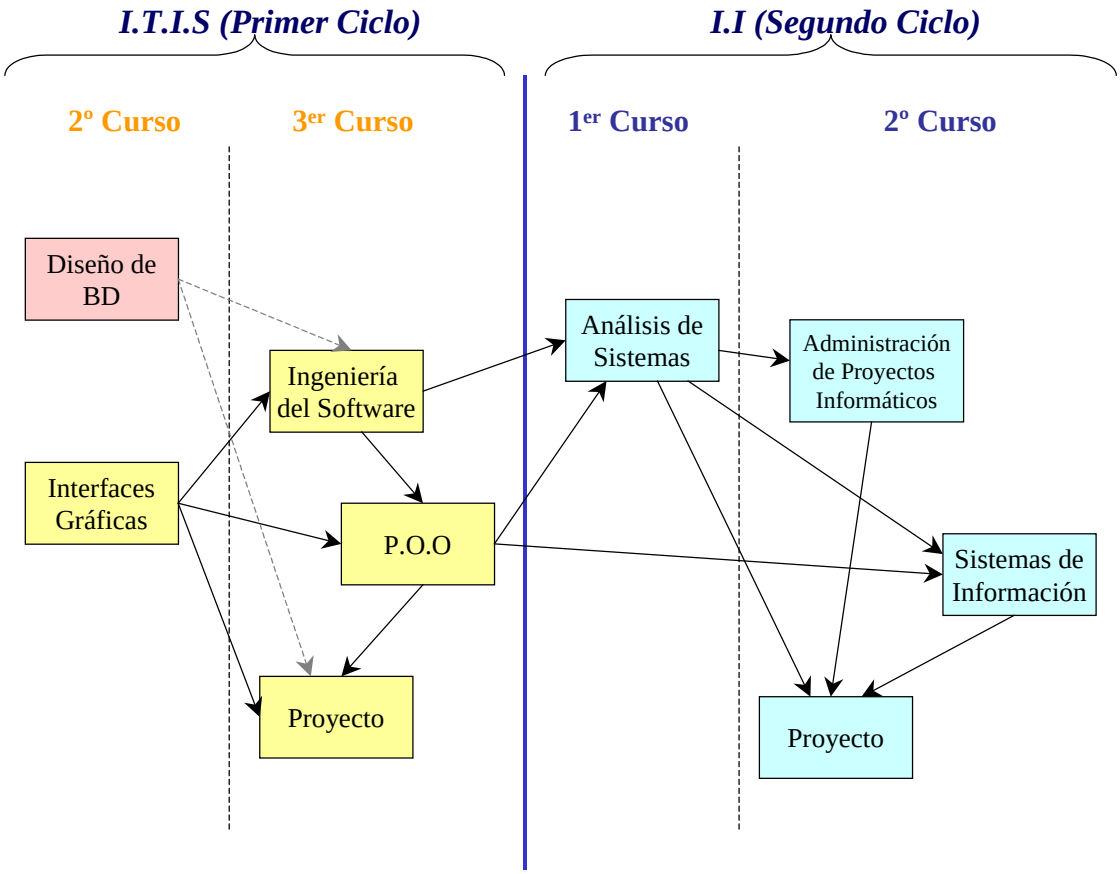


Figura A.1. Dependencias entre las asignaturas que forman la unidad docente

A.2.3 Reparto de los objetivos en las asignaturas de la unidad docente

	Obj. Teóricos	Obj. Prácticos	Hab. Personales
Interfaces Gráficas	T8	P6	H1, H2, H3
Ingeniería del Software	T1, T2, T3, T4, T6, T7, T9	P1, P2, P4	H1, H2, H3, H4
Programación Orientada a Objetos	T7, T9	P2, P5	H1, H2, H3, H4
Proyecto I.T.I.S		P9	H1, H2, H3
Análisis de Sistemas	T3, T4, T5, T7, T12, T13	P1, P2, P4	H1, H2, H3, H4
Administración de Proyectos Informáticos	T10, T11, T12	P4, P8	H1, H2, H3, H4
Sistemas de Información		P7	H1, H2, H3
Proyecto I.I		P8, P9	H1, H2, H3

Tabla A. 2. Reparto de los requisitos en las asignaturas de la unidad docente

A.3 Estudio de los objetivos de cada una de las asignaturas

A.3.1 Interfaces gráficas

El objetivo de esta asignatura es introducir a los alumnos de I.T.I.S en algunos sistemas de desarrollo de interfaces gráficas de frecuente uso en la actualidad. Se trata pues de un objetivo fundamentalmente práctico, ya que se busca la capacitación del alumno para implementar aplicaciones que incorporen los elementos habituales en las interfaces gráficas de usuario actuales.

Este objetivo se corresponde con los objetivos **T8, P6, H1, H2** y **H3** de la unidad docente.

A continuación, se detallan los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.1.1 Ficha de la asignatura

Asignatura	*Interfaces gráficas (Optativa)*
Créditos	*3T + 3P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*2º*
Cuatrimestre	*2º*
Responsable	*Ana de Luis Reboredo (adeluis@usal.es)*
Página web de la asignatura	

Tabla A.3. Interfaces gráficas

A.3.1.2 Prerrequisitos

Es imprescindible que el alumno esté capacitado para el desarrollo de programas en lenguaje C. Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

Sería muy conveniente que el alumno ya dispusiese de ciertos conocimientos de programación orientada a objetos, que le facilitaran el acceso a ciertas herramientas de creación de interfaces gráficas actuales. No obstante, con la actual distribución temporal de asignaturas,

esto no es muy frecuente. Por ello, en la asignatura se proporcionarán unos conocimientos básicos de programación orientada a objetos y, en concreto, del lenguaje C++ que permitan a los alumnos utilizar dichas herramientas.

A.3.1.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: La GUI de Windows. Programación SDK (11 Horas)
Tema 1. Introducción a la programación en Windows (1 Hora)
Tema 2. GDI. Fundamentos gráficos (2 Horas)
Tema 3. Eventos de entrada. Ratón y teclado (1 Hora)
Tema 4. Controles. Controles predefinidos (2 Horas)
Tema 5. Recursos (1 Hora)
Tema 6. Menús (1 Hora)
Tema 7. Cuadros de diálogo (2 Horas)
Tema 8. Intercambio de datos (1 Hora)
Unidad Docente II: Visual C++ (8 Horas)
Tema 8. Introducción a la programación orientada a objetos (2 Horas)
Tema 9. La biblioteca MFC (6 Horas)
Unidad Docente III: El entorno X Window (10 Horas)
Tema 10. Introducción. Arquitectura del sistema (2 Horas)
Tema 11. Conceptos básicos. Terminología (2 Horas)
Tema 12. Ventanas (2 Horas)
Tema 13. Entradas. El ratón y el teclado (2 Horas)
Tema 14. Gráficos (2 Horas)

Tabla A.4. Programa de teoría de Interfaces Gráficas (3 créditos)

Prácticas (30 Horas)
Realización de aplicaciones mediante programación SDK (12 Horas)
Realización de aplicaciones con Visual C++ (8 Horas)
Realización de aplicaciones con X Window (10 Horas)
Práctica obligatoria
Realización de una aplicación completa que incorpore la mayoría de los elementos estudiados.

Tabla A.5. Programa de prácticas de Interfaces Gráficas (3 créditos)

A.3.1.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir.

T8 Diseño de la interfaz gráfica de usuario.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación de los diferentes elementos que componen una GUI y sus distintos modos de implementación	Clases Teóricas	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P6 Realización de interfaces gráficas de usuario en diferentes plataformas.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de sucesivas aplicaciones que incorporen elementos de interfaces gráficas para Windows y X-Window	Clases prácticas	Responsable de la asignatura Alumnos	Aula de Informática Transparencias Bibliografía básica	Defensa de la práctica obligatoria Examen

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Ordenador	Defensa de la práctica

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Documentación de la práctica obligatoria	Defensa de la práctica	Grupo de alumnos Responsable de la asignatura	Bibliografía	Defensa de la práctica

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca Recursos de Internet	Examen

A.3.1.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (50% de la nota final).

- Un examen final en el mes de Junio.

- Un examen final en el mes de Septiembre.

b) Parte Práctica (50% de la nota final)

- Desarrollo de una aplicación con una interfaz gráfica de usuario que incluya la mayoría de los elementos estudiados. Se realizará en grupos de un máximo de tres miembros.

Se realizará una defensa por grupos de dicho trabajo.

Tras la defensa, los diferentes miembros del grupo deberán responder a las preguntas que se les planteen de forma individualizada sobre el desarrollo de la aplicación y las posibles modificaciones de la misma.

La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Cada una de las partes se deberá aprobar por separado exigiéndose un mínimo de 5 puntos en cada una.

A.3.1.6 Bibliografía básica de referencia

🕮 **Petzold, C.** "*Programación en Windows 95*". McGraw-Hill, 1996.

Es la traducción del libro "*Programming Windows 95*" que a su vez es la continuación del clásico "*Programming Windows 3.0*" del mismo autor. Es el libro básico para la introducción en la programación SDK, imprescindible para comprender la programación en Windows. Explica detalladamente el funcionamiento de la API de Windows desarrollando de modo progresivo los distintos aspectos relacionados con la misma. Cuenta con multitud de ejemplos en C que están disponibles en CD. Se utilizará para todos los temas de la Unidad Docente I de la asignatura.

🕮 **Reiss, L., Radin, J.** "*Aplique X Window*". McGraw-Hill, 1993.

Se trata de la traducción del libro de los mismos autores "*X Window Inside & Out*". Parte de una completa introducción al sistema y sus componentes para pasar a describir las funciones X Window ilustrándolas con programas completamente desarrollados. Será el libro que se utilizará en la Unidad Docente III de la asignatura.

🕮 **Yao, P., Leinecker, R. C.** "*Todo Visual C++ 5* ". INFORBOOK'S, 1998.

Es la traducción del libro "*Visual C++ 5 Bible*" de los mismos autores. El contenido está dividido en 4 partes. La primera parte contiene una introducción a la programación en Windows en la que hace algunas consideraciones sobre la programación con la API de Microsoft Windows. A continuación, incluye una pequeña guía de uso de la herramienta de desarrollo Visual C++. La segunda parte es una introducción a la programación orientada a objetos y a las ampliaciones del lenguaje C++ sobre el lenguaje C. La tercera parte trata todos los aspectos relacionados con la librería MFC. La última parte incluye algunos temas de interés como, por ejemplo algunos formatos de archivos de imágenes o la gestión de memoria de Windows. Todo el libro está ilustrado con abundantes ejemplos que están disponibles en un CD adjunto.

A.3.2 Ingeniería del Software

El objetivo principal de esta asignatura es introducir a los alumnos de I.T.I.S en los principios fundamentales, métodos y herramientas para el desarrollo sistemático de proyectos informáticos. Adicionalmente, los alumnos deberán aplicar éstas trabajando en equipo.

Este objetivo se corresponde con los objetivos **T1, T2, T3, T4, T6, T7, T9, P1, P2, P4, H1, H2, H3** y **H4** de la unidad docente.

Para ver satisfacer estos objetivos, se va a detallar los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.2.1 Ficha de la asignatura

Asignatura	*Ingeniería del software (obligatoria)*
Créditos	*4,5T + 1,5P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*1º*
Responsable	*Francisco José García Peñalvo (fgarcia@usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla A.6. Ingeniería del Software

A.3.2.2 Prerrequisitos

El alumno debe estar familiarizado con la teoría así como con la práctica del diseño y codificación en lenguajes procedurales (por ejemplo C). Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

El alumno debe tener los conocimientos y la práctica necesaria en la creación de modelos de información conceptuales y lógicos, en concreto, dominio del modelo entidad-relación, paso al modelo relacional y normalización. Estos conceptos se adquieren en la asignatura de segundo curso **Diseño de Bases de Datos** (asignatura troncal en el Plan de Estudios vigente).

Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

A.3.2.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (6 Horas)
Tema 1. Introducción a la Ingeniería del Software (6 Horas)
Unidad Docente II: Paradigma estructurado de desarrollo (23 Horas)
Tema 2. Análisis y especificación de requisitos (11 Horas)
Tema 3. Análisis estructurado (2 Horas)
Tema 4. Diseño del software (6 Horas)
Tema 5. Diseño estructurado (4 Horas)
Unidad Docente III: Introducción al paradigma objetual (13 Horas)
Tema 6. Introducción a la Orientación a Objetos (6 Horas)
Tema 7. UML (6 Horas)
Tema 8. Visión general de la metodología OMT (1 Hora)
Unidad Docente IV: Miscelánea (2 Horas)
Tema 9. Herramientas CASE (2 Horas)

Tabla A.7. Programa teórico de Ingeniería del Software (4,5 créditos)

Talleres (10 Horas)
Práctica 1. Taller de modelado de datos. Repaso al modelo entidad-interrelación (2 Horas)
Práctica 2. Taller de modelado funcional (Enfoque Clásico)(2 Horas)
Práctica 3. Taller de modelado funcional (Yourdon) (4 Horas)
Práctica 4. Taller de orientación al objeto (2 Horas)
Laboratorio (2 Horas)
Práctica 5. Manejo de una herramienta CASE (2 Horas)
Práctica obligatoria
Práctica 6. Realización de una ERS y un prototipo de una aplicación software

Tabla A.8. Programa práctico de Ingeniería del Software (1,5 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.9 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T1, T2, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T9, H3
Tema 5	T6, H3
Tema 6	T3, T7, H3
Tema 7	T7, H3
Tema 8	T7, H3
Tema 9	P4, H3
Práctica 1	P1, H1, H2, H4
Práctica 2	P1, H1, H2, H4
Práctica 3	P1, H1, H2, H4
Práctica 4	P2, H1, H2, H4
Práctica 5	P4
Práctica 6	P1, P2, P4, H1, H2, H4

Tabla A.9. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

La Unidad Docente IV: Miscelánea, es difícil de impartir en la realidad por limitaciones de tiempo. El Tema 9, dedicado a las herramientas CASE, puede introducirse a la vez que se realiza la Práctica 4.

A.3.2.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir. Cada línea de acción va a definirse mediante los cuatro apartados indicados en [García et al., 1999a]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T1 Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.

T2 Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación de la problemática del desarrollo de programas con calidad industrial, poniendo de manifiesto la necesidad de emplear técnicas de ingeniería	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Reuniones con los alumnos Examen
Presentación de ejemplos de empresas que hagan uso de métodos de Ingeniería del Software bien implantados	A lo largo de la asignatura Conferencias impartidas por parte de representantes del mundo empresarial	Ponentes invitados	Medios audiovisuales Contactos en la empresa	Opinión de los alumnos Opinión de otros profesores del Departamento
Estudio y comprensión de los diferentes paradigmas de ciclo de vida de un desarrollo software	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Presentación del papel protagonista de los requisitos y de las especificaciones dentro de un desarrollo software	Durante toda la asignatura, especialmente en la Unidad II, Unidad III y en la práctica de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Reuniones con los alumnos Examen
Distinción entre elicitación, documentación y especificación de requisitos	Unidad docente II. Se habla de requisitos de cliente y de desarrollador	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T6 Método de análisis/diseño estructurado.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas y modelos para la especificación de requisitos y diseño en el paradigma estructurado	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Introducción de las extensiones de notación funcional para tiempo real	Unidad docente I	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método estructurado para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T7 Método de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas y modelos para la especificación de requisitos y diseño en el paradigma objetual	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Ver los fundamentos de diseño como la base necesaria para obtener sistemas software de calidad, con independencia del paradigma de desarrollo empleado	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Talleres prácticos Defensa de la práctica obligatoria Examen

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de talleres en parte de las horas prácticas de la asignatura donde los alumnos se organizan en grupos para resolver un problema utilizando técnicas estructuradas y orientadas a objeto. Posteriormente uno de los grupos expondrá su solución y se debatirá sobre ella	Parte práctica de la asignatura	Grupos de alumnos Responsable de la asignatura	Enunciados con casos prácticos Transparencias	Informes realizados por los alumnos Comentarios de los alumnos
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Introducción a las herramientas CASE	Unidad docente IV Clase práctica	Responsable de la asignatura Alumnos	Transparencias Bibliografía Herramientas CASE	Defensa de la práctica
Creación de un documento de ERS de un caso real, partiendo de unas entrevistas a los interesados (los alumnos pueden elegir el método objetual para especificar los requisitos del desarrollador)	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Promover los debates sobre las soluciones presentadas en los talleres prácticos	Clases prácticas	Responsable de la asignatura Grupos de alumnos	Pizarra Medios audiovisuales	Estudio del seguimiento de los talleres Opinión alumnos
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para esos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de un informe técnico por cada uno de los talleres prácticos realizados. Estos informes quedarán publicados en la página web de la asignatura	En las 3 semanas siguientes a una clase práctica	Responsable de la asignatura Grupos de alumnos	Página web de la asignatura	Nota *extra* para estos trabajos voluntarios
Documentación de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para esos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
La práctica obligatoria debe ser realizada en grupos de trabajo de 3 a 5 personas	Práctica obligatoria de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados pos los alumnos	Defensa de la práctica
Realización de talleres en parte de las horas prácticas de la asignatura donde los alumnos se organizan en grupos para resolver un problema utilizando técnicas estructuradas y orientadas a objeto. Posteriormente uno de los grupos expondrá su solución y se debatirá sobre ella	Parte práctica de la asignatura	Grupos de alumnos Responsable de la asignatura	Enunciados con casos prácticos Transparencias	Informes realizados por los alumnos Comentarios de los alumnos

A.3.2.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (50% de la nota final)

- Un examen final en el mes de febrero.

- Un examen final en el mes de septiembre.

b) Parte Práctica (50% de la nota final)

- ERS y prototipo realizado en grupos de trabajo.

Se realizará una defensa individualizada de dicho trabajo.

La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Entre 0,5 y 0,75 puntos sumados a la nota de la parte práctica por la defensa y posterior redacción de un informe sobre los supuestos tratados en los talleres prácticos.

d) Trabajos voluntarios presentados (de 0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida)

e) Fórmula para la obtención de la calificación final.

Si (Teoría ≥ 4,75) **y** (Práctica ≥ 5.0)

Nota Final = (Teoría*0,5) + ((Práctica+Nota Talleres)*0,5) + Nota trabajos

Sino

Ø

Fin si

A.3.2.6 Bibliografía básica de referencia

En este apartado se va a incluir aquellas referencias que el alumno debe consultar para completar el material que el responsable facilita para seguir las clases (transparencias). Estos títulos han sido elegidos en función de tres factores: *su adecuación a los contenidos*, *su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

Existen otras referencias que se han manejado por el responsable de la asignatura para preparar los temas, pero éstas, al ser más específicas, no son incluidas en este apartado, aunque aparecen citadas en las transparencias y son recomendadas como otras lecturas en muchas ocasiones.

🕮 **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

Este libro es la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. De la trilogía de libros publicados por los tres máximos responsables del lenguaje modelado UML, éste es el que se corresponde con la explicación detallada de la notación de este lenguaje de modelado. Es un libro de consulta obligada para el tema 6, aunque su primera parte de introducción, especialmente el capítulo 1, se puede utilizar en el primer tema de la parte teórica.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Corresponde a la traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997. Esta segunda edición es más que una simple actualización de la primera edición, con una ampliación de conceptos más que considerable. Aunque es un libro que se adecua más a una asignatura de programación orientada a objetos, los contenidos de los tres primeros capítulos se adaptan muy bien a la asignatura de Ingeniería del Software, especialmente el Capítulo 3 dedicado a la modularidad, que es una referencia importante para el Tema 4.

🕮 **Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Libro de introducción a la Ingeniería del Software, centrado en la construcción de software de gestión. Consta de tres partes: Sistemas de Información (capítulos 1 y 2), El Proceso de Desarrollo de Software (capítulos 3-16) y Tecnología (capítulos 17-19).

Es un libro de consulta que se puede utilizar en todos los temas de la parte teórica de la asignatura, aunque con una mayor profusión en los temas 1, 2, 4 y 5.

🕮 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill, 1998.

Es uno de los mejores libros de referencia y consulta para la práctica totalidad de los temas relacionados con la Ingeniería del Software. Corresponde a la traducción del libro del mismo autor "*Software Engineering: A Practitioner´s Approach*", 4th Edition, McGraw-Hill, 1997. En este texto se realiza una amplia exposición de prácticamente todos los temas relacionados con la Ingeniería del Software desde la doble perspectiva del producto y el proceso. Es de destacar el conjunto de referencias tanto bibliográficas como de recursos disponibles en Internet de cada uno de los temas tratados. Se considera un texto imprescindible de referencia para esta asignatura. En este sentido hay que señalar que en el prólogo de esta edición española se indica que explica todos los temas incluidos en la asignatura o materia troncal Ingeniería del Software de los Planes de Estudios de las universidades españolas.

En lo que respecta a la asignatura, se adecua como una referencia especialmente interesante en los temas 1, 2, 4, 5, 7 y 8.

🕮 **Rational Software Corporation.** "*The Unified Modeling Languague Documentation Set 1.1*". http://www.rational.com. September 1997.

Esta documentación corresponde al texto completo de la propuesta de UML realizada por Rational al OMG. Esta documentación consta de un metamodelo formal, una notación y su semántica. En el metamodelo se realiza una exposición de los conceptos e ideas, de la sintaxis y de la semántica de UML. En la notación se describen los diferentes tipos de diagramas: de casos de uso, de clases, de secuencias, de colaboración, de estados, de actividad, de componentes y de despliegue. En el documento de semántica se establece la semántica de los elementos de UML utilizando un subconjunto de la notación. Esta documentación es básica para el seguimiento de la Unidad Didáctica III de la asignatura.

🕮 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". 2ª Reimpresión. Prentice Hall, 1998.

Corresponde a la traducción del libro de los mismos autores "*Object Oriented Modeling and Design*", Prentice Hall, 1991. En el libro se presenta una de las primeras metodologías orientadas al objeto que fueron desarrolladas, OMT. En esta metodología de desarrollo de sistemas se considera que el modelado orientado al objeto consta de tres partes: el modelo objeto, el modelo funcional y el modelo dinámico. A pesar de ser una metodología superada en muchos aspectos, desde el punto de vista de la docencia a impartir tiene como ventaja que toma algunas de las herramientas de las metodologías clásicas, por ejemplo el DFD para el elaborar el modelo funcional, lo que permite realizar una transición suave desde los métodos clásicos a los de la Orientación al Objeto. La tercera parte del libro aborda los aspectos de paso del diseño a la implementación en relación con los lenguajes de programación y las bases de datos relacionales. Incluye la descripción de tres casos de estudio. Este libro se utiliza principalmente como referencia para los temas 1 y 6.

🕮 **Sommerville, I.** "*Software Engineering*". 5th Edition, Addison-Wesley, 1996.

Esta es la última edición de uno de los libros más difundidos sobre Ingeniería del Software. Explica un amplio conjunto de técnicas y muestra como pueden ser aplicadas en la práctica de la Ingeniería del Software. Los temas están agrupados de acuerdo a las actividades del proceso de desarrollo y mantenimiento del software. Respecto a las versiones anteriores hay que destacar la incorporación de temas como re-ingeniería o la ampliación de temas como CASE y gestión y evolución del software. El autor, profesor de la asignatura de Ingeniería del Software en una universidad inglesa, se ha preocupado de adaptar los contenidos a la recomendación curricular ACM/IEEE-CS91 para los módulos SE2 a SE5. Además, puede obtenerse material suplementario de apoyo a la docencia en Internet. El nivel de algunos de los capítulos es superior al exigido a los alumnos de I.T.I.S, pero se considera uno de los libros imprescindibles de consulta en la materia.

🕮 **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993.

Corresponde a la traducción del texto del mismo autor "*Modern Structured Analysis*", Prentice-Hall, 1989. En él, Yourdon revisa las teorías del análisis estructurado desarrolladas por DeMarco en el año 1979. Es un libro con un diseño muy didáctico y adaptado a un curso de análisis estructurado. Tiene algunos capítulos, que en su momento constituyeron un cierto avance en las metodologías estructuradas y que, a pesar del paso de los años, mantienen su vigencia. Es de destacar el capítulo en el que se realiza un estudio general de sistemas, la propuesta de los diferentes modelos a realizar de un sistema y los casos prácticos desarrollados en los apéndices. Sigue constituyendo uno de los libros de referencia obligada para los cursos de Ingeniería del Software. En esta asignatura es una referencia complementaria en los temas 1 y 2, siendo la referencia fundamental del tema 3.

A.3.3 Programación Orientada a Objetos

El principal objetivo de esta asignatura es transmitir al alumno los conocimientos necesarios para el diseño e implementación de aplicaciones software bajo el paradigma objetual.

Aunque se va a utilizar un lenguaje orientado a objetos (C++) para que el alumno pueda llevar al terreno práctico los conceptos transmitidos en esta asignatura, se pretende una exposición de los mismos desde un plano más abstracto, más centrado en el diseño, de forma que el alumno pueda aplicarlos en su vida laboral con independencia del entorno de desarrollo orientado a objetos con el que se encuentre.

Este objetivo general se corresponde con los objetivos concretos **T7, T9, P2, P5, H1, H2, H3** y **H4** de la unidad docente.

Para ver satisfacer dichos objetivos, se va a detallar los prerrequisitos de esta asignatura, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta recomendada.

A.3.3.1 Ficha de la asignatura

Esta asignatura se considera una extensión de la asignatura Ingeniería del Software, en la que se amplía la Unidad Docente III de ésta con aspectos de diseño y programación orientada objetos. De hecho, la asignatura de Programación Orientada a Objetos se imparte en el segundo cuatrimestre del tercer curso de I.T.I.S, después de la finalización de la asignatura de Ingeniería del Software.

Asignatura	*Programación orientada a objetos (optativa)*
Créditos	*3T + 3P*
Estudios	*I.T.I.S*
Plan	*B.O.E de 4-11-1997*
Curso	*3º*
Cuatrimestre	*2º*
Responsable	*Francisco José García Peñalvo (fgarcia@usal.es)*
Página web de la asignatura	*http://tejo.usal.es/~fgarcia/docencia.html*

Tabla A.10. Programación Orientada a Objetos

A.3.3.2 Prerrequisitos

El alumno debe estar familiarizado con la teoría así como con la práctica del diseño y codificación en lenguajes procedurales (por ejemplo C). Estos conocimientos deben adquirirse en las asignaturas relacionadas con la programación en el primer y segundo curso **Algoritmia, Programación, Laboratorio de Programación** y **Estructuras de Datos.**

Es deseable que el alumno conozca la problemática de las interfaces de usuario y esté familiarizado con la problemática de construir interfaces gráficas de usuario. Estos aspectos deben tratarse en la asignatura **Interfaces Gráficas** del segundo curso.

El alumno debe conocer los fundamentos básicos que rigen un proyecto software, estar familiarizado con la elicitación y especificación de requisitos y manejar los principios de la Ingeniería del Software orientada a objetos, en concreto el lenguaje de modelado UML y conocimientos generales de una metodología orientada a objetos del tipo OMT. Todos estos conceptos le son transmitidos al alumno en la asignatura de tercer curso **Ingeniería del Software**.

A.3.3.3 Temario teórico/práctico

Presentación de la asignatura (1 Hora)
Unidad Docente I: Conceptos básicos (4 Horas)
Tema 1. Lenguajes de programación orientados a objetos (2 Horas)
Tema 2. Orientación a objeto y reutilización del software (2 Horas)
Unidad Docente II: Diseño orientado a objetos básico (14 Horas)
Tema 3.Técnicas básicas de diseño orientado a objetos (8 Horas)
Tema 4. Genericidad (4 Horas)
Tema 5. Manejo de excepciones (2 Horas)
Unidad Docente III: Diseño orientado a objetos avanzado (11 Horas)
Tema 6. Patrones de diseño (8 Horas)
Tema 7. Diseño por contrato (3 Horas)

Tabla A.11. Programa de teoría de Programación Orientada a Objetos (3 créditos)

C++ (18 Horas)
Tema C++ 1. Modelo objeto en C++ (10 Horas)
Tema C++ 2. STL (8 Horas)
Taller (2 Horas)
Tarjetas CRC
Prácticas libres (10 Horas)
Práctica obligatoria
Realización de una aplicación utilizando técnicas orientadas a objeto

Tabla A.12. Programa de prácticas de Programación Orientada a Objetos (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.13 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T7
Tema 2	T7
Tema 3	T7, T9, P2
Tema 4	T7, T9, P2
Tema 5	T7, T9, P2
Tema 6	T7, T9, P2
Tema 7	T7, T9, P2
Tema C++ 1	P5
Tema C++ 2	P5
Taller CRC	P2, H1, H2, H3, H4
Práctica obligatoria	P2, P5, H1, H2, H3, H4

Tabla A.13. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.3.4 Líneas de acción

Para cada uno de los objetivos de la unidad docente que debe satisfacerse en esta asignatura se van a identificar las líneas de acción a seguir. Cada línea de acción va a definirse mediante los

cuatro apartados indicados en [García et al., 1999a]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T7 Método de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Desarrollo de aplicaciones bajo el paradigma objetual, centrando la atención en las fases de diseño e implementación	Clases teóricas y prácticas	Responsable de la asignatura	Bibliografía Transparencias Lenguaje de programación orientado a objetos	Examen teórico Prácticas Opinión de los alumnos

T9 Estudio y comprensión de los fundamentos del diseño de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Aplicación de principios y técnicas del diseño orientado a objetos	Parte teórica	Responsable de la asignatura	Transparencias Bibliografía básica	Examen Práctica obligatoria
Introducción de técnicas avanzadas de diseño orientado a objetos	Parte teórica	Responsable de la asignatura	Transparencias Bibliografía básica	Examen Práctica obligatoria

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de una práctica obligatoria en la que se cree una aplicación utilizando técnicas orientadas a objeto	Parte práctica de la asignatura	Grupos de alumnos	Caso práctico	Defensa de la práctica

P5 Programación orientada a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de un lenguaje orientado a objetos	Clases prácticas	Responsable de la asignatura Alumnos	Bibliografía Pequeños supuestos prácticos	Defensa de la práctica
Realización de una práctica obligatoria en la que se cree una aplicación utilizando técnicas orientadas a objeto	Parte práctica de la asignatura	Grupos de alumnos	Caso práctico	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa oral de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Documentación de la práctica obligatoria	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
La práctica obligatoria debe ser realizada en grupos de trabajo de 2 ó 3 personas	Práctica obligatoria de la asignatura	Alumnos Responsable de la asignatura	Supuestos prácticos	Defensa de la práctica

A.3.3.5 Criterios de evaluación de la asignatura

a) Parte de Teoría (*50% de la nota final*)

- Un examen final.

b) Parte Práctica (*50% de la nota final*)

- Realización de un supuesto práctico por parejas.

Se realizará una defensa de dicho trabajo.

La parte práctica se guardará hasta la convocatoria de septiembre, pero nunca para futuros cursos.

c) Trabajos voluntarios presentados (de *0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida*)

d) Fórmula para la obtención de la calificación final.

Si (Teoría ≥ 4,75) **y** (Práctica ≥ 5.0)

Nota Final = (Teoría*0,5) + (Práctica*0,5) + Nota trabajos

Sino

Ø

Fin si

A.3.3.6 Bibliografía básica de referencia

En este apartado se va a incluir aquellas referencias que el alumno debe consultar para completar el material que el responsable facilita para seguir las clases (transparencias). Estos títulos han sido elegidos en función de tres factores: *su adecuación a los contenidos, su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

Existen otras referencias que se han manejado por el responsable de la asignatura para preparar los temas, pero éstas, al ser más específicas, no son incluidas en este apartado, aunque aparecen citadas en las transparencias y son recomendadas como otras lecturas en muchas ocasiones.

📖 **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

Este libro es la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. De la trilogía de libros publicados por los tres máximos responsables del lenguaje modelado UML, éste es el que se corresponde con la explicación detallada de la notación de este lenguaje de modelado.

Es un libro de consulta para repasar o aclarar cualquier duda que pudiera surgir sobre UML a lo largo del desarrollo de la asignatura.

📖 **Deitel, H. M., Deitel, P. J.** "*Como Programar en C/C++*". 2ª Ed. Prentice Hall, 1995.

Traducción del libro de los mismos autores "*C How to Program*", 2nd Edition, Prentice-Hall, 1994. Es un libro con dos partes bien diferenciadas, los primeros catorce capítulos están dedicados al lenguaje C, mientras que los siete últimos a C++. Es por tanto un libro de referencia general que, además de tratar temas de C++, sirve como consulta para posibles dudas sobre lenguaje C.

🕮 **Gamma, E., Helm, R., Johnson, R., Vlissides, J.** "*Design Patterns. Elements of Reusable Object-Oriented Software*". Addison-Wesley, 1995.

Este libro es un clásico del diseño orientado a objeto y el máximo responsable de la difusión de los patrones de diseño en la comunidad de la Orientación a Objetos. El libro es un catálogo de 23 patrones de diseño divididos en tres categorías: patrones de creación, patrones estructurales y patrones de comportamiento.

Es la referencia más importantes de la parte teórica de la asignatura, así como el texto base del tema 5.

🕮 **Glass, G., Schuchert, B.** "*The STL <Primer>*". Prentice-Hall, 1996.

Libro que hace un recorrido muy detallado por la biblioteca de plantillas estándar de C++ **STL** (*Standard Template Library*). El libro está dividido en cinco secciones: *Presentación de STL, Utilizando STL, Catálogo de las clases de STL, Catálogo de Algoritmos* y *Apéndices.*

Este libro sirve como referencia complementaria en el tema 3 de la parte teórica dedicado a la genericidad, y como referencia base en el tema 2 de prácticas dedicado a la biblioteca STL.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Corresponde a la traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997. Esta segunda edición es más que una simple actualización de la primera edición, con una ampliación de conceptos más que considerable. Es un libro sobre el cual se podía construir una asignatura de programación orientada a objetos, especialmente si elige Eiffel como lenguaje de programación para la parte práctica.

En la asignatura de Programación Orientada a Objetos va a ser una referencia de consulta más que una guía de los contenidos, aunque es la referencia clave en el tema 6.

🕮 **Rational Software Corporation.** "*The Unified Modeling Languague Documentation Set 1.1*". http://www.rational.com. September 1997.

Esta documentación corresponde al texto completo de la propuesta de UML realizada por Rational al OMG. Esta documentación consta de un metamodelo formal, una notación y su semántica. En el metamodelo se realiza una exposición de los conceptos e ideas, de la sintaxis y de la semántica de UML. En la notación se describen los diferentes tipos de diagramas: de casos de uso, de clases, de secuencias, de colaboración, de estados, de actividad, de componentes y de despliegue. En el documento de semántica se establece la semántica de los elementos de UML utilizando un subconjunto de la notación.

Aunque el estudio de UML no es un objetivo de esta asignatura, sino de la asignatura de Ingeniería del Software, va a ser el lenguaje de modelado que se utilice para representar los diseños a discutir en la asignatura.

🕮 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.

Corresponde a la traducción del libro de los mismos autores "*Object Oriented Modeling and Design*", Prentice Hall, 1991. En el libro se presenta una de las primeras metodologías orientadas al objeto que fueron desarrolladas, OMT. En esta metodología de desarrollo de sistemas se considera que el modelado orientado al objeto consta de tres partes: el modelo objeto, el modelo funcional y el modelo dinámico. A pesar de ser una metodología superada en muchos aspectos, desde el punto de vista de la docencia a impartir tiene como ventaja que toma algunas de las herramientas de las metodologías clásicas, por ejemplo el DFD para el elaborar el modelo funcional, lo que permite realizar una transición suave desde los métodos clásicos a los de la Orientación al Objeto. La tercera parte del libro aborda los aspectos de paso del diseño a la implementación en relación con los lenguajes de programación y las bases de datos relacionales. Incluye la descripción de tres casos de estudio.

Este libro constituye una referencia de consulta válida en todos los temas de la parte teórica de la asignatura, especialmente por su excelente exposición del modelo objeto de OMT.

🕮 **Stroustrup, B.** "*El Lenguaje de Programación C++*". 3ª Ed. Addison-Wesley. 1998.

Este libro es la traducción de la obra del mismo autor "*The C++ Programming Language*", 3rd Edition, Addison-Wesley, 1997. Es el libro de referencia por excelencia cuando de C++ se trata, al ser el autor del mismo el padre de este lenguaje de programación orientado a objetos. En el momento de escribir esta tercera edición el proceso de estandarización de C++ se encontraba en su recta final, no previéndose cambios significativos, por lo que se puede considerar que el libro contiene una presentación de C++ actualizada y validada por el comité de estandarización de este lenguaje. El libro se divide en 6 partes: *Introducción* (capítulos 1-3), *Tipos Predefinidos en C++* (capítulos 4-9), *Programación Orientada a Objetos y Genérica con C++* (capítulos 10-15), *Biblioteca C++ Estándar* (capítulos 16-22), *Diseño y Desarrollo de Software* (capítulos 23-25) y *Apéndices*.

Es un libro que se ajusta perfectamente a la práctica de la asignatura, en especial las tres primeras partes.

A.3.4 Proyecto I.T.I.S.

La realización del proyecto de final de carrera se convierte en una de las primeras oportunidades con las que cuenta un estudiante de una Ingeniería Técnica en Informática para aplicar de forma integrada los conocimientos teóricos y prácticos obtenidos en el conjunto de asignaturas cursadas en la carrera.

El objetivo principal de éste casa con los objetivos **P9, H1, H2** y **H3** de la unidad docente (en los casos en que el proyecto se lleve a cabo por más de una persona tendría sentido incluir el objetivo **H4**).

En un proyecto de final de carrera el cometido y la bibliografía será establecida por el tutor o tutores del mismo, aunque como guía de referencia general para la realización y documentación de los proyectos final de carrera se tiene [García et al., 2000], guía que se ajusta especialmente a los proyectos que tienen como cometido la creación de un sistema software.

A.3.5 Análisis de Sistemas

El propósito fundamental de esta asignatura es que los alumnos del segundo ciclo de Ingeniería en Informática profundicen en los conocimientos adquiridos en la asignatura *Ingeniería del Software* de primer ciclo, especialmente en lo que se refiere a métodos de análisis y especificación de requisitos de sistemas orientados a objetos y a técnicas formales de especificación.

Los objetivos de la unidad docente en que se desglosa son los siguientes:

T3, T4, T5, T6, T7, T12, T13, P1, P2, P4, H1, H2, H3 y **H4**.

Seguidamente se expondrán los prerrequisitos de esta asignatura, necesarios para conseguir los objetivos mencionados, las líneas de acción a seguir en cada uno de ellos, el temario teórico/práctico de la asignatura, los criterios de evaluación y la bibliografía básica de consulta que se recomienda a los alumnos.

A.3.5.1 Ficha de la asignatura

Asignatura	*Análisis de Sistemas (troncal)*
Créditos	*6T + 3P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*1º*
Cuatrimestre	*1º y 2º (anual)*
Responsable	*María N. Moreno García (mmg@usal.es)*
Página web de la asignatura	*http://lisisu02.usal.es/~mmoreno/analisis.html*

Tabla A.14. Análisis de Sistemas

A.3.5.2 Prerrequisitos

Además de los prerrequisitos que se establecen en la asignatura Ingeniería del Software, de esta unidad docente, se deben tener conocimientos teóricos y prácticos de los objetivos de la Ingeniería del Software, de los modelos de ciclo de vida del software más utilizados y de los

métodos de análisis y diseño estructurado. Asimismo sería conveniente conocer las bases del desarrollo de software orientado a objetos.

Las asignaturas que tratan estos aspectos son **Ingeniería del Software y Programación Orientada a Objetos**, que se imparten en 3º de ITIS.

A.3.5.3 Temario teórico/práctico

Presentación de la asignatura (1 hora)
Unidad Docente I: Definición del proceso de software (7 horas)
Tema 1. Modelos avanzados de ciclo de vida (3 Horas)
Tema 2. Prototipos (4 Horas)
Unidad Docente II: Métodos de desarrollo (36 horas)
Tema 3. Métodos estructurados. Técnicas avanzadas (6 Horas)
Tema 4. Métodos orientados a objetos (10 Horas)
Tema 5. El lenguaje UML y el proceso unificado (10 Horas)
Tema 6. Técnicas formales de especificación (10 Horas)
Unidad Docente III: Desarrollo de sistemas especiales (12 horas)
Tema 7. Sistemas de tiempo real (6 horas)
Tema 8. Otros sistemas (6 horas)
Unidad Docente IV: Evolución del software (4 horas)
Tema 9. Evolución y mantenimiento del software (4 horas)

Tabla A.15. Programa de teoría de Análisis de Sistemas (6 créditos)

Laboratorio: uso de herramientas CASE (30 horas)
Práctica 1. Análisis y Diseño estructurado de un sistema (15 horas)
Práctica 2. Análisis y Diseño orientado a objetos (15 horas)

Tabla A.16. Programa de prácticas de Análisis de Sistemas (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas
- *Workshop* de trabajos realizados por los alumnos sobre temas de Ingeniería del Software y objetos

En la Tabla A.17 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T3,T4, H3
Tema 2	T3, T4, H3
Tema 3	T6, H3
Tema 4	T7, H3
Tema 5	T7, H3
Tema 6	T5, H3
Tema 7	T13, H3
Tema 8	T13, H3
Tema 9	T12, H3
Práctica 1	P1, H1, H2, H4
Práctica 2	P2, H1, H2, H4

Tabla A.17. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.5.4 Líneas de acción

En este apartado se comentarán las líneas de acción a seguir para la consecución de cada uno de los objetivos de la unidad docente. Cada línea de acción va a definirse mediante los cuatro apartados indicados en [García et al., 1999a]: *cuándo se lleva a cabo, quién es el responsable de realizarla, qué medios son necesarios* y *cómo se evaluarán los resultados de dicha línea de acción.*

T3 Importancia de los requisitos en el ciclo de vida del software.

T4 Elicitación, documentación, especificación y prototipado de los requisitos de un sistema software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Mostrar la relevancia de los requisitos al estudiar modelos avanzados de ciclo de vida.	En el tema 1 y en las prácticas de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Estudiar el prototipado de sistemas como modelo de ciclo de vida y como técnica de verificación y validación de requisitos	En el tema 2 y en las prácticas de la asignatura	Responsable de la asignatura	Transparencias Bibliografía básica	Defensa práctica Examen
Realización del análisis y especificación de requisitos de un sistema haciendo uso de herramientas CASE.	Prácticas de la asignatura	Clientes Alumnos Responsable	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T5 Especificaciones formales de requisitos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las principales técnicas formales de especificación de sistemas, incluyendo la especificación de aspectos dinámicos y los métodos formales orientados a objetos	En el tema 6	Responsable de la asignatura	Transparencias Bibliografía básica	Ejercicios prácticos Examen

T7 Métodos de análisis/diseño orientado a objetos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Profundización en los métodos orientados a objetos. Estudio del Lenguaje Unificado de Modelado y del Proceso Unificado	Unidad docente II	Responsable de la asignatura	Transparencias Bibliografía básica	Ejercicios prácticos Defensa de la práctica Examen
Realización del análisis y diseño de un sistema orientado a objetos con la ayuda de una herramienta CASE	Práctica 2 de la asignatura	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudiar los procesos relacionados con la modificación de un producto software después de su entrega.	Tema 9	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

T13 Conocimiento sobre el uso de la Ingeniería del Software en dominios de aplicación específicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Estudio de las particularidades que presenta el desarrollo de sistemas de tiempo real y de las técnicas más apropiadas para el modelado de este tipo de sistemas	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Análisis de las características de los sistemas distribuidos y su relación con la aplicación de métodos de Ingeniería del Software.	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Examen de otros sistemas no convencionales como sistemas expertos, sistemas para el soporte a las decisiones, etc.	Unidad docente III	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P1 Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.

P2 Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.

P4 Utilización de herramientas CASE para la gestión y desarrollo de sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Con la ayuda de herramientas CASE, los alumnos realizarán el análisis y diseño de sendos sistemas de software aplicando, respectivamente, un método estructurado y otro orientado a objetos.	Clases prácticas	Responsable de la asignatura Alumnos Clientes	Transparencias Bibliografía Herramientas CASE Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Promover los debates sobre las soluciones presentadas en los ejercicios realizados en clase	Clase de teoría	Responsable de la asignatura Grupos de alumnos	Pizarra Medios audiovisuales	Estudio del seguimiento de los ejercicios Opinión alumnos
Defensa oral de las prácticas	Defensa de la práctica	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Propuesta de ejercicios de aplicación de los conceptos teóricos	Durante las clases de teoría	Responsable de la asignatura Grupos de alumnos	Transparencias Pizarra	Evaluación de los ejercicios entregados
Documentación de las prácticas	Defensa de las prácticas	Grupos de alumnos Responsable de la asignatura	Medios audiovisuales	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumno para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de las prácticas en grupos de 2 personas	Durante las clases prácticas	Clientes Alumnos Responsable de la asignatura	Clientes que se presten a ser entrevistados por los alumnos	Defensa de la práctica

A.3.5.5 Criterios de evaluación de la asignatura

a) Parte de Teoría

- Un examen final en el mes de junio.

- Un examen final en el mes de septiembre.

b) Parte Práctica

- En junio la evaluación de esta parte de la asignatura se realizará mediante la valoración de los trabajos presentados. La nota será la media de las obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individualizada.

- En septiembre se realizará un examen en el que el alumno tendrá que resolver un supuesto práctico con la ayuda de una herramienta CASE.

c) Trabajos voluntarios presentados (de 0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida).

d) En el caso de haber superado la parte teórica y la práctica, se aplicará la siguiente fórmula para la obtención de la calificación final:

Nota Final = ((Teoría*0,6) + (Práctica*0,3))* 10/9 + Nota trabajos

A.3.5.6 Bibliografía básica de referencia

En el apartado de bibliografía se recogen aquellas referencias que pueden ayudar al alumno a completar los conocimientos impartidos en las clases. La elección de las mismas, como ocurre en otras asignaturas de la unidad, se ha basado en tres aspectos: *su adecuación a los contenidos, su disponibilidad en la biblioteca* y *el idioma (primando, si fuera posible títulos en español).*

- **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

  El texto en castellano corresponde a la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. Es uno de los libros de la trilogía publicada por los creadores del lenguaje modelado UML, en él se proporciona una referencia de uso de las características específicas de UML, centrándose fundamentalmente la notación gráfica. Es indispensable su consulta en el tema 4 y fundamentalmente en el 5.

- **Graham, I.** "*Métodos Orientados a Objetos*". 2ª Ed. Addison-Wesley/Díaz de Santos, 1996.

  Traducción del libro del mismo autor "*Object-Oriented Methods*" 2nd ed. Addison-Wesley, 1994. En un libro introductorio a la Orientación al Objeto. Cubre de forma amplia y correcta todos los aspectos de la Orientación al Objeto: conceptos básicos, lenguajes de programación, bases de datos, aplicaciones y métodos. En los capítulos de métodos de análisis y de diseño realiza una exposición y un examen de los de mayor difusión, cubriendo prácticamente todas las escuelas existentes. Incorpora un capítulo interesante relacionado con la inteligencia artificial y la teoría de los conjuntos difusos. Se considera un buen libro de referencia donde se puede adquirir una panorámica amplia de la Orientación al Objeto.

- **Hatley, D. J., Pirbhai, I. A.** "*Strategies for Real-Time System Specification"*. Dorset House Pub. Co., 1987.

  Se describen los métodos para la especificación de los requisitos y el diseño de sistemas de tiempo real basados en computadora. Los métodos propuestos integran las herramientas del análisis estructurado, fundamentalmente el DFD, para la construcción del modelo de requisitos y del modelo arquitectónico. Partiendo de la base de los DFD propone una herramienta de modelado para los procesos de control los CFD (*Control Flow Diagrams*), y otra para la representación de la configuración física del sistema los ACF (*Architecture Flow Diagrams*). Los métodos y la construcción de modelos están apoyados por un ejemplo que sirve de hilo conductor a lo largo del libro. Aunque el método propuesto para la asignatura de Ingeniería del Software I va a seguir más la propuesta de Ward & Mellor, este libro tiene aportaciones muy interesantes en el modelado de sistemas de tiempo real, sobre todo en el modelado arquitectónico.

🕮 **Henderson-Sellers, B**. *"A Book of Object-Oriented Knowledge"*. 2nd ed. Prentice Hall PTR, 1997.

Es una introducción básica al enfoque orientado al objeto en la Ingeniería del Software. El libro describe las ideas básicas relacionadas con el análisis, el diseño y la implementación e incluye un buen conjunto de referencias para el estudio con profundidad de cada uno de esos temas. Aporta en cada tema el diseño de las transparencias que pueden ser utilizadas en la explicación del mismo. Es una buena presentación de los conceptos y los métodos de la Orientación al Objeto.

🕮 **Jacobson, I.; Booch, G., Rumbaugh, J**. "*The Unified Software Development Process*". Addison-Wesley Object Technology Series, 1999.

Otro de los libros de la trilogía de los creadores de UML, en el que se ofrece una referencia completa del proceso de desarrollo unificado que se adapta a UML. Su consulta puede ser beneficiosa en el estudio de los temas 4 y 5.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Esta traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997 en su segunda edición supone una extensa revisión respecto a la primera. Se considera una referencia importante para el Tema 4.

🕮 **Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma. 1996.

Libro recomendado en la asignatura Ingeniería del Software de primer ciclo que puede ser útil también en el segundo para repasar y profundizar en algunos aspectos relacionados con: *Sistemas de Información* (capítulos 1 y 2), *El Proceso de Desarrollo de Software* (capítulos 3-16) y *Tecnología* (capítulos 17-19) que se tratan en las dos primeras unidades docentes de la asignatura.

🕮 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGraw-Hill. 1998.

Este libro, aunque ya ha sido manejado ampliamente por los alumnos en primer ciclo, puede ser igualmente útil en las asignaturas de Ingeniería del Software de segundo ciclo ya que abarca la mayoría de los temas relacionados con esta materia, tratándolos con bastante profundidad. Cada tema se complementa además con un gran número de referencias de otras fuentes de información sobre el tema, incluyendo recursos disponibles en Internet. Corresponde a la traducción del libro del mismo autor "*Software Engineering: A Practitioner´s Approach*", 4th Edition, McGraw-Hill, 1997. Se considera un texto imprescindible de referencia para esta asignatura siendo una referencia interesante en la práctica totalidad de los temas.

📖 **Sommerville, I.** "*Software Engineering*". 5th Edition, Addison-Wesley, 1996.

En esta última edición de la obra de Sommerville, éste realiza una extensa revisión de la anterior edición incluyendo nuevos temas y modificando sustancialmente algunos de los existentes como los dedicados a la tecnología CASE y evolución del software. Los contenidos se adaptan a la recomendación curricular ACM/IEEE-CS91 para los módulos SE2 a SE5. Se considera un libro de consulta importante para la práctica totalidad de la asignatura.

📖 **Rational Software Corporation.** "*OMG Unified Modeling Language Specification. Version 1.3*". http://www.rational.com. Abril, 1999.

Esta documentación corresponde a la última versión del estándar UML en la que se hace una revisión de la versión 1.1, enfocada fundamentalmente a la resolución de inconsistencias y clarificación de ambigüedades. Esta documentación incluye un apartado dedicado a la semántica de UML en el que se presenta la semántica de los elementos de los modelos de objetos estáticos y dinámicos, además se describe de una manera semi-formal el metamodelo. En otra sección del documento denominada *guía de la notación UML* se ofrece la notación propuesta para la representación gráfica de los modelos, compuesta de nueve tipos de diagramas diferentes. La consulta de esta documentación es imprescindible en el tema 5.

📖 **Rumbaugh, J., Jacobson, I., Booch, G.** "*The Unified Modeling Language. Reference Manual*". Addison-Wesley Object Technology Series, 1999.

Manual completo de referencia de UML que completa la trilogía de libros sobre UML de los tres autores. En el libro se expone en primer lugar la forma de llevar a cabo el modelado y los diagramas utilizados en cada una de las vistas del sistema. Posteriormente aparece en orden alfabético la descripción de todos los componentes del lenguaje. El libro puede ser de gran ayuda en el estudio de los temas 4 y 5, especialmente en éste último.

📖 **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993.

La versión en castellano de este libro corresponde a la traducción del texto del mismo autor "*Modern Structured Analysis*", Prentice-Hall, 1989. En él se describe el método propuesto por Yourdon que se basa en las ideas de DeMarco en el año 1979. Recomendado como referencia importante en la asignatura de Ingeniería del Software de primer ciclo, puede ser de ayuda en esta asignatura para el estudio del tema 3.

A.3.6 Administración de Proyectos Informáticos

El propósito fundamental de esta asignatura es proporcionar los conocimientos y capacidades prácticas necesarias para llevar a cabo las actividades de Ingeniería del Software relacionadas con los aspectos de gestión y control de proyectos. Dichas tareas comienzan antes del inicio de cualquier actividad técnica y continúan a lo largo de todo el ciclo de vida del software. El

objetivo final de dichas actividades es obtener un producto software final de calidad y fiable desarrollado mediante un proceso eficiente y productivo.

Los objetivos de la unidad docente que se pretenden conseguir en esta asignatura son los siguientes:

T10, T11, T12, P3, P8, H1, H2, H3 y **H4**.

Los apartados que se exponen a continuación siguen la pauta marcada en las asignaturas precedentes.

A.3.6.1 Ficha de la asignatura

Asignatura	*Administración de Proyectos Informáticos (troncal)*
Créditos	*6T + 3P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*2º*
Cuatrimestre	*1º y 2º (anual)*
Responsable	*María N. Moreno García (mmg@usal.es)*
Página web de la asignatura	*http:/lisisu02.usal.es/~mmoreno/api.html*

Tabla A.18. Administración de Proyectos Informáticos

A.3.6.2 Prerrequisitos

Para lograr los objetivos anteriores es fundamental que el alumno haya adquirido previamente conocimientos sobre el proceso de desarrollo, ya que dichas nociones son básicas para la comprensión y asimilación de los conceptos y métodos que se tratan en esta asignatura. Los conocimientos requeridos se adquieren en las asignaturas **Ingeniería del Software** que se imparte en 3º de ITIS y **Análisis de Sistemas** perteneciente a primer curso del segundo ciclo.

A.3.6.3 Temario teórico/práctico

Presentación de la asignatura (1 hora)
Tema 1. Visión general de la administración de proyectos (6 horas)
Tema 2. Medición del software (11 horas)
Tema 3. Métodos de estimación y gestión del riesgo (10 horas)
Tema 4. Planificación temporal de proyectos (12 horas)
Tema 5. Gestión de la calidad (15 horas)
Tema 6. Gestión de configuraciones (5 horas)

Tabla A.19. Programa teórico de la asignatura Administración de Proyectos Informáticos (6 créditos)

Laboratorio: uso de herramientas automatizadas (30 horas)
Práctica 1. Aplicación de métricas (4 horas)
Práctica 2. Estimación de coste y esfuerzo de un proyecto (8 horas)
Práctica 3. Planificación temporal del proyecto (18 horas)

Tabla A.20. Programa práctico de la asignatura Administración de Proyectos Informáticos (3 créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos
- Trabajos voluntarios realizados por los alumnos
- Conferencias invitadas

En la Tabla A.21 se presenta la correspondencia existente entre el temario y los objetivos perseguidos en esta asignatura.

Elemento Docente	Objetivos
Tema 1	T10, T11, T12, H3
Tema 2	T11, H3
Tema 3	T10, H3
Tema 4	T10, H3
Tema 5	T10, T11, H3
Tema 6	T12, H3
Práctica 1	P8, H1, H2, H4
Práctica 1	P3, H1, H2, H4
Práctica 2	P3, H1, H2, H4

Tabla A.21. Correspondencia entre el temario teórico/práctico y los objetivos de la asignatura

A.3.6.4 Líneas de acción

Como en las asignaturas anteriores, seguidamente se comentarán todos los componentes de las líneas de acción que contribuirán a la consecución de los objetivos.

T10 Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.

T11 Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Ofrecer una visión general de las actividades de gestión, planificación y seguimiento de proyectos	En el tema 1	Responsable de la asignatura	Transparencias Bibliografía básica	Examen
Mostrar el alcance de las métricas del software y las dos vertientes de su aplicación: evaluación y predicción	En el tema 2 y en las prácticas de la asignatura	Responsable de la asignatura Alumnos	Transparencias Bibliografía básica	Defensa de la práctica Examen
Estudiar y aplicar los conceptos y métodos relacionados con las actividades descritas en el primer tema.	Todos los temas de la asignatura Prácticas de la asignatura	Responsable de la asignatura Alumnos	Transparencias Bibliografía básica	Defensa de las prácticas Examen

T12 Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Examinar los conceptos y tareas asociados a la gestión de configuraciones, acentuando la importancia de asegurar la calidad y la consistencia de los cambios.	Tema 6	Responsable de la asignatura	Transparencias Bibliografía básica	Examen

P3 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Utilización de herramientas automatizadas de gestión para la planificación y la estimación de costes.	En las clases prácticas	Responsable de la asignatura Alumnos	Medios audiovisuales Herramientas de gestión. Documentación de trabajos realizados el curso anterior	Defensa de las prácticas

P8 Recolección de diferentes métricas en el desarrollo de sistemas software reales.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Utilización de herramientas automatizadas para la aplicación de métricas.	En las clases prácticas	Responsable de la asignatura Alumnos	Herramientas de gestión. Documentación de trabajos realizados el curso anterior	Defensa de las prácticas

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Fomentar la participación en las clases de problemas	A lo largo del curso	Responsable de la asignatura Alumnos	Pizarra	Participación de los alumnos
Defensa oral de las prácticas	Defensa de la práctica	Alumnos	Medios audiovisuales	Defensa de la práctica
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Propuesta de ejercicios de aplicación de los conceptos teóricos	Durante las clases de teoría	Responsable de la asignatura alumnos	Transparencias Pizarra	Evaluación de los ejercicios entregados
Documentación de las prácticas	Defensa de las prácticas	Alumnos Responsable de la asignatura	Herramientas de gestión Bibliografía	Defensa de la práctica
Memoria de los trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Completar las transparencias utilizadas en clase con la bibliografía recomendada	A lo largo de la asignatura	Alumnos	Biblioteca	Examen
Promover seminarios realizados por los alumnos como resultado de trabajos voluntarios	A lo largo de la asignatura	Grupos de alumnos	Bibliografía Recursos de Internet	Nota *extra* para estos trabajos voluntarios

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Realización de las prácticas en grupos de 2 personas	Durante las clases prácticas	Alumnos Responsable de la asignatura	Documentación de trabajos realizados el curso anterior	Defensa de la práctica

A.3.6.5 Criterios de evaluación de la asignatura

a) Parte de Teoría

- Un examen final en el mes de junio.

- Un examen final en el mes de septiembre.

b) Parte Práctica

- En junio la evaluación de esta parte de la asignatura se realizará mediante la valoración de los trabajos presentados. La nota será la media de las obtenidas en cada uno de los trabajos prácticos. La defensa de dichos trabajos se realizará de forma individualizada.

- En septiembre se realizará un examen en el que el alumno tendrá que resolver un supuesto práctico con la ayuda de una herramienta.

c) Trabajos voluntarios presentados (de 0,5 a 1,5 puntos sumados a la nota conseguida en los apartados anteriores, siempre que en los apartados anteriores se obtenga la calificación mínima exigida).

d) En el caso de haber superado la parte teórica y la práctica, se aplicará la siguiente fórmula para la obtención de la calificación final:

Nota Final = ((Teoría*0,6) + (Práctica*0,3))* 10/9 + Nota trabajos

A.3.6.6 Bibliografía básica de referencia

Siguiendo la misma línea que en el resto de las asignaturas de la unidad docente, la bibliografía que aquí se incluye servirá de complemento al material suministrado para el seguimiento de las clases. Los criterios seguidos para su elección son los mismos que en los casos anteriores.

📖 **Burnett, K.** "*The Project Management Paradigm*". Springer, 1998.

Este texto proporciona un enfoque de la gestión de proyectos basado en la combinación de aspectos humanos y principios metodológicos para conseguir el éxito del proyecto. Es interesante la visión que ofrece sobre los conceptos, principios y métodos de estimación y gestión de riesgos. El capítulo que dedica a la gestión de la calidad es bastante completo, por lo que puede ser muy útil en el estudio del tema 5.

📖 **Cos, M.** "*Teoría General del Proyecto*". Síntesis, 1997.

El primer volumen de este texto cubre los conceptos y métodos de organización de proyectos, estimación, planificación y control. El último capítulo está dedicado al plan de calidad del proyecto.

📖 **Fenton, N. E., Pfleeger, S. L.** "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.

Excelente obra sobre métricas del software que cubre diversos aspectos de su aplicación como son la evaluación y predicción del tamaño y funcionalidad del software, de la productividad, de costes y esfuerzos y de la calidad del producto y del proceso de desarrollo. En el se recogen gran número de métricas existentes en bibliografía así como algunas propuestas por los mismos autores del libro. Los primeros capítulos dedicados a los fundamentos de la medición y a la forma de realizar medidas eficaces son muy recomendables. Imprescindible en el estudio de los temas 2, 3 y 5.

📖 **McConnell, S.** "*Desarrollo y Gestión de Proyectos Informáticos*". McGraw Hill, 1997.

Traducción de la primera edición en inglés de "*Rapid Development*". Este libro, dirigido principalmente a las estrategias de desarrollo rápido, dedica el capítulo 4 a explicar de manera clara las bases del desarrollo de software. Tiene también temas dedicados a la estimación y gestión de riesgos. Los capítulos 11, 12 y 13 constituyen una buena referencia sobre los aspectos a considerar en la formación de equipos de trabajo (motivación, características, estructura...), al igual que el capítulo 15 en el apartado de herramientas de mejora de la productividad. La tercera parte del libro, dedicada a "métodos recomendables" cubre la práctica totalidad de los temas de la asignatura.

📖 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 4ª Edición. McGrawHill. 1998.

Este clásico de la Ingeniería del Software constituye una referencia obligada en casi todos los temas de la asignatura. El capítulo 3 trata de forma global los aspectos de gestión del proyecto. La medición de software aparece en apartados sueltos incluidos en diferentes capítulos y en un tema relacionado con el proceso software que está dedicado casi completamente a la medición. El resto de los temas de la asignatura los contempla también de forma detallada en diferentes capítulos.

📖 **Puig, J.** "*Proyectos Informáticos. Planificación, Desarrollo y Control*", Paraninfo, 1994.

El texto de Puig puede ser de utilidad para obtener una visión global de las actividades clásicas de gestión de proyectos que se tratan de una manera concisa y clara. El libro utiliza, en algunos casos, notaciones y terminología obsoleta y no incluye técnicas actuales, por lo que necesitaría una revisión.

🕮 **Quang, P., Goñi, J.** "*Dirección de Proyectos Informáticos*". Eyrolles, 1994.

En este libro se contemplan los métodos clásicos de organización de equipos, estimación, planificación, seguimiento y control de calidad de un proyecto. Además se ofrecen una serie de recomendaciones obtenidas de experiencias prácticas. El libro proporciona una visión superficial de los temas de esta asignatura, por lo que se aconseja su uso únicamente en el tema de introducción.

🕮 **Romero, C.** "*Técnicas de Programación y Control de Proyectos*". Pirámide, 1997.

Describe de forma detallada tres métodos clásicos de planificación temporal (PERT, CPM y ROY) partiendo de la explicación previa de los conceptos sobre teoría de grafos que son la base de tales métodos. Puede ser de gran ayuda en el estudio del tema 4.

🕮 **Sommerville, I.** "*Software Engineering*". 5th Edition. Addison-Wesley, 1996.

La quinta edición de esta obra sobre ingeniería del software incorpora nuevos capítulos sobre temas de gestión (personal, calidad...) y mejora los dedicados a la tecnología CASE y evolución del software. Estos cambios lo convierten un libro de consulta indispensable de la asignatura.

A.3.7 Sistemas de Información

El propósito fundamental de esta asignatura es la aplicación práctica de los principios de la Programación Orientada a Objetos estudiados en otras asignaturas. Los objetivos de la unidad docente que se pretenden conseguir en esta asignatura son los siguientes:

P7, H1, H2, H3 y **H4**.

Los apartados que se exponen a continuación siguen la pauta marcada en las asignaturas precedentes.

A.3.7.1 Ficha de la asignatura

Asignatura	*Sistemas de Información (troncal)*
Créditos	*0T + 9P*
Estudios	*Ingeniería en Informática (2º ciclo)*
Plan	*B.O.E de 1-7-1999*
Curso	*2º*
Cuatrimestre	*1º*
Responsable	*José Rafael García-Bermejo Giner (coti@usal.es)*
Página web de la asignatura	*http://acebuche.usal.es*

Tabla A.22. Sistemas de Información

A.3.7.2 Prerrequisitos

Para lograr los objetivos anteriores es fundamental que el alumno haya adquirido previamente conocimientos sobre la realización de proyectos software, tanto en la asignatura de **Ingeniería del Software** del tercer curso del primer ciclo, como en la asignatura de **Análisis de Sistemas** del primer curso del segundo ciclo.

Además, sería aconsejable que el alumno tuviera conocimientos prácticos de **Programación Orientada a Objetos**, del tercer curso del primer ciclo.

A.3.7.3 Temario práctico

Presentación de la asignatura (1 hora)

Tema 1. Conceptos básicos de orientación a objetos (9 Horas)

Tema 2. El lenguaje Java. Java frente a C/C++ (20 Horas)

Tema 3. Applets (15 Horas)

Tema 4. Creación de aplicaciones en Java (15 Horas)

Tema 5. Realización de un supuesto práctico (30 Horas)

Tabla A.23. Programa de prácticas (9 Créditos)

Actividades Docentes Complementarias

- Seminarios impartidos sobre temas específicos.
- Trabajos voluntarios realizados por los alumnos.
- Conferencias invitadas.

A.3.7.4 Líneas de acción

P7 Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Búsqueda e instalación del entorno de desarrollo más adecuado y reciente para su uso en la asignatura	En la fase inicial del curso	Responsable de la asignatura Alumnos	Internet Revistas Bibliografía	
Cambio de mentalidad para la adecuada aplicación del paradigma objetual	A partir del tema 2	Responsable de la asignatura	Numerosos ejercicios prácticos	Práctica en dos ocasiones a lo largo del curso y examen teórico

H1 Mejora de la expresión oral.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Defensa en seminarios de algún tema teórico o ejercicio práctico	A lo largo del curso	Responsable de la asignatura Alumnos	Audiovisuales	Participación de los alumnos

H2 Mejora en la redacción de documentos técnicos.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Extracción y documentación de las especificaciones	En el desarrollo de los trabajos prácticos	Alumnos	Enunciados de los problemas	Evaluación de los ejercicios entregados

H3 Potenciación de la capacidad del alumnos para la búsqueda de información (manejo de fuentes bibliográficas, Internet, foros de discusión...).

Línea de acción	Cuándo	Quién	Medios	Evaluación
Búsqueda de información actualizada en Internet	A lo largo de la asignatura	Alumnos	Internet Revistas	Resultados de los ejercicios prácticos

H4 Capacitar a los alumnos para el trabajo en grupo.

Línea de acción	Cuándo	Quién	Medios	Evaluación
Construcción de aplicaciones modulares, realizando los distintos grupos los módulos componentes	Durante las clases prácticas	Alumnos Responsable de la asignatura	Enunciado Entorno de desarrollo	Por la interfaz y la implementación de los módulos

A.3.7.5 Criterios de evaluación de la asignatura

a) Parte Práctica

- En diciembre y en febrero se examinan los ejercicios realizados.
- En febrero se realiza un examen escrito.
- En septiembre se realizará un examen escrito.

c) Se aplica la siguiente fórmula para la obtención de la calificación final:

Nota Final = (Ejercicios entregados*0,5) + (Examen*0,5)

A.3.7.6 Bibliografía básica de referencia

Siguiendo la misma línea que en el resto de las asignaturas de la unidad docente, la bibliografía que aquí se incluye servirá de complemento al material suministrado para el seguimiento de las clases.

🕮 **Jaworski, J**. "*Java 1.2 Unleashed*". Sams, 1998.

- **Cadenhead, R.** "*Teach Yourself JAVA 1.2 in 24 hours*". Sams, 1999.
- **Lemay, L., Cadenhead, R.** "*Teach Yourself Java 2 Platform in 21 Days: Professional Reference Edition*". Sams Teach Yourself in 21 Days Series. Sams, 1999.
- **SUN Microsystems.** "*Java™ Standard Edition Platform Documentation*". http://java.sun.com/docs/index.html. [Última vez visitado, 16-3-2000]. 2000.
- **SUN Microsystems.** "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 12-4-2002]. March 2002.

A.3.8 Proyecto I.I.

El proyecto de la Ingeniería Informática se convierte de nuevo en otra excelente ocasión para la consecución del objetivo **P9**, pero esta vez con la experiencia de haber realizado previamente otro proyecto y haber cursado dos años más estudios universitarios de segundo ciclo.

Los objetivos de este proyecto no deben ser un calco de los del proyecto de I.T.I.S, sino que debe convertirse en un trabajo donde el alumno demuestre su madurez, además de sus conocimientos y dominio de la técnica. Así, el proyecto debe ir acompañado de las métricas oportunas, planes de implantación y mantenimiento, justificaciones teóricas basadas en el estado del arte del tema abordado...

En este sentido la guía recogida en [García et al., 2000] sigue siendo una referencia de consulta válida, pero ha de tenerse en cuenta que dicha guía fue desarrollada para los proyectos de I.T.I.S, no para un proyecto de un segundo ciclo.

A.4 Referencias

[Buxton et al., 1976] Buxton, J. M., Naur, P., Randell, B. (Editors). "*Software Engineering Concepts and Techniques*". In Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[García et al., 1999a] García Peñalvo, F. J., Moreno García, Mª N., González Talaván, G., Moreno Montero, Á. Mª. "*Plan de Calidad para Asignaturas en Ingenierías Técnicas en Informática*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 46 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 1999b] García Peñalvo, F. J., Moreno García, Mª N., Montero García, E., Arranz Val, P. "*Evaluación del Profesorado: Un Protocolo de Evaluación por Pares*". Actas del Congreso Nacional de Informática Educativa CONIED'99. Editores M. Ortega y J. Bravo. (Puertollano (Ciudad Real), 17-19 de Noviembre de 1999). Resumen en página 47 y ponencia en versión digital (CD-ROM). 1999.

[García et al., 2000] García Peñalvo, F, J., Maudes Raedo, J. M., Piattini Velthuis, M. G., García-Bermejo Giner, J. R., Moreno García, Mª N. "*Proyecto de Final de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación*". Departamento de Informática y Automática de la Universidad de Salamanca. Versión 1.5.2. http://tejo.usal.es/~fgarcia/doc/pfc.pdf. Marzo, 2000.

[Leveson, 1997] Leveson, N G. "*Software Engineering: Stretching the Limits of Complexity*". Communications of the ACM 40(2): 129-131. February 1997.

[Parnas, 1997] Parnas, D. L. "*Software Engineering: An Unconsummated Marriage*". Communications of the ACM 40(9): 128. September 1997.

[Shaw y Tomayko, 1991] Shaw, M., Tomayko, J. E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10). Software Engineering Institute – Carnegie Mellon University. Pittsburgh, Pennsylvania 15213 (USA). August 1991.

Apéndice B:
Bibliografía comentada

En este anexo se relacionan y comentan aquellas referencias bibliográficas que constituyen la principal fuente bibliográfica de referencia y consulta para los alumnos de las asignaturas presentadas en este Proyecto Docente e Investigador. Esta es la bibliografía que se considera de manejo obligatorio por parte de los alumnos, tanto para completar las explicaciones de los temas que se realizan en las horas de teoría como para que los alumnos puedan tener una visión más completa y exacta de las materias impartidas.

En los casos en los que existe traducción al español se incluye esta referencia. A pesar de que los alumnos por regla general prefieren la edición traducida, se les anima constantemente a que consulten la bibliografía original. Se considera que esto contribuye a crear buenos hábitos de formación personal, elimina los "errores" de traducción y permite acceder, por lo general, a las fuentes con uno o dos años de anticipación.

Además de esta bibliografía a los alumnos, para cada uno de los temas, se les proporcionan bien referencias o bien artículos relacionados con el tema incluidos en las revistas que se reciben en el Departamento y en la biblioteca.

Por último, señalar que en las últimas ediciones de algunos de los libros aquí presentados suelen incluirse al final del correspondiente capítulo un amplio catálogo de fuentes adicionales de lectura. En este sentido hay que destacar el conjunto de referencias aportadas por el libro de **Roger S. Pressman** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.

📖 **Booch, G**. "*Análisis y Diseño Orientado a Objetos con Aplicaciones*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Corresponde a la traducción del libro "O*bject-Oriented Analysis and Design with Applications*" 2nd Edition, Benjamin/Cummings, 1994. Esta edición es una actualización del libro del año 1991, "*Object-Oriented Design with Applications*". En la primera sección se introduce los conceptos y el lenguaje utilizado en la orientación al objeto. Sobre esta base en la segunda sección introduce la notación y la metodología de análisis y diseño orientado al objeto propuesta por el autor, y que constituye una de las metodologías clásicas en la orientación al objeto. En esta segunda edición, a diferencia de la primera, se distingue entre la notación básica y la avanzada, considerando que esta última es la menos utilizada. Un aspecto a reseñar en las propuestas es la división entre micro y macro proceso, en la descripción del proceso de análisis y diseño orientado al objeto. A diferencia de la primera edición, en esta se hace un mayor énfasis en el análisis y de ahí que de la primera a la segunda edición se le haya añadido el término análisis en el título.

Es uno de los libros de referencia básicos para la asignatura *Ingeniería del Software*. No aparece entre la bibliografía básica de la asignatura *Análisis de Sistemas*, aunque al describir el Tema 6 (Métodos orientados a objetos) de la misma, es uno de los libros que aparecen en la bibliografía básica del tema, sirviendo de bibliografía complementaria para muchos otros temas.

📖 **Booch, G., Rumbaugh, J., Jacobson, I.** "*El Lenguaje Unificado de Modelado*". Addison-Wesley, 1999.

El texto en castellano corresponde a la traducción del libro de estos mismos autores "*The Unified Modeling Language User Guide*", Addison-Wesley, 1999. Es uno de los libros de la trilogía publicada por los creadores del lenguaje modelado UML, en él se proporciona una referencia de uso de las características específicas de UML, centrándose fundamentalmente la notación gráfica. En cada una de las secciones en que se divide el libro, realiza una separación entre las características básicas y las avanzadas, hecho que facilita su estudio y lo hace muy recomendable para completar este tema. También trata el proceso unificado en uno de los apéndices del libro, pero de una manera muy superficial.

Es una referencia básica para las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*, en cualquier tema relacionado con UML (*Unfied Modeling Language*).

📖 **Burnett, K.** "*The Project Management Paradigm*". Springer-Verlag, 1998.

Este libro proporciona un enfoque de la gestión de proyectos basado en la combinación de aspectos humanos y principios metodológicos para conseguir el éxito del proyecto. Es interesante la visión que ofrece sobre los conceptos, principios y métodos de estimación y gestión de riesgos. El capítulo que dedica a la gestión de la calidad es bastante completo, por lo que puede ser muy útil en el estudio del Tema 5 (Gestión y aseguramiento de la calidad) de la asignatura *Administración de Proyectos Informáticos*.

📖 **Cos, M.** "*Teoría General del Proyecto*". Síntesis, 1997.

El primer volumen de este texto cubre los conceptos y métodos de organización de proyectos, estimación, planificación y control. El último capítulo está dedicado al plan de calidad del proyecto.

Está recomendado como bibliografía básica para la asignatura *Administración de Proyectos Informáticos*.

📖 **Fenton, N. E., Pfleeger, S. L.** "*Software Metrics. A Rigorous & Practical Approach*". PWS, 1997.

Excelente libro sobre métricas del software que cubre diversos aspectos de su aplicación como son la evaluación y predicción del tamaño y funcionalidad del software, de la productividad, de costes y esfuerzos y de la calidad del producto y del proceso de desarrollo. En el se recogen gran número de métricas existentes en bibliografía así como algunas propuestas por los mismos autores del libro. Los primeros capítulos dedicados a los fundamentos de la medición y a la forma de realizar medidas eficaces son muy recomendables.

Es una de las referencias Imprescindible en el estudio de los temas 2, 3 y 5 de la asignatura *Administración de Proyectos Informáticos*.

📖 **Graham, I.** "*Métodos Orientados a Objetos*". 2ª Edición. Addison-Wesley/Díaz de Santos, 1996.

Traducción del libro del mismo autor "*Object-Oriented Methods*" 2nd Edition. Addison-Wesley, 1994. En un libro introductorio a la orientación al objeto. Cubre de forma amplia y correcta todos los aspectos de la orientación al objeto: conceptos básicos, lenguajes de programación, bases de datos, aplicaciones y métodos. En los capítulos de métodos de análisis y métodos de diseño realiza una exposición y un análisis de los de mayor difusión y aceptación cubriendo prácticamente todas las escuelas existentes. Incorpora un capítulo

interesante relacionado con la inteligencia artificial y la teoría de los conjuntos difusos, que escapa de las necesidades de los alumnos de primer ciclo. Se considera un buen libro de referencia donde se puede adquirir una panorámica amplia de la orientación al objeto.

Directamente no aparece como bibliografía básica de ninguna de las asignaturas discutidas en este Proyecto Docente, aunque es bibliografía básica y complementaria en los diversos temas de las asignaturas *Ingeniería del Software* y *Análisis de Sistemas* que tienen relación con la orientación a objetos.

🕮 **Harry, A.** "*Formal Methods. Fact File*". John Wiley & Sons, 1996.

Este libro, aunque está dedicado principalmente al estudio de los lenguajes VDM y Z, presenta una amplia introducción a los métodos formales y a sus fundamentos matemáticos como la teoría de conjuntos o la lógica formal. Asimismo, establece una clasificación de los métodos en función de que la especificación sea constructiva (basada en modelos) o axiomática (basada en propiedades) explicando las subclasificaciones de cada una de ellas.

Es referencia básica para la asignatura *Análisis de Sistemas*, en particular para el Tema 8 (Técnicas formales de especificación).

🕮 **Hatley, D. J., Pirbhai, I. A.** "*Strategies for Real-Time System Specification*". Dorset House Pub. Co., 1987.

Se describen los métodos para la especificación de los requisitos y el diseño de sistemas de tiempo real basados en computadora. Los métodos propuestos integran las herramientas del análisis estructurado, fundamentalmente el DFD, para la construcción del modelo de requisitos y del modelo arquitectónico. Partiendo de la base de los DFD propone una herramienta de modelado para los procesos de control los CFD (*Control Flow Diagrams*), y otra para la representación de la configuración física del sistema los ACF (*Architecture Flow Diagrams*). Los métodos y la construcción de modelos están apoyados por un ejemplo que sirve de hilo conductor a lo largo del libro.

No es bibliografía básica de ninguna de las asignaturas presentadas, pero sí lo es de algunos temas concretos de las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*, por ejemplo del Tema 2 (Análisis y Especificación de Requisitos) de *Ingeniería del Software*, y del Tema 9 (Desarrollo de sistemas complejos) de *Análisis de Sistemas*; apareciendo además como bibliografía complementaria en otros temas.

🕮 **Jacobson, I., Booch, G., Rumbaugh, J.** "*El Proceso Unificado de Desarrollo de Software*". Addison-Wesley, 2000.

Este libro es la traducción de la obra de los mismos autores "*The Unified Software Development Process*", Addison-Wesley, 1999. Es otro de los libros que forma la trilogía escrita por los creadores de UML. Este libro en concreto se centra en el aspecto de proceso unificado, dividiendo el libro en tres partes, la primera de ellas es una introducción al proceso unificado, y es la parte que mejor se ajusta como bibliografía básica para los temas 2 y 7 de *Análisis de Sistemas*. Las otras dos partes (flujos de trabajo fundamentales y desarrollo iterativo e incremental) pueden tomarse como complemento para los mismos temas.

🕮 **Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G.** "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

Se presenta y describen la técnica de los casos de uso (*use cases*) y la metodología OOSE (*Object-Oriented Software Engineering*) que constituye la parte pública de la metodología *Objectory*. Los casos de uso constituye una de las técnicas de mayor aceptación en la actualidad para el análisis de requisitos. Esta técnica ayuda a asegurar que el sistema se desarrolla capturando los requisitos del sistema desde el punto de vista del usuario.

No es un libro recomendado como bibliografía básica en ninguna de las asignaturas presentadas, ni tampoco en ningún tema, pero si es frecuentemente referenciado como bibliografía complementaria, y es la base técnica para los casos de uso y en parte para UP (*Unified Process*).

🕮 **McConnell, S.** "*Desarrollo y Gestión de Proyectos Informáticos*". McGraw Hill, 1997.

Traducción de la primera edición en inglés de "*Rapid Development*". Este libro, dirigido principalmente a las estrategias de desarrollo rápido, está recomendado como bibliografía básica de la asignatura *Administración de Proyectos Informáticos*. Dedica el capítulo 4 a explicar de manera clara las bases del desarrollo de software. Tiene también temas dedicados a la estimación y gestión de riesgos. Los capítulos 11, 12 y 13 constituyen una buena referencia sobre los aspectos a considerar en la formación de equipos de trabajo (motivación, características, estructura...), al igual que el capítulo 15 en el apartado de herramientas de mejora de la productividad. La tercera parte del libro, dedicada a "métodos recomendables" cubre la práctica totalidad de los temas de la asignatura.

🕮 **Meyer, B.** "*Construcción de Software Orientado a Objetos*". 2ª Edición. Prentice Hall, 1999.

Corresponde a la traducción del libro "*Object-Oriented Software Construction*", 2nd Edition, Prentice-Hall, 1997. Esta segunda edición es más que una simple actualización de la primera edición, con una ampliación de conceptos más que considerable. Aunque es un libro que se adecua más a una asignatura de programación orientada a objetos, los contenidos de los tres primeros capítulos se adaptan muy bien a la asignatura *Ingeniería del Software*, especialmente el Capítulo 3 dedicado a la modularidad, que es una referencia importante para el Tema 4 (Diseño del software) de esta asignatura.

🕮 **Ministerio de Administraciones Públicas.** "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

Esta obra describe la metodología METRICA 3.0 para las Administraciones Públicas en España. Es bibliografía básica en la asignatura *Análisis de Sistemas*, puesto que uno de sus temas está exclusivamente dedicado al estudio de esta metodología (tema 5), pero también lo es en la asignatura *Administración de Proyectos Informáticos*, puesto que uno de los procesos de la metodología es la Planificación de Sistemas de Información, y dos de sus procesos de interfaz se refieren a la Gestión de Proyectos y la Gestión de Configuraciones, a lo que hay que añadir la explicación de las diversas técnicas utilizadas a lo largo del ciclo de vida.

Esta referencia se utiliza también como bibliografía complementaria en diversos temas de la asignatura *Ingeniería del Software* por la explicación que hace de las técnicas de modelado estructuradas.

La obra en sí se divide en tres volúmenes: una guía de referencia que explica los procesos, actividades y tareas propias de la metodología; una guía que explica los procesos interfaz, las técnicas y las prácticas; y finalmente una guía de usuario resumen de la metodología.

🕮 **Muller, P. A.** "*Modelado de objetos con UML*". Eyrolles-Ediciones Gestión 2000, 1997.

De este libro, traducido del original "*Modélisation Objet avec UML*", se recomienda la lectura de su segundo capítulo en el que se exponen las características generales de la orientación a objetos sin hacer mención a UML, aunque se utiliza su notación para representar los diferentes conceptos. Es también muy aconsejable en este tema, sobre todo en lo que se refiere al aprendizaje de la notación UML, a la que dedica el tercer capítulo de los cinco que componen el libro. Muller presenta la notación a partir de la

descripción de los nueve tipos de diagramas de UML, explicando en cada uno de ellos los elementos que lo componen, las relaciones entre ellos, la forma de construir los diagramas, y otros aspectos destacados. En el último capítulo presenta un caso de estudio en el que se aplican muchos de los conceptos teóricos expuestos en el libro. También es interesante el capítulo 4 en el que se describe el marco de desarrollo de sistemas orientados a objetos.

No es referencia básica de ninguna de las asignaturas presentadas, aunque se utiliza como referencia básica de los temas relacionados con UML tanto en *Ingeniería del Software* como en *Análisis de Sistemas*.

📖 **OMG.** "*OMG Unified Modeling Language Specification Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. September 2001.

Esta documentación corresponde al texto completo de la versión 1.4 de UML aceptada por OMG. Esta documentación consta de un metamodelo formal, una notación y su semántica, un modelo de intercambio basado en XMI, la especificación del lenguaje OCL (*Object Constraint Language*). En el metamodelo se realiza una exposición clara de los conceptos e ideas, de la sintaxis y de la semántica de UML. En la notación se describen los diferentes tipos de diagramas: de casos de uso, de clases, de secuencias, de colaboración, de estados, de actividad, de componentes y de despliegue. Esta documentación es básica tanto en la asignatura *Ingeniería del Software* como en la asignatura *Análisis de Sistemas*, en todos los temas relacionados con UML, y en el caso de *Análisis de Sistemas* también en el tema dedicado a los métodos formales (tema 8).

Aunque es una buena referencia de consulta, la semántica de los elementos de UML está mucho mejor tratada y ampliada en el libro **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000, una de las mejores referencias para UML.

📖 **Piattini, M., Calvo-Manzano, J. A., Cervera, J., Fernández, L.** "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

Libro de introducción a la Ingeniería del Software, centrado en la construcción de software de gestión. Consta de tres partes: Sistemas de Información (capítulos 1 y 2), El Proceso de Desarrollo de Software (capítulos 3-16) y Tecnología (capítulos 17-19).

Es un libro de consulta que se puede utilizar en todos los temas de la parte teórica de la asignatura *Ingeniería del Software*, aunque con una mayor profusión en los temas 1, 2, 4

y 5. Asimismo es una referencia básica en la asignatura *Análisis de Sistemas*, aunque quizás sea una referencia más complementaria para los temas tratados en la asignatura. Sucede lo mismo con la asignatura *Administración de Proyectos Informáticos*, no es un libro de gestión de proyectos, pero algunos de sus capítulos resultan útiles para ciertos temas de la asignatura.

En general es un libro muy adecuado para introducirse en la Ingeniería del Software, entre otras cosas porque es muy fácil de leer, y presenta un conjunto de ejercicios, cuya solución incorpora en un disco flexible. Sus puntos más flojos son que se queda en un nivel introductoria en muchos temas y que comienza a quedar desfasado en algunas de sus partes, siendo muy deseable una nueva edición.

🕮 **Piattini, M. G., Daryanani, S. N.** "*Elementos y Herramientas en el Desarrollo de Sistemas de Información. Una Visión Actual de la Tecnología CASE*". Ra-ma, 1995.

Libro que profundiza en algunos aspectos de las herramientas CASE a la vez que las engloba en un entorno más amplio, con otros aspectos que también persiguen la mejora de la calidad y productividad del desarrollo software (métricas, modelo de proceso software, modelos de evaluación de la capacidad de desarrollo...).

Ha quedado desfasado en algunos temas, pero sigue siendo válido como introducción a la tecnología CASE. Aparece en la bibliografía básica de la asignatura *Ingeniería del Software* como referencia principal del Tema 9 (Herramientas CASE) de esta asignatura.

Cabe destacar que en algún libro, como por ejemplo en la 6ª edición del libro de Ingeniería del Software de Ian Sommerville, el capítulo dedicado a las herramientas CASE ya ha desaparecido, repartiéndose la mención a éstas en el resto de los capítulos.

🕮 **Piattini, M., Villalba, J., Ruiz, F., Bastanchury, T., Polo, M., Martínez, M. Á., Nistal, C.** "*Mantenimiento del Software. Modelos, Técnicas y Métodos para la Gestión del Cambio*". Ra-ma, 2000.

Libro dedicado exclusivamente al mantenimiento de sistemas software. Esta escrito sobre la experiencia de los autores (grupo mixto de profesores universitarios, pertenecientes al Grupo Alarcos de la Universidad de Castilla-La Mancha, y a la industria) al definir una metodología para el mantenimiento de sistemas software MANTEMA.

Es un libro claro, bien escrito y fácil de leer, donde los primeros siete capítulos son referencia básica para el tema 10 de la asignatura *Análisis de Sistemas*, y el resto de capítulos pueden tomarse como bibliografía de consulta para el mismo tema.

📖 **Pfleeger, S. L.** "*Software Engineering. Theory and Practice*". Prentice-Hall, 1998.

Este libro de Ingeniería del Software sirve como bibliografía de consulta para las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*. Aunque es referencia básica en la asignatura *Administración de Proyectos Informáticos*, para la que es especialmente interesante la forma en que Pfleeger aborda el tema de la calidad en diferentes apartados del libro. Destaca el capítulo 11 en el que trata con profundidad la evaluación de procesos, productos y recursos, considerando diferentes enfoques. Discute también diferentes modelos de calidad, analiza las normativas internacionales y hace hincapié en las métricas del software.

📖 **Pressman, R. S.** "*Ingeniería del Software: Un Enfoque Práctico*". 5ª Edición. McGraw-Hill, 2002.

Corresponde a la traducción del libro del mismo autor "*Software Engineering: A Practitioner´s Approach. European Adaptation*", 5th McGraw-Hill, 2000. Este libro es una de las obras clásicas y más utilizadas en la docencia de la Ingeniería del Software. El autor de este Proyecto Docente lo lleva utilizando desde su tercera edición. Esta quinta edición presenta un formato más manejable que la edición anterior. En esta edición el número de capítulos ha aumentado hasta un total de 32, con cinco nuevos capítulos, siendo uno de los más destacados el capítulo 29 dedicado a la Ingeniería Web. El resto de los capítulos han sido revisados incluyendo importantes novedades como los modelos de procesos evolutivos, control estadístico de calidad, COCOMO II, SQA, ingeniería de requisitos, UML, y reglas y teoría de calidad congruentes con la normativa ISO 9000. Con respecto a la cuarta edición el capítulo dedicado al diseño de sistemas de tiempo real ha desaparecido en esta nueva edición.

Este libro explica todos los temas incluidos en la material troncal *Ingeniería del Software* por el Consejo de Universidades de España, así como las unidades SE2 a SE5 del *Computing Curricula 91*.

Resulta de gran valor el conjunto de referencias complementarias que el autor enuncia al final de cada capítulo, así como los recursos en línea que pone al servicio de los lectores en la página http://www.pressman5.com.

Este libro es referencia básica en las tres asignaturas presentadas en este Proyecto Docente e Investigador, siendo bibliografía básica en un alto porcentaje de los temas de ellas.

📖 **Puig, J.** "*Proyectos Informáticos. Planificación, Desarrollo y Control*". Paraninfo, 1994.

Este libro puede ser de utilidad para obtener una visión global de las actividades clásicas de gestión de proyectos que se tratan de una manera concisa y clara. El libro utiliza, en algunos casos, notaciones y terminología obsoleta y no incluye técnicas actuales, por lo que necesitaría una revisión. De todas formas se cita como bibliografía básica de la asignatura *Administración de Proyectos Informáticos*.

📖 **Quang, P., Goñi, J.** "*Dirección de Proyectos Informáticos*". Eyrolles, 1994.

En este libro se contemplan los métodos clásicos de organización de equipos, estimación, planificación, seguimiento y control de calidad de un proyecto. Además se ofrecen una serie de recomendaciones obtenidas de experiencias prácticas. El libro proporciona una visión superficial de los temas de la asignatura *Administración de Proyectos Informáticos*, por lo que se aconseja su uso únicamente en el tema de introducción.

📖 **Romero, C.** "*Técnicas de Programación y Control de Proyectos*". Pirámide, 1997.

Describe de forma detallada tres métodos clásicos de planificación temporal (PERT, CPM y ROY) partiendo de la explicación previa de los conceptos sobre teoría de grafos que son la base de tales métodos. Puede ser de gran ayuda en el estudio del Tema 4 (Planificación temporal de proyectos) de *Administración de Proyectos Informáticos*.

📖 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2ª reimpresión, 1998.

Corresponde a la traducción del libro de los mismos autores "*Object Oriented Modeling and Design*", Prentice Hall, 1991. En el libro se presenta una de las primeras metodologías orientadas al objeto que fueron desarrolladas, OMT. En esta metodología de desarrollo de sistemas se considera que el modelado orientado al objeto consta de tres partes: el modelo objeto, el modelo funcional y el modelo dinámico. A pesar de ser una metodología superada en muchos aspectos, desde el punto de vista de la docencia a impartir tiene como ventaja que toma algunas de las herramientas de las metodologías clásicas, por ejemplo el DFD para el elaborar el modelo funcional, lo que permite realizar una transición suave desde los métodos clásicos a los de la orientación al objeto. La tercera parte del libro aborda los aspectos de paso del diseño a la implementación en relación con los lenguajes de programación y las bases de datos relacionales. Incluye la descripción de tres casos de estudio.

Es referencia básica tanto en la asignatura *Ingeniería del Software* como en la asignatura *Análisis de Sistemas*, en primer ciclo es imprescindible porque su presentación del

modelo objeto es de las mejores, y además OMT es el enfoque sistemático más utilizado por los alumnos de primer ciclo que, habiendo elegido un enfoque orientado a objetos, realizan su proyecto de final de carrera. En la asignatura de segundo ciclo su importancia es más colateral, pero aún así sigue siendo una buena referencia para comprender las bases y evolución del lenguaje de modelado UML, así como para comparar este método de desarrollo con UP (*Unified Process*).

🕮 **Rumbaugh, J., Jacobson, I., Booch, G.** "*El Lenguaje Unificado de Modelado. Manual de Referencia*". Addison-Wesley, 2000.

Traducción del libro de los mismos autores "*The Unified Modeling Language. Reference Manual*", Addison-Wesley, 1999. Es en opinión del que suscribe este proyecto docente uno de los mejores libros de referencia y consulta sobre UML, al detallar, en forma de diccionario, la semántica de cada uno de los elementos que aparecen en UML versión 1.3.

El libro tiene cuatro partes, las dos primeras son introductorias, y la cuarta son los apéndices, pero la que realmente supone el cuerpo central del libro es la tercera parte, donde se incluye la enciclopedia de términos.

Es una referencia básica para las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*, en cualquier tema relacionado con UML (*Unfied Modeling Language*), pero es especialmente importante en el Tema 7 (El lenguaje UML y el proceso unificado) de *Análisis de Sistemas.*

🕮 **Sommerville, I.** "*Ingeniería de Software*". 6ª Edición, Addison-Wesley, 2002.

Traducción del libro del mismo autor "*Software Engineering*", 6th Edition. Addison-Wesley, 2001. Es otro de los libros clásicos sobre Ingeniería del Software. El que suscribe este proyecto docente lo lleva utilizando desde su segunda edición. La sexta edición se ha reorganizado y, en palabras del autor, se ha reducido un 10% con respecto a la quinta edición, así la información sobre métodos formales se ha reducido a un capítulo y el material sobre herramientas CASE se ha vistos también mermado y distribuido en varios capítulos. Hay capítulos nuevos que cubren el proceso software, la arquitectura de sistemas distribuidos, la confiabilidad y los sistemas heredados. También hay cambios en la forma de tratar los requisitos, repartidos en un capítulo sobre requisitos y otro sobre ingeniería de requisitos, la reutilización se enfoca desde el diseño con reutilización y patrones software, se incluye COCOMO II, y como lenguaje de modelado para los ejemplos se utiliza UML. Así, el libro se organiza en siete partes: introducción a la

Ingeniería del Software, especificación, diseño, desarrollo de sistemas críticos, verificación y validación, administración, y evolución del software.

Este libro cubre todas la materia que aconseja el área de conocimiento de Ingeniería del Software en el *Computing Curricula 2001* para Ciencia de la Computación, ajustándose también a las recomendaciones del SWEBOK de IEEE-CS. Esto lo convierte en bibliografía básica para las tres asignaturas presentadas en este proyecto docente.

A juicio del que suscribe este Proyecto Docente, es un libro que no sustituye plenamente a un libro como "Ingeniería del Software. Un Enfoque Práctico" de R. S. Pressman, sino que más bien lo complementa en alguno de sus temas, por ejemplo se destaca el tratamiento que hace del tema de los requisitos, el diseño con reutilización, la parte de pruebas y la parte dedicada a la evolución del software.

Esta última edición se ha traducido al español después de varias ediciones sin traducir (aunque la traducción no sea lo mejor de esta edición) lo que facilita su acceso al alumno. Además, en la URL http://www.software-engin.com se tienen enlaces a gran cantidad de material de apoyo al libro y sobre Ingeniería del Software en general.

📖 **Stevens, P., Pooley, R.** "*Utilización de UML en Ingeniería del Software con Objetos y Componentes*". Addison-Wesley, 2002.

Traducción del libro de los mismos autores "*Using UML Software Engineering with Objects and Components*", Addison-Wesley, 1999. Este libro está organizado en cuatro partes. La primera parte introduce los conceptos de Ingeniería del Software, incluyendo la coletilla de "con componentes", y de desarrollo orientado a objetos. La segunda parte hace un recorrido por los diagramas de UML en su versión 1.3. La tercera parte es quizás la más interesante al presentar tres casos prácticos. Y por último la cuarta parte sirve como cajón de sastre donde incluir temas como la reutilización y la calidad.

📖 **Ward, P. T., Mellor, S. J.** "*Structured Development for Real-Time Systems*". Yourdon Press/Prentice-Hall, 1985.

Este libro presenta las técnicas y procedimientos correspondientes a una de las primeras extensiones para sistemas de tiempo real que se realizaron de los métodos estructurados de análisis y diseño. El libro consta de tres volúmenes. En el primero se describen los conceptos y las herramientas de modelado para representar las tres vistas del sistema: la estática, la dinámica y la de eventos. En el segundo se describe como utilizar estas herramientas para construir los modelos del sistema: el de contexto, el esencial y el de comportamiento. Este volumen se completa con la exposición de algunos casos de

ejemplo. En el tercero se exponen las técnicas para construir los modelos de implementación. Los tres volúmenes mantienen una secuencia lógica, y tienen numerosos ejemplos que ayudan a comprender el proceso completo de análisis y diseño de un sistema de tiempo real.

No es bibliografía básica de ninguna de las asignaturas presentadas, pero sí lo es de algunos temas concretos de las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*, por ejemplo del Tema 2 (Análisis y Especificación de Requisitos) de *Ingeniería del Software*, y del Tema 9 (Desarrollo de sistemas complejos) de *Análisis de Sistemas*; apareciendo además como bibliografía complementaria en otros temas.

🕮 **Yourdon, E.** "*Análisis Estructurado Moderno*". Prentice-Hall Hispanoamericana, 1993.

Corresponde a la traducción del texto del mismo autor "*Modern Structured Analysis*". Prentice-Hall, 1989. En él, Yourdon revisa las teorías del análisis estructurado desarrolladas por DeMarco en el año 1979. Es un libro con un diseño muy didáctico y adaptado a un curso de análisis estructurado. Tiene algunos capítulos, que en su momento constituyeron un cierto avance en las metodologías estructuradas y que, a pesar del paso de los años, mantienen su vigencia. Es de destacar el capítulo en el que se realiza un estudio general de sistemas, la propuesta de los diferentes modelos a realizar de un sistema y los casos prácticos desarrollados en los apéndices. Sigue constituyendo uno de los libros de referencia obligada para los cursos de ingeniería del software, así aparece citado en la bibliografía básica de las asignaturas *Ingeniería del Software* y *Análisis de Sistemas*.

🕮 **Yourdon Inc.** "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions. 1993.

Constituye un manual de referencia en el que se describe completamente la metodología "*The Yourdon System Method (YSM)*" y su utilización en el desarrollo de sistemas software. Describe completamente los modelos, herramientas y métodos que componen el YSM. Dada su estructura de manual de referencia la descripción de las herramientas de modelado y de los modelos se realiza en apartados independientes y sin una secuencia "formativa" específica. No es un libro diseñado para la formación en la metodología. Está diseñado para que sirva de referencia de las herramientas, los modelos o las técnicas de la metodología. En este sentido es un excelente libro de consulta para los alumnos. Se recomienda sobre todo la parte 3, en la que se describen los modelos, porque aporta buenas propuestas acerca de los conjuntos de documentación que se deben elaborar para la especificación de los diferentes modelos del sistema.

Sólo es bibliografía básica en el Tema 3 (Análisis estructurado) de la asignatura *Ingeniería del Software*, aunque aparece como bibliografía complementaria en diversos temas.

Apéndice C:
Fuentes de información sobre Ingeniería del Software

Este apartado ofrece una panorámica de las principales fuentes de información sobre la materia objeto de la plaza a concurso. Se incluye información de publicaciones periódicas, jornadas y congresos, sitios web y grupos de noticias.

C.1 Revistas

Las revistas que se enumeran a continuación tienen en sus contenidos artículos relacionados con la Ingeniería del Software.

- **ACM Computing Surveys**. Disponible desde el volumen 1 (1969) en la biblioteca digital de ACM (http://portal.acm.org/).
- **ACM Crossroads**. Disponible desde el volumen 1 (1994) en la biblioteca digital de ACM (http://portal.acm.org/).
- **ACM SIGCHI Bulletin** (http://www.acm.org/sigchi/bulletin/)**.**
- **ACM SIGSOFT Software Engineering Notes**. (http://www.acm.org/sigs/sigsoft/SEN/).
- **ACM TOSEM (Transaction On Software Engineering and Methodology)**. Disponible desde el volumen 1 (1992) en la biblioteca digital de ACM (http://portal.acm.org/).
- **Acta Informatica.** Disponible en línea desde el volumen 33 (1996) en la biblioteca digital de Springer (http://link.springer.de/).

- **Annals of Software Engineering**. Disponible en línea desde el volumen 3 (1997) en la biblioteca digital de Kluwer Academic Publisher (Kluwer Online) (http://www.kluweronline.com/).
- **Autonomous Agents and Multi-Agent Systems**. Disponible en línea desde el volumen 1 (1998) en la biblioteca digital de Kluwer Academic Publisher (Kluwer Online) (http://www.kluweronline.com/).
- **Communications of the ACM**. Disponible desde el volumen 1 (1958) en la biblioteca digital de ACM (http://portal.acm.org/).
- **Computers & Education**. Disponible desde el volumen 22 (1995) en la biblioteca digital de Elsevier Science Ltd. (http://www.sciencedirect.com).
- **Crosstalk. The Journal of Defense Software Engineering**. Disponible desde el volumen 7 (1994) en http://stsc.hill.af.mil/crosstalk/xtalk.asp.
- **C/C++ Users Journal** (http://www.cuj.com).
- **Dr. Dobb's Journal** (http://www.ddj.com).
- **IBM Systems Journal** (http://www.research.ibm.com/journal/).
- **IEEE Computer**. Disponible desde el volumen 21 (1988) en la biblioteca digital de IEEE (http://computer.org/computer/).
- **IEEE Internet Computing**. Disponible desde el volumen 1 (1997) en la biblioteca digital de IEEE (http://computer.org/internet/).
- **IEEE ITPro**. Disponible desde el volumen 1 (1999) en la biblioteca digital de IEEE (http://computer.org/itpro/).
- **IEEE Software**. Disponible desde el volumen 5 (1988) en la biblioteca digital de IEEE (http://computer.org/software/).
- **IEEE TSE (Transactions on Software Engineering)**. Disponible desde el volumen 21 (1995) en la biblioteca digital de IEEE (http://computer.org/tse/).
- **Information and Software Technology**. Disponible desde el volumen 37 (1995) en la biblioteca digital de Elsevier Science Ltd. (http://www.sciencedirect.com).
- **Information Systems**. Disponible desde el volumen 20 (1995) en la biblioteca digital de Elsevier Science Ltd. (http://www.sciencedirect.com)
- **International Journal of Software Engineering and Knowledge - IJSEKE** (World Scientific Publishing Company).
- **Java Report**. Índices y algunos artículos en línea en http://www.javareport.com/.
- **JavaWorld**. Revista online en http://www.javaworld.com/.
- **JOOP (Journal of Object-Oriented Programming)**. Índices y algunos artículos en línea en http://www.joopmag.com/.
- **Journal of Software Maintenance: Research and Practice** (http://www3.interscience.wiley.com).

- **Journal of Systems and Software**. Disponible desde el volumen 28 (1995) en la biblioteca digital de Elsevier Science Ltd. (http://www.sciencedirect.com).
- **Lecture Notes on Computer Science**. Disponibles en línea en la biblioteca digital de Springer (http://link.springer.de/).
- **Novática** (http://www.ati.es/).
- **Requirements Engineering**. Disponible en línea desde el volumen 3 (1998) en la biblioteca digital de Springer (http://link.springer.de/).
- **SEI Interactive** (http://interactive.sei.cmu.edu/home.htm).
- **Software Development**. Artículos en línea en (http://www.sdmagazine.com/).
- **Software--Practice and Experience** (John Wiley & Sons).
- **Theory and Practice of Object Systems – TAPOS** (John Wiley & Sons).

C.2 Jornadas y congresos

C.2.1 Nacionales

- **Congreso de Tecnología de Objetos.**
- **Jornadas de Ingeniería del Software – JIS**. Vienen realizándose anualmente desde 1996 [Toro, 1996; Díaz y Lopistéguy, 1997; Toval y Nicolás, 1998; Botella et al., 1999; Delgado et al., 2000; Díaz et al., 2001]. Se ha consolidado como el foro nacional de encuentro y discusión de los Grupos de Investigación y Docencia en Ingeniería del Software. En la edición celebrada en noviembre de 1999 en Cáceres, se unieron las Jornadas de Ingeniería del Software y de Bases de Datos.
- **Jornadas de Investigación y Docencia en Bases de Datos – JIDBD**. Vienen celebrándose anualmente desde 1996.
- **Jornadas de Trabajo DOLMEN**. El proyecto coordinado DOLMEN (proyecto CICYT TIC2000-1673-C06-01) está dando como resultado una serie de reuniones entre los Grupos de investigación integrantes del mismo desde junio de 2001 hasta la fecha.
- **Jornadas de Trabajo MENHIR**. El proyecto coordinado MENHIR (proyecto CICYT TIC97-0593-C05-05) ha dado como resultado una serie de reuniones entre los Grupos de investigación integrantes del mismo desde noviembre de 1997 hasta marzo de 2000.
- **Jornadas sobre Calidad del Software**.
- **Simposio Español de Informática Distribuida (SEID)** y posteriormente **Simposio en Informática y Telecomunicaciones (SIT)**.

C.2.2 Internacionales

- **ACM Annual Symposium on Principles of Programming Languages – POPL.**
- **ACM Conference on Object Oriented Programming Systems Languages and Aplications – OOPSLA.**
- **ACM Foundations of Software Engineering – ACM SIGSOFT.**
- **Asia Pacific Conference on Pattern Languages of Programs – KoalaPLoP.**
- **Conference on Human Factors in Computing Systems – CHI.**
- **Conference on Object-Oriented Programming Systems, Languages, and Applications – OOPSLA.**
- **Conference on Pattern Languages of Programs – PLoP.**
- **European Conference on Object-Oriented Programming – ECOOP.**
- **European Conference on Pattern Languages of Programming and Computing – EuroPLoP.**
- **IEEE Asia Pacific Software Engineering Conference – APSEC.**
- **IEEE Australian Software Engineering Conference – ASWEC.**
- **IEEE Basque International Workshop on Information Technology – BIWIT.**
- **IEEE Conference on Software Engineering Education & Training - CSEE&T.**
- **IEEE European Conference on Software Maintenance and Reengineering – CSMR.**
- **IEEE International Conference on Requirements Engineering – ICRE.**
- **IEEE International Conference on Software Maintenance – ICSM.**
- **IEEE International Conference on Software Reuse – ICSR.**
- **IEEE International Symposium on Software Engineering Standards – ISESS.**
- **IEEE International Symposium on Object-Oriented Real-Time Distributed Computing – ISORC.**
- **IEEE International Symposium on Software Engineering for Parallel and Distributed Systems – PDSE.**
- **IEEE International Symposium on Requirements Engineering – RE.**
- **IEEE Real-Time Technology and Applications Symposium – RTAS.**
- **IEEE International Conference on Real-Time Computing Systems and Applications – RTCSA.**
- **IEEE Real-Time Systems Symposium – RTSS.**
- **International Conference on Software Engineering – ICSE.**
- **Jornadas Iberoamericanas de Ingeniería del Requisitos y Ambientes Software – IDEAS**.
- **Software Engineering & Knowledge Engineering – SEKE.**

- **Technology of Object-Oriented Languages and Systems – TOOLS (TOOLS Europe; TOOLS USA; TOOLS Pacific).**
- **<<UML>> Conference.**

C.3 Sitios web

Son innumerables los servidores en Internet con información relacionada con la Ingeniería del Software. Se pueden encontrar libros, documentos, herramientas, lenguajes, datos reales de proyectos a gran escala, estándares...

Ofrecer un listado exhaustivo de todos estos lugares sería impracticable, y su utilidad sería relativa por la volatilidad que presentan algunos de ellos. Por este motivo se ha hecho una selección de aquellos sitios web más importantes y más estables.

- **Association for Computing Machinery (ACM):** http://www.acm.org. Fundada en 1947 fue la primera sociedad científica y de educación del mundo. El portal de información que presenta es impresionante, tanto en cuanto a enlaces de interés, grupos de trabajo, documentos electrónicos, conferencias como por su biblioteca digital conteniendo revistas y actas de congresos (http://portal.acm.org).
- **Center for Software Engineering (USC):** http://sunset.usc.edu/. Sitio donde se canaliza la información de COCOMO (documentación, herramientas, artículos...).
- **Cetus Links - Object-Orientation:** http://www.cetus-links.org/. La colección de enlaces más completa sobre Orientación a Objetos. A fecha de 22 de enero de 2002 se tenían contabilizados 18309 enlaces.
- **Consejo Superior de Informática:** http://www.map.es/csi/csi.htm. Web del Consejo Superior de Informática donde, entre otras cosas, se puede obtener la documentación y herramientas sobre Métrica 3.
- **DACS (Data & Analysis Center for Software):** http://www.dacs.dtic.mil/. Sitio perteneciente al DoD de EEUU con numerosa información y enlaces a lugares relacionados con la Ingeniería del Software.
- **European Software Institute (ESI):** http://www.esi.es. El Instituto del Software Europeo tiene su sede en Bilbao (España). Dispone tanto de documentación privada para los miembros del instituto como documentación pública con los análisis de proyectos, necesidades de empresas y software europeo.

- **Guide to the Software Engineering Body of Knowledge (SWEBOK):** http://www.swebok.org/. Proyecto para establecer un cuerpo de conocimiento común para la Ingeniería del Software.
- **Institute of Electrical and Electronics Engineers (IEEE):** http://www.ieee.org. Otra prestigiosa organización compuesta por diversas sociedades, donde la que más relación tiene con los temas abordados en el presente Proyecto Docente es la IEEE Computer Society (http://computer.org). A semejanza de ACM, ofrece información sobre conferencias, estándares, educación y mantiene otra biblioteca digital con revistas y actas de congresos.
- **Object Management Group (OMG):** http://www.omg.org. Es un consorcio internacional de industrias con el fin de promover el uso de la Orientación a Objetos en la Ingeniería del Software. A diferencia de organizaciones como ISO o IEEE, OMG desarrolla estándares de "*facto*" como consenso entre las empresas que la forman. Dicho servidor ofrece publicaciones electrónicas y enlaces a estándares y herramientas del sector relacionado con la tecnología de objetos.
- **Rational Software Corporation:** http://www.rational.com. Sitio web de la empresa responsable de UML. En esta dirección se tiene valiosa información sobre UML y RUP (documentos, informes, artículos, presentaciones, bibliografía recomendada...). Además, se pueden obtener versiones de demostración de diferentes herramientas que comercializan, siendo Rational Rose la más difundida.
- **R. S. Pressman & Associates, Inc.:** http://www.rspa.com. Bajo la dirección de Roger S. Pressman y la difusión internacional de su libro [Pressman, 2000], cuya quinta edición está actualmente en desarrollo, aparece una empresa de consultoría en Ingeniería del Software. Lo más interesante que ofrece esta dirección es un portal que da entrada a otras fuentes de información relacionadas con cada uno de los capítulos tratados en su libro.
- **REMIS:** http://www.sc.ehu.es/jiwdocoj/remis/remis.htm. Red sobre Experimentación y Medición en Ingeniería de Software.
- **Software Engineering Education Resource Site (SEERS):** http://www.computing.db.erau.edu/SEERS/. Lugar donde se discute sobre el cuerpo de conocimiento de la Ingeniería del Software.
- **Software Engineering Institute (SEI):** http://www.sei.cmu.edu. El Instituto de Ingeniería del Software en la Universidad Carnegie Mellon, es uno de los lugares

más activos en pro de la Ingeniería del Software. Se pueden encontrar documentos asociados a módulos curriculares en Ingeniería del Software, informes técnicos sobre diferentes áreas de la Ingeniería del Software, una revista en línea... Incluye además enlaces a otras organizaciones relacionadas con la Ingeniería del Software.

- **The NATO Software Engineering Conferences:** http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO/. Sitio donde se encuentran en formato PDF las actas de las dos famosas conferencias de la NATO sobre Ingeniería del Software, celebradas en Garmisch (Alemania) y en Roma (Italia) en 1968 y 1969 respectivamente**.**
- **The Pedagogical Patterns Project:** http://www-lifia.info.unlp.edu.ar/ppp/. Proyecto de patrones pedagógicos. Aplicación de la teoría de patrones al terreno de la pedagogía.
- **The World Wide Web Consortium:** http://www.w3.org/. Sitio central donde se recogen las especificaciones de los diversos lenguajes relacionados con la web (HTML, XML, RDF, SOAP...).
- **The World Wide Web Virtual Library Formal Methods:** http://www.afm.sbu.ac.uk/. Sitio especializado en información sobre métodos formales.
- **UML Revision Task Force:** http://www.celigent.com/omg/umlrtf/. Lugar donde aparecen y se recogen las diferentes versiones de la documentación de UML, así como la progresión de esta iniciativa.
- **Webliography Software Engineering:** http://polaris.umuc.edu/~skerby/help/wbib_swe.htm. Colección de enlaces sobre Ingeniería del Software.
- **WWW Virtual Library - Software Engineering:** http://mingo.info-science.uiowa.edu/soft-eng/. Colección de enlaces sobre Ingeniería del Software.

C.4 Grupos de noticias

C.4.1 Ingeniería del Software

- comp.software-eng
- comp.specification
- comp.testing

C.4.2 Orientación a Objetos

- comp-databases.object
- comp.object
- comp.object.corba

C.5 Referencias

[Botella et al., 1999] Botella, P., Hernández, J., Saltor, F. (Editores). "*Actas de las IV Jornadas de Ingeniería del Software y Bases de Datos (JISBD'99)*". Grupo de paralelismo del Departamento de Informática de la Universidad de Extremadura. Cáceres, 24-26 de noviembre de 1999.

[Delgado et al., 2000] Delgado, C., Marcos, E., Marqués, J. M. (Editores). "*Actas de las V Jornadas de Ingeniería del Software y Bases de Datos*". Departamento de Informática y Automática de la Universidad de Valladolid. Valladolid, 8-10 de noviembre de 2000.

[Díaz et al., 2001] Díaz, O., Illarramendi, A., Piattini, M. (Editores). "*Actas de las VI Jornadas de Ingeniería del Software y Bases de Datos*". Grupo Alarcos del Departamento de Informática. Escuela Superior de Informática de la Universidad de Castilla-La Mancha. Almagro (Ciudad Real), 21-23 de noviembre de 2001.

[Díaz y Lopistéguy, 1997] Díaz, O., Lopistéguy, P. (Editores). "*Actas de las II Jornadas de Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad del País Vasco/Euskal Herriko Unibertsitatea. Donosita-San Sebastián, 2-5 de septiembre de 1997.

[Pressman, 2000] Pressman, R. S. "*Software Engineering: A Practitioner's Approach – European Adaptation*". 5th Edition. McGraw-Hill, 2000.

[Toro, 1996] Toro Bonilla, M. (Editor). "*Actas de las I Jornadas de Trabajo en Ingeniería del Software*". Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Sevilla, 14-15 de noviembre de 1996.

[Toval y Nicolás, 1998] Toval Álvarez, A., Nicolás Ros, J. (Editores). "*Actas de las III Jornadas de Ingeniería del Software*". Departamento de Informática, Lenguajes y Sistemas de la Universidad de Murcia. Murcia, 11-13 de noviembre de 1998.

Apéndice D: Acrónimos

1FN	**Primera Forma Normal.**
2FN	**Segunda Forma Normal.**
3FN	**Tercera Forma Normal.**
4FN	**Cuarta Forma Normal.**
4GL	**Fourth Generation Language.**
5FN	**Quinta Forma Normal.**
ABF	**Applications By Forms.**
ACF	**Architecture Flow Diagram.**
ACF	**Asset Certification Framework.**
ACM	**Association for Computing Machinery.**
ACTI	**Abstract and Concrete Templates and Instances.**
ADAGE	**Avionics Domain Application Generation Environment.**
ADL	**Architecture Description Language.**
ADOO	**Análisis y Diseño Orientado al Objeto.**
ADP	**Acyclic Dependencies Principle.**
ADP	**Automated Data Processing.**
ADS	**Application Development Strategies.**
ADT	**Abstract Data Type.**
ADT	**Application Development Trends.**
AECC	**Asociación Española para el Control de Calidad.**
AEIA	**Asociación Española de Informática y Automática.**
AENOR	**Asociación Española de Normalización y Certificación.**
AF	**Applications Frames.**
AF	**Auditoría Funcional.**
AFFTC	**Air Force Flight Test Centre.**
AFI	**Auditoría Física.**
AHP	**Analytic Hierarchy Process.**
AI	**Artificial Intelligence.**
AII	**Asociación de Ingenieros en Informática.**
AIP	**Atributo Identificador Principal.**
AIS	**Automated Information Systems.**
AKO	**A Kind Of.**
ALF	**Asset Library Framework.**
ALGOL	**ALGOrithmic Language.**
ALI	**Asociación de Doctores, Licenciados e Ingenieros en Informática.**
ALOAF	**Asset Library Open Architecture Framework.**
ALS	**ADA Language System.**
ANSI	**American National Standard Institute.**
AO	**Application Objects.**
AOO	**Análisis Orientado al Objeto.**
APD	**Agencia de Protección de Datos.**
API	**Application Programming Interface.**
APO	**A Part Of.**
ARC	**Army Reuse Center.**
ARM	**Annotated Reference Manual.**
ARPANet	**Advanced Research Project Agency.**
ARS	**Análisis Requisitos del Sistema.**
ARSi	**Actividad i del Módulo de Análisis de Requisitos del Sistema.**
ASC	**Aeronautical Systems Center.**
ASCII	**American Standard Code for Information Interchange.**
ASIS	**Ada Semantic Interface Specification.**
ASSET	**Asset Source for Software Engineering Technology.**

ATI **Asociación de Técnicos en Informática.**
ATIS **A Tool Interface Standard.**
ATL **Active Template Library.**
ATM **Asynchronous Transfer Mode.**
AVEIN **Asociación Vallisoletana de Empresas de Informática.**
AWT **Abstract Windows Toolkit.**
BAe **British Aerospace.**
BBS **Bulletin Board System.**
BCNF **Boyce/Codd Normal Form.**
BD **Bases de Datos.**
BDE **Borland Database Engine.**
BDI **Believe, Desire and Intention.**
BDK **Beans Development Kit.**
BDOO **Bases de Datos Orientadas a Objetos.**
BIDM **Basic Interoperability Data Model.**
BIT **Binary digIT.**
BITNET **Because IT's time NETwork.**
BLOB **Bynary Large Object.**
BLOOM **BarceLona Object Oriented Model.**
BNF **Backus Naur Form.**
BOA **Business Object Architecture.**
BOE **Boletín Oficial del Estado.**
BOF **Business Object Facility.**
BOM **Business Object Model.**
BON **Business Object Notation.**
BOO **Behavioural Object-Oriented.**
BPR **Business Process Reengineering.**
BQL **Binary Query Language.**
BRM **Binary Relations Model.**
BSA **Business Software Alliance.**
BSI **British Standards Institute.**
BTW **By The Way.**
C/S **Cliente/Servidor.**
CA **Certification Authority.**
CACM **Communications of the ACM.**
CACS **Computer Audit, Control and Security Conferences.**
CAD **Computer Aided Design.**
CADF **Committee for Advanced Database Function.**
CADMAT **Computer Aided Design Manufacturing and Testing.**
CADSAT **Computer Aided Design and Specification Analysis Tool.**
CAI **Computer Assisted Instruction.**
CAL **Computer Aided Learning.**
CAM **Computer Aided Manufacturing.**
CAMAC **Computer Automated Measurement and Control.**
CAMP **The Common Ada Missile Packages project.**
CARDS **Comprehensive Approach to Reusable Defense Software**
CARE **Computer-Aided Requirements Engineering.**
CAS **Commercially Available Software.**
CASE **Computer Aided/Assisted Software/System Engineering.**
CASRE **Computer Aided Software Reliability Estimation.**
CBD **Component Based Development.**
CBL **Computer Based Learning.**
CBO **Common Business Objects.**
CBO **Coupling Between Object classes.**
CBS **Compound Boolean Selection.**
CBSE **Component-Based Software Engineering.**
Cc **Código Civil.**
CCA **Critical Capability Area.**
CCB **Change Control Board.**
CCC **Comité de Control de Cambios.**
CCCP **Cliente-Canal-Compañía-Producto.**
CCITT **Consultative Committee International Telephone and Telegraph.**
CCP **Common Closure Principle.**
CCTA **Central Computing and Telecommunications Agency.**
CD **Compact Disc.**
CDE **Common Desktop Environment.**
CDIF **CASE Data Interchange Format.**
CDM **Common Data Model.**
CDR **Critical Design Review.**
CDRL **Contract Data Requirements List.**
CE **Comunidad Europea.**
CEC **Configuración de Estado de la Configuración.**
CEF **Centro de Estudios Financieros.**
CEN **Comisión Europea de Normalización.**
CEPIS **Council of European Professional Informatics Societies.**
CETE **Centre d'Etudes Techniques de l'Equipement,**
CF **Common Facilities.**
CFD **Control Flow Diagram.**
CFE **Control Flow Editor.**
CFRP **Conceptual Framework for Reuse Processes.**
CGI **Common Gateway Interface.**
CICS **Customer Information Control System.**
CIM **Computer Integrated Manufacturing.**
CISA **Certified Information Systems Auditor.**
CLLCM **Clear Lake Lifecycle Model.**
CLOS **Common Lisp Object System.**
CLP **Construcción Lógica de Programas.**
CM **Configuration Management.**
CMM **Capability Maturity Model.**

COBOL **COmmon Bussines Oriented Language.**
COCOMO **COnstructive COst MOdel.**
CODASYL **Conference On DAta SYstems Languages.**
COM **Component Object Model.**
COMMA **Common Object-oriented Methodology Metamodel Architecture.**
COMN **Common Object Modelling Notation.**
COP **Component-Oriented Programming.**
CORBA **Common Object Request Broker Architecture.**
COSE **Common Open Software Environment.**
COTS **Commercial Off-The-Shelf.**
CP **Código Penal.**
CPD **Centro de Proceso de Datos.**
CPI **Centro Proveedor de Información.**
CPM **Critical Path Method.**
CPN **Coloured Petri Net.**
CPU **Central Processing Unit.**
CR **Configuración de Referencia.**
CRC **Cyclic Redundancy Checking.**
CRC **Class, Responsability and Collaboration.**
CS&S **Computer Systems and Software.**
CSC **Computer Software Component.**
CSCI **Computer Software Configuration Item.**
CSF **Critical Success Factors.**
CSIC **Consejo Superior de Investigaciones Científicas.**
CSPEC **Control SPECification.**
CSU **Computer Software Unit.**
CTI **Centre Technique d'Informatique.**
CTN **Comité Técnico Nacional.**
CUT **Classes Under Test.**
CWA **Closed World Assumption.**
DA **Data Administrator.**
DA **Domain Analysis.**
DACS **Data & Analysis Center for Software.**
DAFTG **Database Architecture Framework Task Group.**
DAG **Directed Acyclic Graph.**
DALE **Database Access Library for Eiffel.**
DARPA **Defense Advanced Research Project Agency.**
DARTS **Design Approach for Real-Time Systems.**
DAS **Documento de Análisis del Sistema.**
DBA **Database Administrator.**
DBCC **Database Consistency Checker.**
DBMS **Data Base Management System.**
DBSSG **Database System Study Group.**
DBTG **Data Base Task Group.**
DCC **Diagrama de Configuración de Clases.**
DCE **Diagrama de Correspondencia del Efecto.**
DCE **Distributed Computing Environment.**
DCL **Data Control Language.**
DCMC **Defense Contract Management Command.**
DCOM **Distributed Component Object Model.**
DCS **Módulo de Desarrollo de Componentes del Sistema.**
DCSi **Actividad i del Módulo de Desarrollo de Componentes del Sistema.**
DD **Data Dictionary.**
DDBMS **Distributed Data Base Management System.**
DDC **Display Data Channel.**
DDF **Diagramas de Descomposición Funcional.**
DDL **Data Definition Language.**
DDM **Data Dictionary Manager.**
DDS **Data Dictionary System.**
DDS **Digital Data Storage.**
DE/R **Diagrama de Entidad-Interrelación.**
DEC **Diagrama de Estados de Clases.**
DED **Diagrama de Estructura de Datos.**
DEL **Diagrama de Línea de Ensamblaje.**
DER **Diagrama de Entidad-Interrelación.**
DES **Data Encryption Standard.**
DF **Dependencia Funcional.**
DFD **Diagrama de Flujo de Datos.**
DFE **Data Flow Editor.**
DFOQ **Design For Ownership Quality.**
DFS **Distributed File System.**
DIB **Device Independent Bitmap.**
DIO **Diagrama de Interacción entre Objetos.**
DIP **Dependency Inversion Principle.**
DIS **División de Ingeniería del Software.**
DIT **Depth of Inheritance Tree.**
DL **Data Language.**
DLL **Dynamic-Link Libraries.**
DMA **Direct Memory Access.**
DMCS **Data Mapping Control System.**
DML **Data Manipulation Language.**
DMT **Data Management Tool.**
DNS **Domain Name System.**
DoD **Departament of Defense.**
DOE **Distributed Object Everywhere.**
DOLMEN **Objetos Distribuidos, Lenguajes, Modelos y ENtornos.**
DOME **Distributed Object Management Everywhere.**
DOO **Diseño Orientado al Objeto.**
DOR **Domain Oriented Reuse.**
DPU **Módulo de Desarrollo de Procedimiento de Usuario.**
DPUi **Actividad i del Módulo de Desarrollo de Procedimiento de Usuario.**
DR **Deficiency Report.**
DRA **Desarrollo Rápido de Aplicaciones.**

DRI **Diccionario de Recursos de Información.**
DRS **Documento de Requisitos del Sistema.**
DSDL **Data Storage Definition Language.**
DSE **Data-Structure Editor.**
DSED **Desarrollo de Sistemas Estructurados en Datos.**
DSI **Director de Sistemas de Información.**
DSIC **Departamento de Sistemas Informáticos y Computación.**
DSMC **Defense Systems Management College.**
DSN **Data Source Name.**
DSOM **IBM's Distributed Systems Object Model.**
DSS **Decision Support System.**
DSSA **Domain-Specific Software Architecture.**
DTC **Distributed Transaction Coordinator.**
DTE **Diagrama de Transición de Estados.**
DTIC **Defense Technical Information Center.**
DTS **Módulo de Diseño Técnico del Sistema.**
DTSi **Actividad i del Módulo de Diseño Técnico del Sistema.**
DVD **Digital Versatile Disk.**
DW **Data Warehouse.**
E/R **Entity-Relationship.**
E/S **Entrada/Salida.**
EAP **Experimental Aircraft Programme.**
ECA **Event-Condition-Action rules.**
ECMA **European Computer Manufacturer's Association.**
ECS **Embedded Computer Systems.**
EDCS **Evolutionary Design of Complex Systems.**
EDI **Electronic Data Interchange.**
EDIF **Electronic Data Interchange Format.**
EDS **Entornos de Desarrollo de Software.**
EET **Earliest Even Time.**
EFF **Electronic Frontier Foundation.**
EFS **Especificación Funcional del Sistema.**
EFSi **Actividad i del Módulo de Especificación Funcional del Sistema.**
EIA **Electronic Industries Association.**
EIS **Executive Information System.**
EITO **European Information Technology Observatory.**
ELH **Entity Life History.**
ELP **Estructura Lógica del Proceso.**
ELS **Estructura Lógica de la Salida.**
ELSA **Electronic Library Services and Applications.**
EMS **Enterprise Messaging Server.**
ENCORE **Extensible and Natural Common Object REsource.**
ERD **Entity-Relationship Diagram.**
ERE **Entity-Relationship Editor.**
EROOS **Entity-Relationship Object-Oriented Specifications.**
ERS **Especificación de Requisitos del Software.**
ESA **European Space Agency.**
ESI **European Software Institute.**
ESPRIT **European Strategic Programme for Research and development in Information Technology.**
ESSI **European Systems & Software Initiative.**
ETS **Educational Testing Service.**
EUROWARE **Enabling Users to Reuse Over Wide AREas.**
F^3 **From Fussy to Formal.**
FA **Factor de Ajuste.**
FAQ **Frequently Asked Questions.**
FBE **Framework-Based Environment.**
FC **Factor de Complejidad.**
FCM **Factor Criteria Metrics Model.**
FCT **Factor de Complejidad Técnica.**
FD **Flujo de Datos.**
FESABID **Federación Española de Sociedades de Archivística, Biblioteconomía y Documentación.**
FESI **Federación Española de Sociedades de Informática.**
FH **Funcionalidad Hipermedia.**
FI **Facultad de Informática.**
FIFO **First In First Out.**
FMOODS **Formal Methods for Open Object-based Distributed Systems.**
FN **Forma Normal.**
FNBC **Forma Normal de Boyce/Codd.**
FODA **Feature-Oriented Domain Analysis.**
FOOD **Functional Object-Oriented Design.**
FOOL **Foundations of Object-Oriented Languages.**
FORTRAN **FORmula TRANslator.**
FP **Fundamental Principles.**
FPA **Function Point Analysis.**
FS **Feasibility Study.**
FSM **Finite State Machines.**
FTP **File Transfer Protocol.**
FYI **For Your Information.**
F-ORM **Functionality in the Objects with Roles Model.**
GAD **Grafo Acíclico Dirigido.**
GAF **Generic Application Frame.**
GCS **Gestión de la Configuración del Software.**
GDI **Graphics Device Interface.**
GEARS **Gaining Efficiency and quAlity in Real time control Software.**
GI **Grado de Influencia.**
GIF **Graphics Interchange Format.**

GIRO	**Grupo de Investigación en Reutilización y Orientación a Objetos.**
GNAT	**GNU and New York University Ada Translator.**
GoF	**Gang of Four.**
GOOD	**General Object-Oriented Design.**
GOTS	**Government Off-The-Shelf.**
GQM	**Goal-Question-Metric.**
GRAPE	**Graphical Programming for Eiffel.**
GSM	**Global System for Mobile Communications.**
GTE	**Gestión de la Transición de Estado.**
GUI	**Graphics User Interface.**
HAL	**Hardware Abstraction Layer.**
HDTV	**High Definition TeleVision.**
HID	**Human Interface Device.**
HIPO	**Hierarchical Input Process Output.**
HLL	**High Level Languages.**
HOOD	**Hierarchical Object-Oriented Design.**
HP	**Hewlett-Packard.**
HPCC	**High Performance Computation & Communication**
HTML	**HyperText Markup Language.**
HTTP	**HyperText Transfer Protocol.**
HVE	**Historia de la Vida de la Entidad.**
I/O	**Input/Output.**
IA	**Inteligencia Artificial.**
IAA	**Insurance Application Architecture.**
IAB	**Internet Architecture Board.**
IAC	**Information Analysis Center.**
IANA	**Internet Assigned Numbers Administration.**
IC	**Ingeniería del Conocimiento.**
ICASE	**Integrated CASE.**
ICI	**Instituto de Cooperación Iberoamericana.**
ICSR	**International Conference on Software Reuse.**
ICT	**Institut Catalá de Tecnología.**
IDD	**Interface Design Document.**
IDE	**Integrated Development Environment.**
IDeA	**Intelligent Design Aid.**
IDEAL	**Interactive Development Environment for Active Learning.**
IDL	**Interface Definition Language.**
i-DL	**Internal Data Language.**
IDO	**Identificador De Objeto.**
IE	**Information Engineering.**
IE	**Internet Explorer.**
IEC	**International Electrotechnical Commission.**
IEE	**Informáticos Europeos Expertos.**
IEEE	**Institute of Electrical and Electronics Engineers**
IFIP	**International Federation for Information Processing.**
IFPUG	**International Function Point User Group.**
IIS	**Microsoft Internet Information Server.**
IJG	**Independent JPEG Group.**
IMA	**Interactive Multimedia Association.**
IMHO	**In My Humble Opinion.**
IMLS	**Intelligent Multimedia Learning System.**
IMS	**Information Management System.**
IOOM	**Ithaca Object-Oriented Methodology.**
IP	**Internet Protocol.**
IPSE	**Integrated Programming Support Environment.**
IRC	**Internet Relay Chat.**
IRD	**Information Resource Dictionary.**
IRDS	**Information Resource Dictionary System.**
IRP	**Information Resource Planning.**
IRS	**Interface Requirements Specification.**
IS	**Information System.**
IS	**Ingeniería del Software.**
ISAC	**Ingeniería del Software Asistida por Ordenador.**
ISACA	**Information System Audit and Control Association.**
ISC	**Information Systems Contract.**
ISDN	**Integrated Services Data Network.**
ISE	**Interactive Software Engineering.**
ISM	**Integrated System Management.**
ISO	**International Standards Organization.**
ISP	**Interface Segregation Principle.**
ISS	**Integral Software Systems.**
ISV	**Independent Software Vendors.**
ITHACA	**Integrated Toolkit for Highly Advanced Computers Applications.**
ITIG	**Ingeniería Técnica en Informática de Gestión.**
ITIS	**Ingeniería Técnica en Informática de Sistemas.**
ITMRA	**Information Technology Management Reform Act.**
IUCE	**Instituto Universitario de Ciencias de la Educación.**
JAD	**Joint Application Design.**
JAL	**Java Algorithm Library.**
JCL	**Job Control Language.**
JDBC	**Java DataBase Connectivity.**
JDK	**Java Development Kit.**
JECF	**Java Electronic Commerce Framework.**
JGSE	**Joint Group on System Engineering.**
JIT	**Just In Time.**
JOOP	**Journal of Object-Oriented Programming.**
JPEG	**Joint Photographic Experts Group.**
JRP	**Joint Requirement Planning.**
JSD	**Jackson Structured Design.**

JSD **Jackson System Development.**
JSP **Jackson Structured Programming.**
JTC **Joint Technical Committee.**
JVM **Java Virtual Machine.**
KBAS **Knowledge Based Software Assistant.**
KLCD **Miles de Líneas de Código.**
KPA **Key Process Areas.**
KWIC **KeyWord In Context.**
KWOC **KeyWord Out Context.**
L4G **Lenguaje de Cuarta Generación.**
LAN **Local Area Network.**
LaSSIE **Large Software System Information Environment.**
LC **Lógica de Control.**
LCD **Líneas de Código.**
LCM **Language for Conceptual Modeling.**
LCOM **Lack of Cohesion of Methods.**
LCP **Leyes de Construcción de Programas.**
LDD **Lenguaje de Definición de Datos.**
LDP **Lenguaje de Diseño de Programas.**
LEP **Lenguaje de Especificación Procedimental.**
LET **Latest Even Time.**
LFN **Long File Name.**
LIFO **Last In First Out.**
LIL **Library Interconnection Language.**
LLNL **Lawrence Livermore National Laboratory.**
LOB **Line Of Business.**
LOC **Lines Of Code.**
LOO **Lenguajes Orientados a Objetos.**
LORTAD **Ley Orgánica de Regulación del Tratamiento Automatizado de los Datos de carácter personal.**
LOU **Ley Orgánica de Universidades.**
LPI **Ley de Protección Intelectual.**
LPL **Ley de Protección Laboral.**
LPOO **Lenguajes de Programación Orientados a Objetos.**
LRU **Ley de Reforma Universitaria**
LS **Logical System Specification.**
LSA **Ley de Asociaciones Anónimas.**
LSB **Least Significant Bit.**
LSC **Large Scale Components.**
LSP **Liskov Substitution Principle.**
LTR **Lógica de Tiempo Real.**
MAN **Metropolitan Area Network.**
MAP **Ministerio para las Administraciones Públicas.**
MAPI **Messaging Application Program Interface.**
MBR **Master Boot Record.**
MCD **Modelo Conceptual de Datos.**
MCM **Method for Conceptual Modeling.**
MCTA **Modelo Conceptual de Tratamientos Analítico.**
MCVO **Modelo de Ciclo de Vida de los Objetos.**
MDA **Milestone Decision Authority.**
MENHIR **Modelos, Entornos y Nuevas Herramientas para la Ingeniería de Requisitos.**
ME/R **Modelo Entidad/Interrelación.**
MFC **Microsoft Foundation Class library.**
MIB **Management Information Base.**
MIDAS **Multi-Tier Distributed Application Server.**
MIME **Multipurpose Internet Mail Extensions.**
MIPS **Millones de Instrucciones Por Segundo.**
MIS **Management Information System.**
MIT **Massachusetts Institute of Technology.**
MLD **Modelo Lógico de Datos.**
MLDR **Modelo Lógico de Datos Repartido.**
MLT **Modelo Lógico de Tratamientos.**
MLTR **Modelo Lógico de Tratamientos Repartido.**
MOD **Modelo Organizativo de Datos.**
MOI **Módulos de Ocultamiento de Información.**
MOM **Message Oriented Middleware.**
MOOD **Material's Object-Oriented Database.**
MOON **Méthode Orientée Objects Normalisés.**
MORE **Multimedia Oriented Repository Environment**.
MOSES **Methodology for Object-oriented Software Engineering of Systems.**
MOTA **Modelo Organizativo de Tratamientos Analítico.**
MPEG **Moving Pictures Experts Group.**
MRG **Modelo de Reutilización GIRO.**
MRP **Manufacturing Resource Planning.**
MS **Microsoft.**
MSB **Most Significative Bit.**
MST **Módulos de Sincronización de Tareas.**
MTA **Mail Transfer Agent.**
MVC **Model View Controller.**
MVE **Modular Visualisation Environment.**
MVS **Multiple Virtual Storage.**
NAG **Numerical Algorithms Group.**
NAS **Network Applications Support.**
NASA **National Aeronautics and Space Administration.**
NBS **National Bureau of Standards.**
NC **Network Computer.**
NCC **Norwegian Computer Center.**
NCOSE **National Council On Systems Engineering.**
NCS **Network Computing System.**
NCSA **National Center for Supercomputing Applications.**
NCSS **Non-Commented Source Statements.**
NDL **National Database Language.**

NFS **Network File System.**
NGPM **Next Generation Process Model.**
NHSE **National HPCC Software Exchange.**
NIAM **Nijssen Information Analysis Methodology.**
NICE **Non Profit International Consortium for Eiffel.**
NIH **National Institute of Health.**
NIST **National Institute of Standards and Technology.**
NLH/E **Natural Language Help/English.**
NNTP **Network News Transfer Protocol.**
NOC **Number of Children.**
NOS **Network Operating System.**
NSA **National Security Agency.**
NSDIR **National Software Data and Information Repository.**
NT **New Technology.**
NTDFS **NT Distributed Filing System.**
NTOFS **NT Object Filing System.**
OAG **Open Application Group.**
OAI **Organización de Auditoria Informática.**
OASIS **Open and Active Specification of Information Systems**
OBA **Object Behavior Analysis.**
OCL **Object Constraint Language.**
OCP **Open-Closed Principle.**
OCR **Optical Character Readers.**
OCU **Object Connection Update.**
OCX **Object Linking and Embedding Custom Controls.**
ODA **Open Document Architecture.**
ODBC **Open Data Base Connectivity.**
ODM **Object Database Model.**
ODM **Organization Domain Modeling.**
ODMG **Object Database Management Group.**
ODMS **Object-oriented Database Management Systems.**
ODS **Object Description Skeleton.**
ODT **Object Definition Tool.**
OED **Oxford English Dictionary.**
OEM **Original Equipment Manufacturer.**
OLAP **On-Line Analytical Processing.**
OLE **Object Linking and Embedding.**
OLTP **On-Line Transaction Processing.**
OM **OASIS Method.**
OMA **Object Management Architecture.**
OMA **Object Mentor Associates.**
OMG **Object Management Group.**
OML **OPEN's Metamodel and Notation.**
OMT **Object Modeling Technique.**
ONC **Open Network Computing.**
OO **Object-Oriented.**
OO **Orientación a Objetos.**
OOA **Object-Oriented Analysis.**
OOAD **Object-Oriented Analysis and Design.**
OOAP **Object-Oriented Assistant Prototyper.**
OOCL **Object-Oriented Change and Learning.**
OOD **Object-Oriented Design.**
OODB **Object-Oriented Data Base.**
OODBMS **Object-Oriented Data Base Management System.**
OODCE **Object-Oriented Distributed Computing Environment.**
OODLE **Object-Oriented Design Language.**
OOIS **Object-Oriented Information Systems.**
OOL **Object-Oriented Language.**
OOP **Object-Oriented Programming.**
OOPSLA **Object-Oriented Programming Systems, Languages and Applications.**
OOSC **Object-Oriented Software Construction.**
OOSD **Object Oriented Structured Design.**
OOSE **Object Oriented Software Engineering.**
OOUI **Object Oriented User Interface.**
OOZE **Object Oriented Z Environment.**
OPEN **Object-oriented Process, Environment and Notation.**
OQL **Object Query Language.**
ORB **Object Request Broker.**
ORM **Object with Roles Model.**
OSA **Object-oriented Systems Analysis.**
OSD **Office of the Secretary of Defense.**
OSF **Open Software Foundation.**
OSI **Open System Interconnection.**
OT **Object Technology.**
OTAN **Organización del Tratado del Atlántico Norte.**
OTC **Object Technology Centers.**
OTUG **Object Technology Users Group.**
OUT **Object Under Test.**
OWL **Object Windows Library.**
PAL **Public Ada Library.**
PC **Personal Computer.**
PCT **Private Communication Technology.**
PCTE **Portable Common Tool Environment.**
PD **Patrón de Diseño.**
PD **Physical Design.**
PDF **Portable Document Format.**
PDL **Programs Design Language**.
PDR **Preliminary Design Review.**
PERT **Program Evaluation and Review Technique.**
PF **Puntos de Función.**
PFA **Puntos de Función Ajustados.**

PFNA **Puntos de Función No Ajustados.**
PGGC **Plan General de Garantía de Calidad.**
PGP **Pretty Good Privacy.**
PGSR **Plan de Gestión y de Supervisión del Riesgo.**
PHS **Prosperity Heights Software.**
PIA **Modelo de Pruebas, Implantación y Aceptación del Sistema.**
PIAi **Actividad i del Modelo de Pruebas, Implantación y Aceptación del Sistema.**
PIE **Process Improvement Experiment.**
PIER **Process Improvement Experiment in Reuse.**
PII **Process Improvement Institute.**
PIN **Plan Informático Nacional.**
PIP **Process Improvement Paradigm.**
PIRS **Preliminary Interface Requirements Specification.**
PJ/NF **Projection-Join Normal Form.**
PLoP **Pattern Languages of Program Design.**
PLoPD **Pattern Languages of Program Design.**
PM **Program Manager.**
PMM **Process Maturity Model.**
PNG **Portable Network Graphics.**
POO **Programación Orientada a Objetos.**
POSS **Persistent Object Service Specification.**
PP **Proceso Primitivo.**
PPP **Point to Point Protocol.**
PROLOG **Programming in LOGic.**
PSL **Problems Specification Language.**
PSL/PSA **Problem Statement Language/Problem Statement Analyzer.**
PSRS **Preliminary Software Requirements Specification.**
PVVS **Plan de Verificación y Validación del Software.**
PYME **Pequeñas Y Medianas Empresas.**
QBE **Query By Example.**
QFD **Quality Function Deployment.**
QoS **Quality Of Service.**
QUEL **QUEry Language.**
QWAN **the Quality Without a Name.**
RA **Requirements Analysis.**
RAAT **Reuse Acquisition Action Team.**
RAD **Rapid Application Development.**
RAM **Random Access Memory.**
RAPPeL **Requirements-Analysis-Process Pattern Language for object-oriented development.**
RBSE **Repository Based Software Engineering.**
RD **Requirements Design.**
RDC **Revisión del Diseño Crítico.**
RDD **Responsibility Driven Design.**
RdP **Red de Petri.**
RDP **Revisión del Diseño Preliminar.**
RDSI **Red Digital de Servicios Integrados.**
REBOOT **Reuse Based on Object-Oriented Techniques.**
RECAST **Requirements Composition And Specification Tool.**
RENOIR **European Requirements Engineering Network of Excellence.**
REP **Reuse/Release Equivalence Principle.**
RFC **Request For Comments.**
RFC **Response For a Class.**
RFP **Request For Proposals.**
RG **Revisión de Gestión.**
RIAT **Reuse Issues Action Team.**
RIB **Repository In a Box.**
RICIS **Research Institute for Computing and Information System.**
RID **Record ID.**
RIG **Reuse library Interoperability Group.**
RISC **Reduced Instruction Set Computer.**
RLF **Reuse Library Framework.**
RM **Reference Model.**
RMFF **Reuse Methodology Fusion Framework.**
RMI **Remote Method Invocation.**
ROAD **Report on Object Analysis & Design.**
ROADS **Reuse Oriented Approach for Domain based Software**.
ROI **Return Of Investment.**
ROM **Read Only Memory.**
ROOM **Real-Time Object-Oriented Modeling.**
ROSE **Reusable Object Software Engineering.**
ROSE **Reuse Of Software Elements.**
RP **Red de Petri.**
RPC **Remote Procedure Call.**
RPVVS **Revisión del Plan de Verificación y Validación del Software.**
RR.HH **Recursos Humanos.**
RRM **Reglamento Registro Mercantil.**
RRS **Revisión de Requisitos del Software.**
RS **Requirements Specification.**
RSC **Reuse Steering Committee.**
RSL **Reusable Software Library.**
RSOC **Rockwell Space Operations Company.**
RSRG **Reusable Software Research Group.**
RSVP **Rapid System Virtual Prototyping.**
RTB **Red Telefónica Básica.**
RTEE **Real Time Engineering Environment.**
RTF **Rich Text File.**
RTSA **Real-Time Structured Analysis.**
RTTI **Run Time Type Identification.**
RUP **Rational Unified Process.**
SA **Structured Analysis.**

SA/SD **Structured Analysis/Structured Design.**
SA-CMM **Software Acquisition Capability Maturity ModelSM.**
SADT **Structured Analysis and Design Technique.**
SAF **Specific Application Frame.**
SAIC **Science Applications International Corporation.**
SAP **Stable Abstractions Principle.**
SATC **Software Assurance Technology Center.**
SC **Subcomité.**
SCE **Software Capability Evaluation.**
SCE **Structure Chart Editor.**
SCSI **Small Computer System Interface.**
SDCCR **Software Development Capability/Capacity Review.**
SDCE **Software Development Capability Evaluation.**
SDD **Software Design Document.**
SDK **Software Development Kit.**
SDL **Software Development Library.**
SDLC **Systems Development Life Cycle.**
SDM **Semantic Data Model.**
SDP **Software Development Plan.**
SDP **Stable Dependencies Principle.**
SDRI **Sistema de Diccionario de Recursos de Información.**
SE **Sistemas Expertos.**
SE **Software Engineering.**
SEDDR **Software Engineering Data Definition and Representation.**
SEE **Software Engineering Environment.**
SEI **Software Engineering Institute.**
SEIS **Sociedad Española de Informática Sanitaria.**
SEN **Software Engineering Notes.**
SER **Software Evolution and Reuse**.
SERR **Software Engineering Risk Repository.**
SESC **Software Engineering Standards Committee.**
SET **Secure Electronic Transactions.**
SFA **Software Frameworks Association.**
SGBD **Sistema Gestor de Bases de Datos.**
SGBDR **Sistema Gestor de Bases de Datos Relacionales.**
SGBDRO **Sistema Gestor de Bases de Datos Objeto-Relacional.**
SGDRI **Sistema de Gestión de Diccionarios de Recursos de Información.**
SGI **Silicon Graphics Incorporated.**
SGL **Standard General Ledger.**
SGML **Standard Generalized Markup Language.**
SI **Sistema de Información.**
SIA **Sistema de Información Automatizado.**
SIB **Software Information Base.**
SICS **Swedish Institute for Computer Science.**
SIG **Sistemas de Información Geográfica.**
SIG **Special Interest Group.**
SIGMOD **Special Interest Group on Management Data.**
SLCSE **Software-Life Cycle Support Environment.**
SMS **Systems Management Server.**
SMTP **Simple Mail Transfer Protocol.**
SNAP **Semantic Net for Analysis & Prototyping.**
SNOBOL **StriNg Oriented symBOlic Language.**
SO **Sistema Operativo.**
SOA **Start Of Authority.**
SOHO **Small Office Home Office.**
SOM **IBM's System Object Model.**
SOM **Structured Object Method.**
SOMA **Semantic Object Modeling Approach.**
SOTR **Sistema Operativo en Tiempo Real.**
SPC **Software Productivity Centre.**
SPI **Software Process Improvement.**
SPICE **Software Process Improvement and Capability dEtermination.**
SPQR **Software Productivity Quality and Reliability.**
SPR **Software Productivity Research.**
SQA **Software Quality Assurance.**
SQL **Structured Query Language.**
SRBM **Software Reuse Business Model.**
SREM **Software Requirements Engineering Methodology.**
SRS **Software Requirements Specification.**
SSADM **Structured System Analysis and Design Method.**
SSDD **System/Segment Design Document.**
SSR **Software Specification Review.**
SSR **Symposium on Software Reusability.**
SSS **System Segment Specification.**
SST **Sistemas Software Tradicionales.**
SST **Strategic Systems Technology.**
STARS **Software Technology Adaptable Reliable System.**
STC **Software Technology Conference.**
STL **Standard Templates Library.**
STP **Software Technology Program.**
STP **Software Test Plan.**
STR **Software Test Report.**
STSC **Software Technology Support Center.**
SUM **Software Usage Monitor.**
SURF **Software re-Use: a process impRovement experiment at an IBM Italia Facility**.
SWSC **Air Force Space and Warning System Centre.**
TAD **Tipo Abstracto de Datos.**

TAGS **Technology for the Automated Generation of Systems.**
TAPI **Telephony Application Programming Interface.**
TBD **To Be Determined.**
TCL/TK **Tool Command Language Toolkit.**
TCM **Toolkit for Conceptual Modeling.**
TCP/IP **Transmission Control Protocol/Internet Protocol.**
TDE **Transition Diagram Editor.**
TE **Tiempo Early.**
TI **Tecnologías de la Información.**
TL **Tiempo Late.**
TO **Tecnologías de Objetos.**
TO **Triggering Operation.**
TOA **The Object Agency.**
TODS **ACM Transactions On Database Systems.**
TOG **The Open Group.**
TP **Transaction Processing.**
TPI **Tamaño de Proceso de Información.**
TPV **Terminal Punto de Venta.**
TROOPER **The Reusable OO Parser for Eiffel Re-engineering.**
UAL **Unidad Aritmético Lógica.**
UBU **Universidad de Burgos.**
UC **Unidad de Control.**
UCP **Unidad Central de Proceso.`**
UDDI **Universal Description, Discovery & Integration.**
UDL **Unified Database Language.**
UE **Unión Europea.**
UIMS **User Interface Management Systems.**
UML **Unified Modeling Language.**
UoD **Universe of Discourse.**
UPM **Universidad Politécnica de Madrid.**
UPRR **Union Pacific RailRoad.**
UPV **Universidad Politécnica de Valencia.**
URI **Uniform Resource Identifier.**
URL **Uniform Resource Locator.**
USAL **Universidad de Salamanca.**
USD **United States Dollars.**
UVA **Universidad de Valladolid.**
UWA **User Work Area.**
V&V **Verificación y Validación.**
VB **Visual Basic.**
VBA **Visual Basic for Applications.**
VBX **Visual Basic eXtensions.**
VC++ **Visual C++.**
VCL **Visual Component Library.**
VDD **Version Description Document.**
VDL **Vienna Definition Language.**
VDM **Vienna Development Method.**
VHLL **Very High Level Languages**
VIFF **Visualization Image File Format.**
VM **Virtual Machine.**
VRML **Virtual Reality Modeling Language.**
W3C **World Wide Web Consortium.**
WAIS **Wide Area Information Servers.**
WAN **Wide Area Network.**
WBS **Working Breakdown Structure.**
WDM **Win32 Drivel Model.**
WEL **Windows Eiffel Library.**
WG **Working Group.**
WIDE **Workflow on Intelligent Distributed database Environment.**
WIMP **Windows, Icons, Menus and Pointing.**
WISR **Workshop on Institutionalizing Software Reuse.**
WISE **World Wide Information System.**
WMC **Weighted Methods per Class.**
WMRA **Write Many, Read Always.**
WORM **Write Once, Read Many Times.**
WSLD **Web Services Description Language.**
WSRD **Worldwide Software Resource Discovery.**
WWW **World Wide Web.**
WYSIWYG **What You See Is What You Get.**
XIE **X Image Extension.**
XML **eXtensible Markup Language.**
XPARC **Xerox Palo Alto Research Park.**
Y2K **Year 2000.**
YACC **Yet Another Compiler Compiler.**
YACL **Yet Another Class Library.**
YOOCC **Yes! An OO Compiler Compiler.**
YP **Yellow Pages.**
YSM **Yourdon Structured Method.**

Apéndice E:
Glosario de Términos

A

Abstracción: Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes [Graham, 1994]. Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador [Booch, 1994].

Acoplamiento: Es el grado de interdependencia entre los módulos [Piattini et al., 1996].

Agregación: Forma especial de asociación que especifica una relación todo/parte entre el agregado (todo) y un componente (parte) [OMG, 1999].

Alfabeto: Se llama alfabeto a un conjunto finito y no vacío. Los elementos de un alfabeto se denominan letras o símbolos. Un alfabeto se representa por Σ [Alfonseca et al., 1990].

Algoritmo: Conjunto finito de reglas que dan una secuencia de operaciones para resolver un tipo específico de problema, o lo que es lo mismo, la traducción de aquél en una serie de instrucciones simples, en secuencia, encaminadas a realizar una tarea concreta.

Algoritmo Genético: Modelos computacionales basados en los mecanismos naturales de evolución genética de los organismos biológicos, que se aplican a la resolución de una amplia gama de problemas [García y Maudes, 1996].

Análisis: Distinción y separación de las partes de un todo hasta llegar a conocer sus principios o elementos [DRAE, 1995].

Análisis de Requisitos: Proceso de estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos [IEEE, 1999].

Análisis del Dominio: Actividad de identificar objetos y operaciones de un tipo de sistemas similares en un dominio del problema particular.

Análisis Orientado al Objeto: (AOO) Proceso que modela el dominio del problema identificando y especificando un conjunto de objetos semánticos que interaccionan y se comportan de acuerdo a los requisitos del sistema.

Árboles de Decisión: Modelo de una función discreta en la que se determina el valor de una variable y en función de su valor se lleva a cabo una acción.

Arquitectura del Software: Es la estructura global del software y las maneras en que esa estructura proporciona integridad conceptual a un sistema [Shaw y Garlan, 1995]. La estructura lógica y física de un sistema, forjada por todas las decisiones de diseño estratégicas y tácticas aplicadas durante el desarrollo [Booch, 1994]. La arquitectura del software se refiere a dos características importantes del software de computadora: la estructura jerárquica de los componentes procedimentales (*módulos*) y a la estructura de los datos [Pressman, 1992].

Arquitectura de Cuatro Capas: Es el framework conceptual de metamodelado generalmente aceptado. Explica las relaciones entre el meta metamodelo, el metamodelo, el modelo y el nivel de datos de usuario. Juntos forman las cuatro capas una encima de la otra [Metamodel, 1997].

Asociación: Una asociación describe un grupo de enlaces con una estructura y una semántica en común [Rumbaugh et al., 1991].Una relación que describe un conjunto de vínculos [OMG, 1999]. Una asociación representa una dependencia semántica entre clase e implica la dirección de esta dependencia [Joyanes, 1998].

Asset: Cualquier producto del ciclo de vida del software que pueda ser potencialmente reutilizado. Esto incluye: modelo de dominio, arquitectura de dominio, requisitos, diseño, código, bases de datos, esquemas de bases de datos, documentación, manuales de usuario, casos de prueba... [DoD, 1992], [NIST, 1994].

Atributo: Es un valor de un dato que está almacenado en los objetos de una clase [Rumbaugh et al., 1991]. Un atributo de una clase describe una información singular almacenada en cada instancia [Booch, 1994].

B

Bean: Es un componente software reutilizable que puede ser manipulado visualmente en una herramienta de desarrollo Java [Hamilton, 1997].

C

Calidad: Totalidad de características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas [AENOR, 1992].

Calidad del Software: Conjunto global de características de un producto software, relacionado con su capacidad de satisfacer unas necesidades dadas; Grado en el que un software posee una combinación de atributos deseada; Grado en que el usuario percibe que el software satisface sus expectativas [AECC, 1986]. Grado con el que un sistema, componente o proceso cumple: *los requisitos especificados* y *las necesidades o expectativas del cliente o usuario* [IEEE, 1999].

Camino Crítico: Secuencia más larga de actividades conectadas a través de la red y que determina la duración total del proyecto [Piattini et al., 1996].

Campo Clave: Aquellos campos que identifican unívoca y mínimamente los registros.

Característica: Palabra genérica tanto para un atributo como para una operación [Rumbaugh et al., 1991].

CASE: Conjunto de herramientas y metodologías que soportan un enfoque de ingeniería en el desarrollo del software en alguna o en todas las fases de este proceso [Piattini et al., 1996].

Caso de Uso: Forma o patrón o ejemplo concreto de utilización, un escenario que comienza con algún usuario del sistema que inicia alguna transacción o secuencia de eventos interrelacionados [Jacobson et al., 1993].

Un caso de uso es una unidad coherente de funcionalidad que proporciona un clasificador (un sistema, un subsistema o clase) tal como lo manifiestan las secuencias de mensajes que se intercambian entre el sistema y uno o más usuarios externos (que se representan como actores), junto con acciones que realiza el sistema [Rumbaugh et al., 1999].

Centro de Transformación: Es la parte del DFD que contiene las funciones esenciales del sistema, independientemente de la implementación particular de la entrada y de la salida [Piattini et al., 1996].

Cibernética: Ciencia que estudia comparativamente los sistemas de comunicación y regulación automática de los seres vivos con sistemas electrónicos y mecánicos semejantes a aquellos. Entre sus aplicaciones está el arte de construir y manejar aparatos y máquinas que mediante procedimientos electrónicos efectúan automáticamente cálculos complicados y otras operaciones similares [DRAE, 1995].

Ciclo de Desarrollo del Software: Se denomina ciclo de desarrollo del software al período de tiempo que comienza con la decisión de desarrollar un producto software y finaliza cuando se ha entregado éste. Este ciclo incluye, en general, una fase de requisitos, una fase de diseño, una fase de implantación, una fase de pruebas, y a veces, una fase de instalación y aceptación [AECC, 1986].

Ciclo de Vida del Software: Es un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso [ISO/IEC, 1995].

Clase: Es un conjunto de objetos que comparten una estructura común y un comportamiento común [Booch, 1994]. Es la implementación de un tipo de objeto (*considerando los objetos como instancias de las clases*) [Piattini et al., 1996].

Clase Abstracta: Clase que no representa completamente un objeto. En su lugar, representa un amplio rango de diferentes clases de objetos. Sin embargo, esta representación mantiene sólo las características que dichas clases de objetos tienen en común. Por lo tanto, una clase abstracta sólo ofrece una descripción parcial de sus objetos [Martin, 1992]. Una clase que no tiene instancias. Una clase abstracta se escribe con la intención de que sus subclases concretas añadan elementos nuevos a su estructura y comportamiento, normalmente implementando sus operaciones abstractas [Booch, 1994]. Clase que no puede tener instancias directas pero cuyos descendientes sí pueden tenerlas [Rumbaugh et al., 1991].

Clase Concreta: Clase que puede tener instancias directas [Rumbaugh et al., 1991].

Clase Contenedora: Clase de objetos contenedores. Entre los ejemplos se incluyen los conjuntos, las matrices, los diccionarios y las asociaciones [Rumbaugh, et al., 1991].

Clase Genérica: Véase clase parametrizada.

Clase Parametrizada: Plantilla para la creación de clases reales que pueden diferir en formas bien definidas, como indican sus parámetros en el momento de creación. A menudo, los parámetros son tipos de datos o clases, pero pueden incluir otros atributos, como el tamaño de una colección. También denominada clase genérica [Rumbaugh et al, 1991].

Cohesión: Medida de la relación funcional de los elementos de un módulo [Piattini et al., 1996].

Complejidad Ciclomática: Métrica del software que proporciona una medición cuantitativa de la complejidad lógica de una programa [Pressman, 1997].

Componente: Un componente es una unidad de composición con unas interfaces contractualmente especificadas con dependencias exclusivamente del contexto. Los componentes pueden ser desarrollados de forma independiente y son integrados por terceras partes [Szyperski y Pfister, 1996].

Componentware: Estrategia de desarrollo que pretende, en lugar de construir grandes aplicaciones de forma completa, dedicarse a la creación de unidades arquitectónicas, denominadas componentes, con el objetivo de integrarlas para, que colaborando entre sí, obtener una funcionalidad mucho mayor que la que cada uno de ellos puede ofrecer por separado.

Comportamiento de un objeto: El comportamiento de un objeto es cómo actúa y reacciona un objeto, en función de sus cambios de estado y paso de mensajes [Booch, 1994].

Composición: Relación semántica que describe una forma de agregación con un matiz fuerte de propiedad y de coincidencia de vida como partes de un todo. Las partes con una multiplicidad que no es fija pueden ser creadas después del objeto compuesto, pero una vez creadas ellas viven y mueren con él. Estas partes pueden ser también eliminadas antes de la muerte del objeto compuesto. La composición puede ser recursiva [OMG, 1999].

Concurrencia: (En OO) Es la propiedad que distingue un objeto activo de uno que no está activo [Booch, 1994].

Control de Proyecto: Consiste en la supervisión periódica y en la comparación de los resultados con los previstos en el calendario [Piattini et al., 1996].

Covarianza: Propiedad por la que cuando en un proceso de especialización, se produce una redefinición de un atributo en la subclase, el atributo de la superclase y el atributo redefinido deben variar juntos [Meyer, 1997].

D

Dato: Están constituidos por registros de los hechos, acontecimientos, transacciones... [Lucey, 1991].

Deliverable: Véase producto a entregar.

Descomposición del Trabajo: Técnica que permite representar las actividades que hay que realizar a distinto nivel de detalle por medio de un diagrama de estructuras. Se corresponde con el término *Working Breakdown Structure* (WBS) [Piattini et al., 1996].

Diagrama de Clases: Es el diagrama que muestra una colección de elementos declarativos (estáticos) del modelo, como clases y tipos, sus contenidos y relaciones [OMG, 1999].

Diagrama de Colaboraciones: Es un diagrama que muestra interacciones entre objetos organizados alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero de una forma diferente [OMG,1999].

Diagrama de Contexto: El diagrama de contexto, o diagrama de nivel 0, es el primero de la jerarquía de DFD. El objetivo de este diagrama es delimitar la frontera entre el sistema con el mundo exterior y definir sus interfaces, esto es, los flujos de entrada y salida del sistema con el entorno. El diagrama de contexto está formado por un proceso que se representa como una caja negra del sistema completo, un conjunto de entidades externas que presentan la procedencia y el destino de la información del sistema, y un conjunto de flujos de datos que representan los caminos por los que fluye dicha información [Piattini et al., 1996].

Diagrama de Estructuras: Técnica que permite definir cuándo, bajo qué condiciones y cuántas veces se tienen que realizar los tratamientos identificados en los procesos de un DFD [MAP, 1995].

Diagrama de Estructura de Cuadros de Constantine: Nombre que recibe en Métrica 2.1 el *diagrama de estructuras*. Véase Diagramas de Estructuras. [MAP, 1995].

Diagrama de Estructura de Datos: Modelo de información compuesto exclusivamente por relaciones **1:N** [Piattini et al., 1996].

Diagrama de Flujo de Datos: DFD. Es un diagrama en forma de red que representa el flujo de datos y las transformaciones que se aplican sobre ellos al moverse desde la entrada hasta la salida del sistema [Piattini et al., 1996].

Diagrama de Instancias: Describe la forma en que un cierto conjunto de objetos se relacionan entre sí. Describe instancias de objetos [Rumbaugh et al., 1991]. Este concepto se corresponde con el concepto de diagrama de objetos de UML [OMG, 1999].

Diagrama de Objetos: Los diagramas de objetos proporcionan una notación gráfica formal para el modelado de objetos, clases y sus relaciones entre sí, son útiles, tanto para el modelado abstracto como, para diseñar programas reales [Rumbaugh et al., 1991]. Diagrama que contiene objetos y sus relaciones en un momento dado del tiempo. Un diagrama de objetos puede ser considerado un caso especial de un diagrama de clases o de un diagrama de colaboraciones [OMG, 1999]. Esta definición se ajusta al concepto de diagrama de instancias de Rumbaugh en OMT [Rumbaugh et al., 1991].

Diagrama de Secuencias: Un diagrama que muestra interacciones entre objetos organizadas en secuencia temporal. En particular muestra los objetos participantes de la interacción y la secuencia de mensajes intercambiados. A diferencia del diagrama de colaboraciones, un diagrama de secuencias incluye la secuencia temporal pero no incluye las relaciones entre los objetos. Pueden existir diagramas de secuencias genéricos (que describen todos los escenarios posibles) y de instancias (que describen un escenario particular). Los diagramas de secuencias y de colaboraciones expresan información similar pero de una manera diferente [OMG, 1999].

Diagrama de Transición de Estados: DTE. Diagrama que representa los estados que puede tomar un componente o un sistema y que, además, muestra los eventos o circunstancias que implican el cambio de un estado a otro.

Diagrama Estático de Estructuras: Los diagramas estáticos de estructura muestran la estructura estática del modelo; en concreto muestran las cosas que existen como son las clases y tipos, su estructura interna, y su relación con el resto de los elementos [OMG, 1999]. Este concepto se corresponde con el concepto de diagrama de objetos enunciado por Rumbaugh para OMT [Rumbaugh et al., 1991].

Diccionario de Datos: DD. Lista organizada de todos los datos utilizados por el sistema, con definiciones precisas y rigurosas, para que cliente y analista tengan una visión común de todos los flujos y almacenes.

Diccionario de Recursos de Información: DRI o en inglés – ***IRD*** *Information Resource Dictionary*. Depósito integrado de todos los datos sobre la organización, automatizados o no, que son utilizados para efectuar las labores de planificación, control y operación que permitan a la empresa cumplir sus objetivos [Piattini et al., 1996].

Diseño Software: Es el proceso de definición de la arquitectura software: componentes módulos, interfaces, procedimientos de prueba y datos de un sistema que se crean para satisfacer unos requisitos especificados.

Diseño Arquitectónico: Define las relaciones entre los principales elementos estructurales del programa [Pressman, 1997].

Diseño de Datos: El diseño de datos transforma el modelo de dominio de la información, creado durante el análisis, en las estructuras de datos necesarias para la implementación del software [Pressman, 1997].

Diseño de la Interfaz: El diseño de la interfaz describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean [Pressman, 1997].

Diseño Estructurado: Es el arte de diseñar los componentes de un sistema y la relación entre ellos de la mejor forma posible [Yourdon y Constantine, 1979]. Es el proceso de decidir la forma en la cual componentes interconectados resolverán un problema bien especificado [Yourdon y Constantine, 1979].

Diseño Orientado al Objeto: (DOO) Proceso que modela el dominio de la solución, lo que incluye a las clases semánticas con posibles añadidos, y las clases de interfaz, aplicación y utilidad identificadas durante el diseño.

Diseño Procedimental: El diseño procedimental transforma elementos estructurales de la arquitectura del programa en una descripción procedimental de los componentes del software [Pressman, 1997].

Director de Proyecto: Persona que tiene la responsabilidad de planificar, controlar y dirigir las actividades del proyecto. Muchas veces estas responsabilidades suponen la coordinación e integración de actividades a través de las unidades organizativas [Piattini et al., 1996].

Documento Base: Producto que ha sido formalmente revisado y aceptado, y que sólo puede cambiar mediante un procedimiento formal de control de cambios. Esto se realiza por medio de una revisión formal del grupo de control de cambios. El término equivalente en la bibliografía en inglés es *baseline* [Piattini et al., 1996].

E

Encapsulamiento: Es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar la interfaz contractual de una abstracción y su implantación [Booch, 1994]. Es un principio de estado que agrupa datos y

procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto [Piattini et al., 1996].

Enfoque Sistemático: Es la manera de estudiar o analizar sistemas adoptando una visión global de los mismos, que se va refinando progresivamente mediante una descomposición de arriba abajo [Piattini et al., 1996].

Enlace: Conexión física o conceptual entre instancias de objetos [Rumbaugh et al., 1991].

Entidad: Objeto real o abstracto acerca del cual se quiere almacenar información en una base de datos.

Envoltorio: Véase *wrapper*.

Ergonomía: Estudio de datos biológicos y tecnológicos aplicados a problemas de mutua adaptación entre el hombre y la máquina.

Especificación: Es un documento que define, de forma completa, precisa y verificable, los requisitos, el diseño, el comportamiento u otras características de un sistema o componente de un sistema [IEEE, 1999].

Especificación Formal: Una especificación formal sería la que se describe mediante sintaxis y semánticas formales con las que se especifica el funcionamiento y el comportamiento del sistema. Una especificación formal a menudo tiene una forma matemática (*por ejemplo, se puede utilizar el cálculo de predicados como base de un lenguaje de especificación formal*) [Pressman, 1992].

Estado: Conjunto de circunstancias o atributos que caracterizan a una persona o cosa en un tiempo dado.

Estado de un objeto: El estado de un objeto abarca todas las propiedades (*normalmente estáticas*) del mismo más los valores actuales (*normalmente dinámicos*) de cada una de esas propiedades [Booch, 1994].

Estructura de Datos: Es una representación de la relación lógica existente entre los elementos individuales de datos [Pressman, 1997].

Estructura del Programa: Ver jerarquía de control.

Excepción: Una excepción es una señal especial que puede ser activada por una cierta instrucción y manejada por otra, posiblemente en una parte remota del sistema [Meyer, 1997].

F

Forma de Utilización: Véase caso de uso.

Función Primitiva: Las funciones o procesos primitivos son aquellos procesos de un DFD que no se descomponen en diagramas de nivel inferior. Por cada función primitiva tiene que existir una especificación que la describa [Piattini et al., 1996].

G

Generalización: Es una relación de clasificación entre un elemento más general y un elemento más específico. El elemento más específico es completamente consistente con el elemento más general, conteniendo información adicional.

Gestión de la Configuración: Disciplina de coordinar el desarrollo del software y controlar el cambio y evolución de los productos software y de sus componentes [Schmerl, 1996].

Gestión de Proyectos: Proceso de planificar, organizar, asignar personal, dirigir y controlar la producción de un sistema [CERN, 1997].

Gestión de Riesgos: Disciplina cuyos objetivos son identificar, direccionar y eliminar los elementos de riesgo para el software antes de que consigan entorpecer el éxito del software o convertirse en una fuente de trabajo software (*trabajo ya realizado anteriormente de otras formas*) más caro [Boehm, 1989].

H

Herencia: Mecanismo por el que elementos más específicos incorporan estructura y comportamiento de elementos más generales relacionados por el comportamiento [OMG, 1999]. Relación entre clases, en la que una clase comparte la estructura o comportamiento definido en otra (*herencia simple*) u otras (*herencia múltiple*) clases. La herencia define una relación "*de tipo*" entre clases en la que una subclase hereda de una o más superclases generalizadas; una subclase suele especializar a sus superclases aumentando o redefiniendo la estructura y comportamiento existentes [Booch, 1994].

Herencia de Constantes: Tipo de herencia de propiedades en la que todas las propiedades de la superclase **A** son constantes que describen objetos compartidos [Meyer, 1997].

Herencia de Conversión a Diferida: Tipo de herencia con variación. Se aplica cuando **B** redefine algunas características concretas de **A** en características abstractas [Meyer, 1997].

Herencia de Extensión: Perteneciente a la familia de herencia de modelado. Se aplica cuando la subclase **B** introduce características no presentes en la superclase **A** y no aplicables a las diferentes instancias de **A**. La clase **A** debe ser concreta [Meyer, 1997].

Herencia de Implementación: Perteneciente a la familia de herencia software. Herencia estructural que se aplica si una subclase **B** obtiene de una superclase **A** un conjunto de características (*otras que atributos constantes y funciones*) necesarias para la implementación de la abstracción asociada con **B**. Ambas clases **A** y **B** deben ser concretas [Meyer, 1997].

Herencia de Máquina: Tipo de herencia de propiedades en la que todas las propiedades de la superclase **A** son rutinas que pueden verse como operaciones de una máquina abstracta [Meyer, 1997].

Herencia de Modelado: Refleja relaciones "es un" entre las abstracciones de un modelo [Meyer, 1997].

Herencia de Propiedades: Perteneciente a la familia de herencia software. Se aplica cuando una superclase **A** existe solamente con el propósito de ofrecer un conjunto de propiedades relacionadas lógicamente para el beneficio de sus subclases como **B**. Existen dos variantes: ***herencia de constantes*** y ***herencia de máquina*** [Meyer, 1997].

Herencia de Reificación: (o de materialización) Perteneciente a la familia de herencia de software. Se aplica si una superclase **A** representa un tipo general de estructura de datos, y la subclase **B** representa una elección de implementación parcial o completa para las estructuras de datos de ese tipo. **A** es abstracta; **B** puede ser abstracta, dejando espacio para posteriores reificaciones a través de sus propias subclases, o puede ser concreta [Meyer, 1997].

Herencia de Restricción: Perteneciente a la familia de herencia de modelado. Se aplica si las instancias de **B** son aquellas instancias de **A** que cumplen una cierta restricción, expresada si es posible como parte de la invariante de **B** y no incluida en la invariante de **A**. Cualquier característica introducida por **B** debe

ser una consecuencia lógica de añadir una restricción. **A** y **B** deben ser ambas abstractas o ambas concretas [Meyer, 1997].

Herencia de Variación: Sirve para describir una clase mediante sus diferencias con otras clases [Meyer, 1997]. La herencia con variación se aplica cuando **B** redefine algunas características de **A**; **A** y **B** son ambas abstractas o ambas concretas, y **B** no puede introducir más características excepto las necesarias para redefinir las características. Existen dos casos herencia con variación funcional y herencia con variación de tipo [Meyer, 1997].

Herencia de Variación de Tipo: Herencia con variación, donde todas las redefiniciones son redefiniciones de las signaturas de las operaciones [Meyer, 1997].

Herencia de Variación Funcional: Herencia con variación, donde las redefiniciones afectan al cuerpo de las implementaciones operaciones, en lugar de a las signaturas de las mismas [Meyer, 1997].

Herencia de Vistas: Perteneciente a la familia de herencia de modelado. Varios descendentes de una cierta clase representan conjuntos de instancias no disjuntos debido a la existencia de varias formas de clasificar las instancias del padre [Meyer, 1997].

Herencia Estructural: Perteneciente a la familia de herencia software. Se aplica si una superclase **A**, una clase abstracta, representa una propiedad estructural y una subclase **B**, que puede ser abstracta o concreta, representa un cierto tipo de objetos que poseen dicha propiedad [Meyer, 1997].

Herencia Múltiple: Tipo de herencia que permite a una clase tener más de una superclase y heredar características de sus ancestros [Rumbaugh et al., 1991].

Herencia por Subtipado: Perteneciente a la familia de herencia de modelado. La herencia por subtipado se aplica si **A** y **B** representan ciertos conjuntos **A'** y **B'** de objetos externos, de forma que **B'** es un subconjunto de **A'** y el conjunto modelado por otro cualquier subtipo de herencia de **A** es disjunto de **B'**. **A** debe ser abstracta [Meyer, 1997].

Herencia Simple: Tipo de herencia por la que una clase sólo puede tener una superclase [Rumbaugh et al., 1991].

Herencia Software: Expresa relaciones software, que no son obvias en el modelo [Meyer, 1997].

Hito: Del inglés *milestone*. Representa un suceso o evento en el tiempo del que algún miembro del proyecto se hace responsable y que se utiliza para medir el progreso [Piattini et al., 1996].

I

Identidad: Es aquella propiedad de un objeto que lo distingue de todos los demás [Booch, 1994].

Información: Dato o conjunto de datos que, en un contexto determinado, tienen un significado para alguien y transmiten un mensaje útil en un lugar determinado [Monforte, 1995].

Ingeniería de Requisitos: Todas las actividades relacionadas con: (a) identificación y documentación de las necesidades de clientes y usuarios; (b) creación de un documento que describe la conducta externa y las restricciones asociadas [de un sistema] que satisfará dichas necesidades; (c) análisis y validación del documento de requisitos para asegurar consistencia, compleción y viabilidad; (d) evolución de las necesidades [Hsia et al., 1993].

Ingeniería del Software: Es la aplicación de herramientas, métodos y procedimientos de forma eficiente en cuanto al coste para producir y mantener una solución a un problema de procesamiento real, automatizándolo parcial o totalmente mediante el software [Horan, 1995].

Ingeniería Inversa: La ingeniería inversa del software es el proceso consistente en analizar un programa en un esfuerzo por crear una representación del programa con un nivel de abstracción más elevado que el código fuente [Pressman, 1997].

Instanciación: Proceso de creación de instancias de clases [Rumbaugh et al., 1991].

Interrelación: Es aquella asociación o correspondencia existente entre entidades.

J

Jerarquía: Es una clasificación u ordenación de abstracciones [Booch, 1994].

Jerarquía de Control: Representa la organización (a menudo jerárquica) de componentes del programa (módulos) e implica una jerarquía de control [Pressman, 1997].

L

Lenguaje: Lenguaje sobre el alfabeto Σ es todo subconjunto del lenguaje universal de Σ [Alfonseca et al., 1990].

Lenguaje Estructurado: Lenguaje de especificación que hace uso de un vocabulario y una sintaxis limitados.

Lenguaje Formal: Es un lenguaje compuesto de tres componentes principales: una sintaxis, que define la notación específica con la que se representa la especificación; una semántica, que ayuda a definir un universo de objetos que se usarán para describir el sistema, y por último, un conjunto de relaciones que definen las reglas que indican qué objetos satisfacen adecuadamente la especificación [Pressman, 1992].

Lenguaje Universal: Es el conjunto de todas las palabras que se pueden formar de un alfabeto Σ. También se denomina ***universo del discurso***, y se representa por $W(\Sigma)$ o Σ^*. El lenguaje universal es un conjunto infinito [Alfonseca et al., 1990].

Ligadura: Ligadura es el proceso por el que se vincula el nombre del servicio a la implementación. En general el sistema selecciona el método más específico [Piattini, 1996].

M

Matriz Entidad/Función: Visualiza las relaciones existentes entre las funciones que lleva a cabo un sistema y la información necesaria para soportar las mismas [Piattini et al., 1996].

Mensaje: Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento [OMG, 1999].

Meta metamodelo: Un meta metamodelo define el lenguaje para expresar metamodelos [Metamodel, 1997].

Metaclase: Una clase cuyas instancias son a su vez clases [Berard, 1996].

Metadatos: Datos acerca de los datos. Los metadatos que están presentes durante la ejecución permiten a la aplicación razonar acerca de su propia estructura y capacidades, con posibilidad cambiarlas; aquí se incluyen las operaciones que admiten los objetos, los atributos que poseen o los tipos de los atributos.

Metamodelo: El modelo de información para la información que puede ser expresada durante el modelado [Metamodel, 1997]. Un modelo que define el lenguaje para expresar un modelo [OMG, 1999].

Método: Es la implementación de una operación en una clase. El algoritmo o procedimiento que permite llegar al resultado de una operación [OMG, 1999].

Metodología: Una metodología de Ingeniería del Software es un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas [Rumbaugh et al., 1991].

Milestone: Véase hito.

Modelo: Abstracción de un sistema semánticamente cerrada [OMG, 1999]. Una colección de elementos ensamblados durante el modelado de un sistema, como puede ser un sistema software [Metamodel, 1997].

Modelo de Componentes Software: Es una especificación de cómo desarrollar componentes software reutilizables, y cómo estos objetos pueden comunicarse con el resto.

Modelo de Objetos: La colección de principios que forman las bases del diseño orientado a objetos; un paradigma de Ingeniería del Software que enfatiza los principios de abstracción, encapsulamiento, modularidad, jerarquía, tipos, concurrencia y persistencia [Booch, 1994].

Modelo de Proceso Software: Formalismos que permiten la representación de los conceptos que subyacen al proceso software.

Modelo Esencial: El modelo esencial del sistema es un modelo de lo que el sistema debe hacer para satisfacer los requisitos del usuario, diciendo lo menos posible (*preferiblemente nada*) acerca de cómo se implantará [Yourdon, 1989].

Modelo Semántico: Modelo que captura el significado de las entidades y las relaciones del mundo real.

Modularidad: Es el atributo individual del software que permite a un programa ser intelectualmente manejable [Myers, 1978]. Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados [Booch, 1994].

Módulo: Parte lógica separable de un programa [AECC, 1986]. Subconjunto coherente de un subsistema que contiene un grupo de clases fuertemente acotadas y sus interrelaciones [Rumbaugh et al., 1991].

Multiplicidad: Especifica el número de instancias de una clase que pueden estar relacionadas con una única instancia de una clase asociada [Rumbaugh et al., 1991].

N

Normalización: Reducción sucesiva de un conjunto dado de relaciones a una forma más deseable.

O

Objetivo de Proyecto: Enunciado que especifica los resultados a conseguir. Las características que debe cumplir un objetivo de proyecto para que quede bien definido son: *Asequible*: Meta que se puede alcanzar dentro de un tiempo determinado y con unas restricciones dadas. *Definitivo*: Especifica qué es lo que se

tiene que conseguir de forma concreta. *Cuantificable*: Especifica un criterio de finalización. *De duración específica*: Define la duración de las actividades. [Piattini et al., 1996].

Objeto: En el ámbito conceptual un objeto es una entidad percibida en el sistema que se está desarrollando, mientras que al nivel de implementación, un objeto se corresponde con un encapsulamiento de un conjunto de operaciones (servicios) que pueden ser invocadas externamente y de un estado que recuerda el efecto de sus servicios. Un objeto se describe por sus propiedades, también llamadas atributos (*estructura del objeto*) y por los servicios que puede proporcionar (*comportamiento del objeto*). El estado de un objeto viene determinado por los valores que toman sus atributos, valores que han de cumplir siempre las restricciones impuestas sobre ellos [Piattini et al., 1996]. Algo a lo cual se le puede hacer algo. Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares se definen en su clase común. Los términos instancia y objeto son intercambiables [Booch, 1994].

Objeto Activo: Aquel objeto extraído de una abstracción del mundo real que representa un hilo separado de control (*una abstracción de un proceso*) [Booch, 1994].

Operación: Es una función o transformación que se puede aplicar o que puede ser aplicada por los objetos de una clase [Rumbaugh et al., 1991].

P

Palabra: Se denomina *palabra formada con los símbolos de un alfabeto*, a toda secuencia finita de letras de ese alfabeto [Alfonseca et al., 1990].

Paradigma: (Del latín *paradigma*, del griego *paradeigma*) El significado original era el de ejemplo ilustrativo, en particular un enunciado modelo que muestra todas las reflexiones de una palabra. Sin embargo, el historiador Thomas Kuhn en su libro *The Structure of Scientific Revolutions* (La estructura de las revoluciones científicas) extendió la definición de la palabra para abarcar un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, esto es, una forma de ver el mundo [Kuhn, 1971].

Persistencia: Es la propiedad de un objeto por la que su existencia trasciende el tiempo (*es decir, el objeto continúa existiendo después de que su creador deja de existir*) y/o el espacio (*es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado*) [Booch, 1994].

Plan de Proyecto: Un documento de gestión que describe la realización del proyecto. El plan describe, entre otros, el trabajo que ha de llevarse a cabo, los recursos necesarios y los métodos a utilizar [AECC, 1986].

Polimorfismo: La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases [Graham, 1994]. Concepto de la teoría de tipos, de acuerdo con el cual un nombre (como una declaración de una variable) puede denotar objetos de muchas clases diferentes que se relacionan mediante alguna superclase común; así todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes modos [Booch, 1994]

Política de Calidad: Directrices y objetivos generales de una Institución relativos a la calidad, expresados formalmente por la Dirección General.

Powertype: Es un clasificador estereotipado que denota que el clasificador es un metatipo, cuyas instancias son subtipo de otro tipo [OMG, 1999]. Es una dependencia estereotipada cuya fuente es un conjunto de generalizaciones y cuyo destino es un clasificador que especifica que el destino es el *powertype* de la fuente [OMG, 1999].

Proceso Ligero: Suele existir dentro de un solo proceso del sistema operativo en compañía de otros procesos ligeros, que comparten el mismo espacio de direcciones [Booch, 1994].

Proceso Pesado: Aquel típicamente manejado de forma independiente por el sistema operativo de destino, y abarca su propio espacio de direcciones [Booch, 1994].

Producto a entregar: Del inglés *deliverable*. Es un producto o servicio final que especifica el cliente o usuario y que se le entrega al final del proyecto [Piattini et al., 1996].

Proceso Software: El conjunto de actividades necesarias para transformar las ideas iniciales del usuario que desea automatizar un determinado trabajo en software.

Programación Orientada a Objetos (POO): La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia [Booch, 1994].

Protocolo: (*En orientación al objeto*) Al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las formas de invocación u órdenes que admite, se le denomina su protocolo. Un protocolo denota las formas en las que un objeto puede actuar y reaccionar, y de esta forma constituye la visión externa completa, estática y dinámica, de la abstracción [Booch, 1994].

Prototipo: Sistema software que incluye sólo algunas características del sistema final con objeto de mejorar la comprensión del diseño o de su realización [AECC, 1986].

Prototipo Robusto: Prototipo que puede evolucionar hacia un sistema final.

Proyecto: Conjunto de etapas, actividades y tareas para alcanzar un objetivo que implica un trabajo no inmediato a un plazo relativamente largo [Piattini et al., 1996].

Puntos de Función: Es una medida del tamaño de las aplicaciones de ordenador y de los proyectos donde se construyen éstas. El tamaño es medido desde un punto de vista funcional, o de usuario. Es independiente del lenguaje de programación, de la metodología de desarrollo, de la tecnología o de la capacidad del equipo del proyecto [Boehm, 1996].

R

RAD: (*Rapid Application Development*) Los entornos de desarrollo rápido de aplicaciones se basan en la creación de las aplicaciones alrededor de formularios, que no son más que ventanas inicialmente vacías. En los formularios se van colocando los controles necesarios, los cuales tienen una serie de propiedades asociadas cuyos valores son definidos a la hora de crear el programa.

Red de Petri: Una red de Petri es un grafo orientado formado por plazas o lugares, representados por círculos, transiciones, representadas por segmentos rectilíneos, y por un conjunto de arcos dirigidos que unen ambos.

Red Neuronal: Es una red de varios procesadores simples, cada uno de los cuales puede tener una pequeña cantidad de memoria local. Estas unidades están conectadas por canales de comunicación los cuales normalmente transportan datos numéricos codificados de diferentes formas. Estas unidades operan sólo con sus datos locales sobre las entradas que reciben por sus conexiones de entrada [Sarle, 1996].

Redes de Precedencia: Las redes de precedencia constituyen una representación gráfica del proyecto donde se relacionan las actividades de tal forma que se pueden visualizar las que son críticas. Además, se pueden representar en una escala de tiempos para facilitar la distribución de recursos y la determinación del presupuesto [Piattini et al., 1996].

Refinamiento: Es la descomposición funcional del sistema mediante refinamientos sucesivos a diferentes niveles de abstracción.

Regla de Negocio: Una sentencia que define o restringe algunos aspectos del negocio. Es un intento de recoger la estructura del negocio o de controlar o influenciar el comportamiento del negocio.

Reificación: Proceso de hacer que conceptos externos estén disponibles en el nivel objeto [Madany et al., 1991]. El proceso de considerar algo abstracto en una entidad material [Web, 1997].

Reingeniería: Transformación sistemática de un sistema existente en una nueva forma que realice mejoras de calidad en las operaciones, capacidades del sistema, funcionalidades de rendimiento o capacidad de evolucionar a un menor coste, calendario o riesgo para el cliente [Tilley et al., 1995].

Relación semántica: Una conexión semántica entre elementos de un modelo [OMG,1999].

Repositorio: Tipo de diccionario que contienen las herramientas CASE donde se almacenan los datos generados durante el ciclo de vida de un desarrollo: *esquemas, grafos, matrices, información relativa a la gestión de proyectos…* Existen dos tipos de repositorios. Los **r**epositorios (*con r minúscula*) que almacenan los objetos de una herramienta CASE particular. Los **R**epositorios (*con R mayúscula*), que tienen un alcance mayor, se basan en estándares e implementan un modelo de información abierto y extensible, soportando un entorno integrado de ingeniería del software [Piattini et al., 1996].

Requisito: (a) Una condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo. (b) Una condición o capacidad que debe tener un sistema o un componente de un sistema para satisfacer un contrato, una norma, una especificación u otro documento formal. (c) Una representación en forma de documento de una condición o capacidad como las expresadas en (a) o en (b) [IEEE, 1990].

Característica del sistema que es una condición para su aceptación [DoD, 1994].

Reutilización: Utilización de conceptos y objetos existentes en un sistema o situación nueva, directamente o adaptándolos. Para ello, estos conceptos y objetos deberán encontrarse codificados en un nivel de abstracción establecido y deberán poder ser recuperados [Krueger, 1992]. Cualquier procedimiento que produce o ayuda a producir un sistema mediante el nuevo uso de algún elemento procedente de un esfuerzo de desarrollo anterior [Freeman, 1987].

Riesgo: Circunstancias potencialmente adversas que pueden afectar a un proceso de desarrollo o a la calidad de los productos [Ghezzi et al., 1991].

S

Sistema: Conjunto ordenado de cosas que, ordenadamente relacionadas entre sí, contribuyen a un determinado objetivo [DRAE, 1995].

Sistema de Información: Un conjunto formal de procesos que, operando sobre una colección de datos estructurada según las necesidades de la empresa, recopilan, elaboran y distribuyen la información (*o parte de ella*) necesaria para las operaciones de dicha empresa y para las actividades de dirección y control correspondientes (*decisiones*) para desempeñar su actividad de acuerdo a su estrategia de negocio [Andreu et al., 1996].

Software: Programas, procedimientos, reglas y la posible documentación asociada y datos que pertenezcan a la explotación de un sistema de ordenador [AECC, 1986].

Software de Tiempo Real: Sistema que controla un ambiente o un entorno determinado. El sistema recoge datos de este entorno, los procesa o analiza y produce una salida con suficiente rapidez como para influir en ese entorno [Pressman, 1992].

Structure Chart: Véase Diagrama de Estructuras.

T

Tecnologías de la Información: (**TI**) Término para hacer referencia a la informática, las tecnologías de comunicación y cualquier otra técnica que permita manejar, comunicar y procesar la información en cualquiera de los formatos en que pueda presentarse [Piattini et al., 1996].

Telos: Lenguaje de representación que soporta una especificación de clases multinivel y atributos de clase, herencia e instanciación. Se utiliza como lenguaje de especificación en el **SIB** de **ITHACA**.

Término Local: Es aquél que se define explícitamente en una especificación de proceso individual [Yourdon, 1989].

Tiempo de vida de un objeto: El tiempo de vida de un objeto es el tiempo que se extiende desde el momento en que se crea un objeto por primera vez (consumiendo así espacio por primera vez) hasta que ese espacio se recupera [Booch, 1994].

Tipo Abstracto de Datos: TAD o **ADT**. Un tipo abstracto de datos es una colección de valores y de operaciones que se definen mediante una especificación que es independiente de cualquier representación [Peña, 1998].

U

Uso: Relación semántica que supone un refinamiento de una asociación, por el que se establece qué abstracción es el cliente y qué abstracción es el servidor que proporciona ciertos servicios [OMG, 1999].

V

Versión: Una versión de un objeto es una instantánea semánticamente significativa del objeto, tomada en un momento dado en el tiempo [Bertino, 1993].

Viabilidad Legal: Comprobación de que los requisitos del sistema no vayan contra alguna ley o reglamento (por ejemplo, la LORTAD) o a disposiciones legales de contratos, responsabilidad civil… [Piattini et al., 1996].

W

Working Breakdown Structure: WBS – Véase Descomposición del trabajo.

Wrapper: Clase u operación que envuelve o encapsula llamadas a rutinas de biblioteca o cualquier otro código que está siendo reutilizado. Sinónimo de envoltorio [Rumbaugh et al., 1991].

E.1 Referencias

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[AENOR, 1992] AENOR. "*Normas para la Gestión y el Aseguramiento de la Calidad*". AENOR, 1992.

[Andreu et al., 1996] Andreu, R., Ricart, J., Valor J. "*Estrategia y Sistemas de Información*". 2ª Edition. Serie de Management. McGraw-Hill, 1996.

[Alfonseca et al., 1990] Alfonseca, M., Sancho, J., Martínez Orga, M. "*Teoría de Lenguajes, Gramáticas y Autómatas*". Ediciones Universidad y Cultura, 1990.

[Berard, 1996] Berard, E. V. "*Basic Object-Oriented Concepts*". The Object Agency, Inc. 1996.

[Bertino y Martino, 1993] Bertino, E., Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

[Boehm, 1989] Boehm, B. "*Tutorial on Software Risk Management*". IEEE Computer Society Press, 1989.

[Boehm, 1996] Boehm, R. "*Function Point FAQ*". Software Composition Technologies, Inc. June 30, 1996.

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[CERN, 1997] CERN. "*STING Software Engineering Glossary*". CERN. http://dxsting.cern.ch/sting/glossary.html. April 1997.

[DoD, 1992] DoD. "*DoD Software Reuse Initiative Vision and Strategy*". DOD, 1992.

[DoD 1994] DoD. "*Military Standard 498: Software Development and Documentation*". Departament of Defense of the United States of America, 1994.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Freeman, 1987] Freeman, P. "*A Perspective on Reusability*". IEEE Tutorial: Software Reusability (ed. P. Freeman), IEEE Computer Society Press: 2-8. 1987.

[García y Maudes, 1996] García Peñalvo, F. J., Maudes Raedo, J. M. "*Introducción a los Algoritmos Genéticos*". ALI Base, Nº28, pp. 49-52. Abril, 1996.

[Ghezzi et al., 1991] Ghezzi, C., Jazayeri, M., Mandrioli, D. "*Fundamentals of Software Engineering*". Prentice-Hall International Inc., 1991.

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd Edition. Addison-Wesley, 1994.

[Hamilton, 1997] Hamilton, G. "*Java Beans Version 1.01*". Sun Microsystems. July 1997.

[Horan, 1995] Horan, P. "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[Hsia et al., 1993] Hsia, P., Davis, A., Kung, D. "*Status Report: Requirements Engineering*". IEEE Software, 10(6):75-79, November 1993.

[IEEE, 1990] IEEE. "*IEEE Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12–1990*". Institute of Electrical and Electronics Engineers, 1990.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Joyanes, 1998] Joyanes Aguilar, L. "*Programación Orientada a Objetos*". 2ª Edición. McGraw-Hill, 1998.

[Krueger, 1992] Krueger, C. W. "*Software Reuse*". ACM Computing Surveys, 24(2):131-183. June 1992.

[Kuhn, 1971] Kuhn, T. S. "*La Estructura de las Revoluciones Científicas*". Fondo de Cultura Económica, 1971.

[Lucey, 1991] Lucey, T. "*Management Information Systems*". DP Publications, 1991.

[Madany et al., 1991] Madany, P. W., Islam, N., Kougiouris, P., Campbell, R. H. "*Reification and Reflection in C++: An Operating Systems Perspective*". Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801, USA, 1991.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[Martin, 1992] Martin, R. C. "*Abstract Classes and Pure Virtual Functions*". C++ Report. June/July 1992.

[Metamodel, 1997] "*Metamodelling Glossary*". http://www.metamodel.com/glossary.html. 1997.

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Monforte, 1995] Monforte, M. "*Sistemas de Información para la Dirección*". Ediciones Pirámide, 1995.

[Myers, 1978] Myers, G. "*Composite/Structured Design*". Van Nostrand Reinhold, 1978.

[NIST, 1994] National Institute of Standards and Technology (NIST). "*Glossary of Software Reuse Terms*". NIST, http://sw-eng.falls-church.va.us/ReuseIC/pubs/reference/terminology.htm, December 1994.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://uml.shl.com:80/docs/UML1.3/99-06-08-pdf. June 1999.

[Peña, 1998] Peña Marí, R. "*Diseño de Programas. Formalismo y Abstracción*". 2ª Edición. Prentice-Hall, 1998.

[Piattini, 1996] Piattini Velthuis, M. G. "*Tecnología Orientada al Objeto*". En las notas del curso Tecnología Orientada al Objeto. ALI-CyL, Valladolid, Noviembre 1996.

[Piattini et al., 1996] Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pressman, 1992] Pressman, R. S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1999] Rumbaugh, J.; Jacobson, I.; Booch, G. "*The Unified Modeling Language. Reference Manual*". Addison-Wesley, Object Technology Series, 1999.

[Sarle, 1996] Sarle, W. S. "*Neural Network FAQ*". June 1996.

[Schmerl, 1996] Schmerl, B. "*Configuration Management and Version Control*". http://tuvalu.csflinders.edu.au/seweb/scm/lectures/cmvc1.html. 23 May 1996.

[Shaw y Garlan, 1995] Shaw, M., Garlan, D. "*Formulations and Formalisms in Software Arquitecture*". Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.

[Szyperski y Pfister, 1996] Szyperski, C., Pfister, C. "*First International Workshop on Component-Oriented Programming WCOP'96*". 8 July 1996.

[Tilley et al., 1995] Tilley, Scott, Smith, R., Dennis B. "*Perspectives on Legacy System Reengineering*". Software Engineering Institute. Draft – Version 0.3. Carnegie Mellon University, Pittsburgh. 1995.

[Web, 1997] Principia Cybernetica Web. "*Web Dictionary of Cybernetics and Systems*". http://pespmc1.vub.ac.be/ASC. 1997.

[Yourdon, 1989] Yourdon, E. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Yourdon y Constantine, 1979] Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.