

## 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)
<b>Ingeniería del Software</b>	<b>I.T.I.S</b>	<b>3º</b>	<b>Obligatoria</b>	<b>6 (4,5T + 1,5P)</b>
Programación Orientada a Objetos	I.T.I.S	3º	Optativa	6 (3T + 3P)
Proyecto	I.T.I.S	3º	Obligatoria	9 (9P)
<b>Análisis de Sistemas</b>	<b>I.I</b>	<b>1º</b>	<b>Troncal</b>	<b>9 (6T + 3P)</b>
<b>Administración de Proyectos Informáticos</b>	<b>I.I</b>	<b>2º</b>	<b>Troncal</b>	<b>9 (6T + 3P)</b>
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
<b>Ingeniería del Software</b>	<b>T1, T2, T3, T4, T6, T7, T9</b>	<b>P1, P2, P4</b>	<b>H1, H2, H3, H4</b>
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

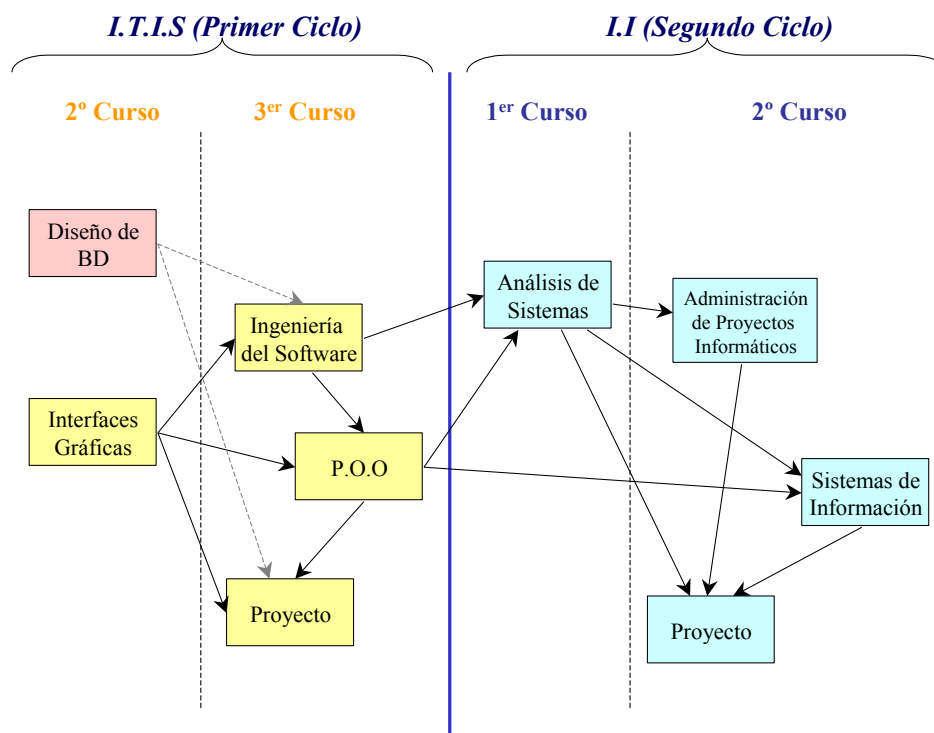


Figura 5.1. Dependencias entre las asignaturas que forman la unidad docente

## 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 ( <a href="mailto:fgarcia@usal.es">fgarcia@usal.es</a> )
Página web de la asignatura	<a href="http://tejo.usal.es/~fgarcia/docencia.html">http://tejo.usal.es/~fgarcia/docencia.html</a>

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).



<b>T1</b>	Descripción de las actividades técnicas e ingenieriles que se llevan a cabo en el ciclo de vida de un producto software.
<b>T2</b>	Descripción de los problemas, principios, métodos y tecnologías asociadas con la Ingeniería del Software.
<b>T3</b>	Importancia de los requisitos en el ciclo de vida del software.
<b>T4</b>	Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
<b>T6</b>	Método de análisis/diseño estructurado.
<b>T7</b>	Método de análisis/diseño orientado a objetos.
<b>T9</b>	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

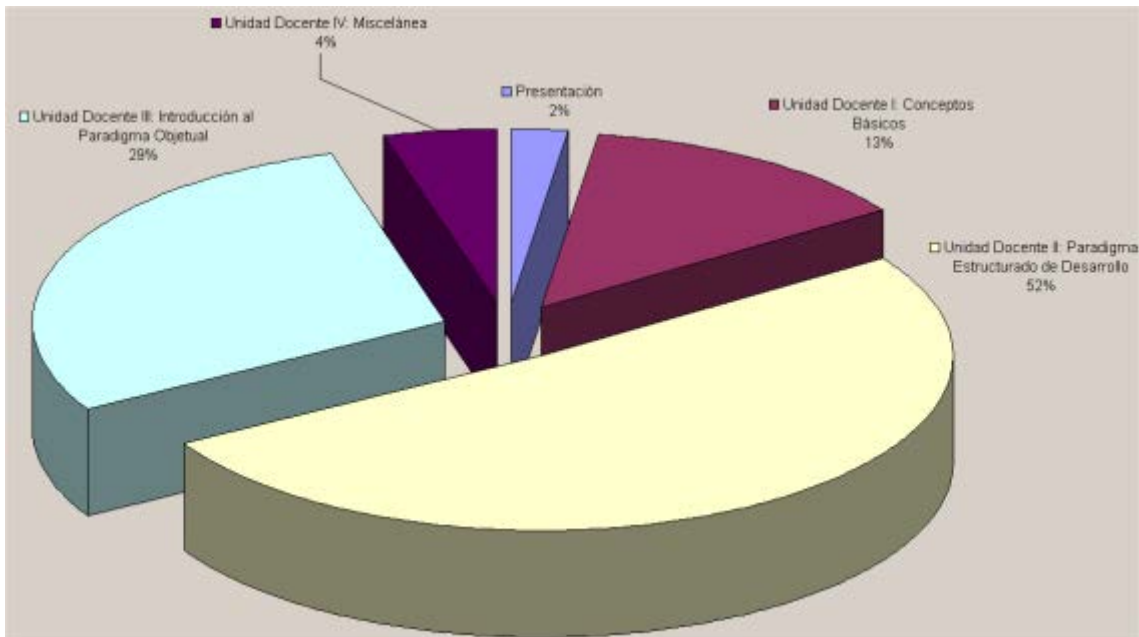
Para lograr los objetivos marcados, el programa de la parte teórica de esta asignatura se compone de nueve temas, organizados en cuatro unidades docentes, como se muestra en la Tabla 5.5.

<b>Presentación de la asignatura (1 Hora)</b>
<b>Unidad Didáctica I: Conceptos básicos (6 Horas)</b>
Tema 1. Introducción a la Ingeniería del Software (6 Horas)
<b>Unidad Didáctica II: Paradigma estructurado de desarrollo (23 Horas)</b>
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)
<b>Unidad Didáctica III: Introducción al paradigma objetual (13 Horas)</b>
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)
<b>Unidad Didáctica IV: Miscelánea (2 Horas)</b>
Tema 9. Herramientas CASE (2 Horas)

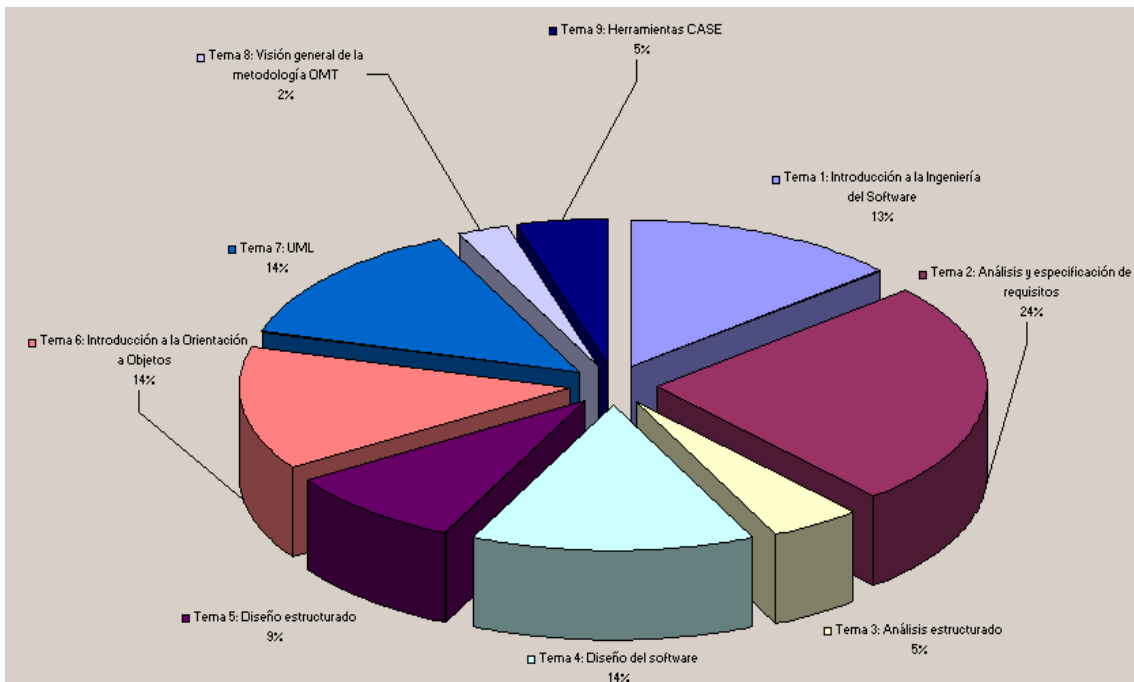
**Tabla 5.5. Estructura del programa de teoría de la asignatura Ingeniería del Software (4,5 créditos)**

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,

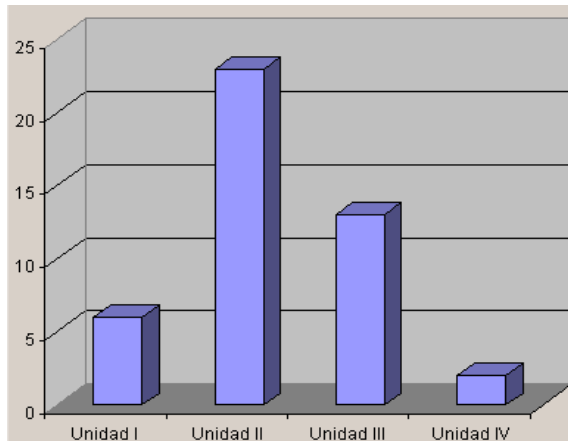
2001], y totalmente actualizado en la página web de la asignatura (<http://tejo.usal.es/~fgarcia/docencia.html>).



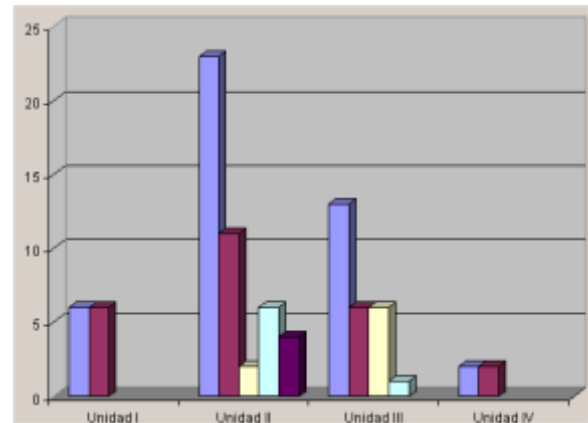
**Figura 5.2. Reparto porcentual de las horas de teoría de Ingeniería del Software entre las unidades didácticas**



**Figura 5.3. Reparto de las horas de teoría de Ingeniería del Software entre los temas**

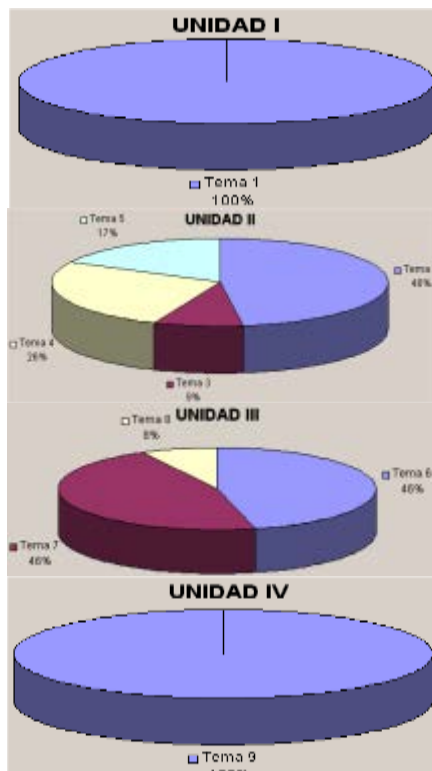


**Figura 5.4. Distribución temporal por unidades didácticas**



**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:

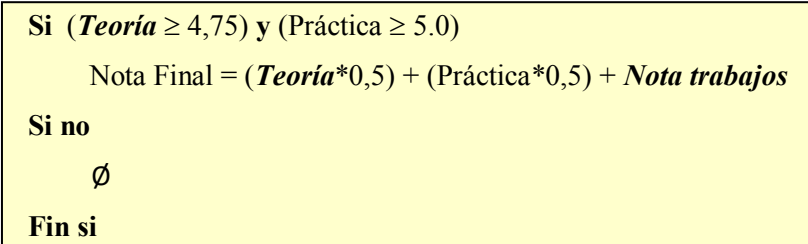
- Obliga al profesor a realizar diferentes cambios de ritmo, de tono... para no caer en la monotonía del pasar transparencias.

- 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** ( $Teoría \geq 4,75$ ) y ( $Práctica \geq 5.0$ )

$$Nota\ Final = (Teoría * 0,5) + (Práctica * 0,5) + Nota\ trabajos$$

**Si no**

$\emptyset$

**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.
- **Descriptorios:** 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**

<b>1.1 Software</b>
<b>1.2 Sistemas de información</b>
<b>1.3 Ingeniería del Software</b>
<b>1.4 Ciclo de vida del software</b>
<b>1.5 Paradigmas de la Ingeniería del Software</b>
<b>1.6 Metodologías</b>

**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 aquellas 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/Reuse/C/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., Nejme, 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 Crisis... 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. Salto 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*”. 2<sup>nd</sup> 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](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*”. 5<sup>th</sup> 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

<b>2.1 Introducción al análisis</b>
<b>2.2 Especificación de requisitos del software</b>
<b>2.3 Técnicas de especificación</b>
<b>2.4 Modelado funcional</b>
<b>2.5 Modelado de la información</b>
<b>2.6 Modelado del comportamiento dinámico</b>
<b>2.7 Balanceo de modelos</b>

**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 que 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 4<sup>th</sup> 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*”. 2<sup>a</sup> Edición. Prentice Hall, 1997.
- Hawryszkiewicz, 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 Informatik 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 Modelo Relacional*”. 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*”. 5<sup>th</sup> 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*”. 2<sup>a</sup> 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**

<b>3.1 Introducción</b>
<b>3.2 Enfoque de modelado clásico</b>
<b>3.3 Enfoque de modelado de Yourdon</b>

**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 ciclómica, 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**

<b>4.1 Introducción</b>
<b>4.2 Proceso de diseño</b>
<b>4.3 Actividades de diseño</b>
<b>4.4 Fundamentos de diseño</b>
<b>4.5 Diseño modular</b>

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 dividido 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*”. Rama, 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*”. 2<sup>nd</sup> Edition. Prentice Hall, 1992.
- Booch, G.** “*Análisis y Diseño Orientado a Objetos con Aplicaciones*”. 2ª Edición. Addison-Wesley/Díaz 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 Informatik 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/libro.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 Decomposing 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*”. 5<sup>th</sup> edition. McGraw-Hill, 2002.
- Shaw, M., Garlan, D.** “*Formulations and Formalisms in Software Architecture*”. Volume 1000-Lecture Notes in Computer Science, Springer-Verlag, 1995.
- Shneiderman, B.** “*Designing the User Interface. Strategies for Effective Human-Computer Interaction*”. 2<sup>nd</sup> 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

<b>5.1 Introducción</b>
<b>5.2 Diagrama de estructuras</b>
<b>5.3 Estrategias de diseño</b>

**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 transformación, 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*”. 2<sup>nd</sup> 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 de modelar los objetos del negocio, o del dominio 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

<b>6.1 Introducción y evolución de la Orientación a Objetos</b>
<b>6.2 Modelo objeto</b>
<b>6.3 Análisis y diseño orientados a objetos</b>

**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/Díaz 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., Lorenzen, 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.

Girou, A. *“Objects and Binary Relations”*. Object Currents, 1(6), SIGS Publications, June 1996.

Girou, 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”*. 2<sup>nd</sup> 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*”. 5<sup>th</sup> 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., Opper, 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

<b>7.1 Introducción</b>
<b>7.2 Una visión general de UML</b>
<b>7.3 Diagramas de estructura</b>
<b>7.4 Diagramas de interacción</b>
<b>7.5 Casos de uso</b>

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., Övergaard, G.** “*Object Oriented Software Engineering: A Use Case Driven Approach*”. Addison-Wesley, 1992. Revised 4<sup>th</sup> 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](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].
- Recaltar 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

<b>8.1 Introducción</b>
<b>8.2 Fases</b>
<b>8.3 Análisis</b>
<b>8.4 Diseño del sistema</b>
<b>8.5 Diseño de los objetos</b>

**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 posteriores 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://arkhpl.kek.jp/managers/computing/activities/OO\\_CollectInfor/Methodologies/OMT/OMTBook/OMTBook.html](http://arkhpl.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 tratan 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*

<b>9.1 Introducción</b>
<b>9.2 Componentes de una herramienta CASE</b>
<b>9.3 Clasificación de las herramientas CASE</b>
<b>9.4 Integración de CASE</b>

**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*”. 5<sup>th</sup> 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).

<b>P1</b>	Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.
<b>P2</b>	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
<b>P4</b>	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.

<p><b>Si</b> (Teoría <math>\geq 4,75</math>) y (<b>Práctica</b> <math>\geq 5.0</math>)</p> <p style="text-align: center;">Nota Final = (Teoría*0,5) + (<b>Práctica</b>*0,5) + <i>Nota trabajos</i></p> <p><b>Sino</b></p> <p style="text-align: center;"><math>\emptyset</math></p> <p><b>Fin si</b></p>
--

**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.

<b>Asignatura</b>	Análisis de Sistemas (troncal)
<b>Créditos</b>	6T + 3P
<b>Estudios</b>	Ingeniería Informática (2º ciclo)
<b>Plan</b>	B.O.E de 1-7-1999
<b>Curso</b>	1º
<b>Cuatrimestre</b>	1º y 2º (anual)
<b>Responsables</b>	Francisco José García Peñalvo ( <a href="mailto:fgarcia@usal.es">fgarcia@usal.es</a> ) María N. Moreno García ( <a href="mailto:mmg@usal.es">mmg@usal.es</a> )
<b>Página web de la asignatura</b>	<a href="http://lisisu02.usal.es/~mmoreno/analisis.html">http://lisisu02.usal.es/~mmoreno/analisis.html</a>

**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].

<b>T3</b>	Importancia de los requisitos en el ciclo de vida del software.
<b>T4</b>	Obtención, documentación, especificación y prototipado de los requisitos de un sistema software.
<b>T5</b>	Especificaciones formales de requisitos.
<b>T6</b>	Método de análisis /diseño estructurado.
<b>T7</b>	Método de análisis /diseño orientado a objetos.
<b>T12</b>	Conceptos, métodos, procesos y técnicas destinadas al mantenimiento y evolución de los sistemas software.
<b>T13</b>	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

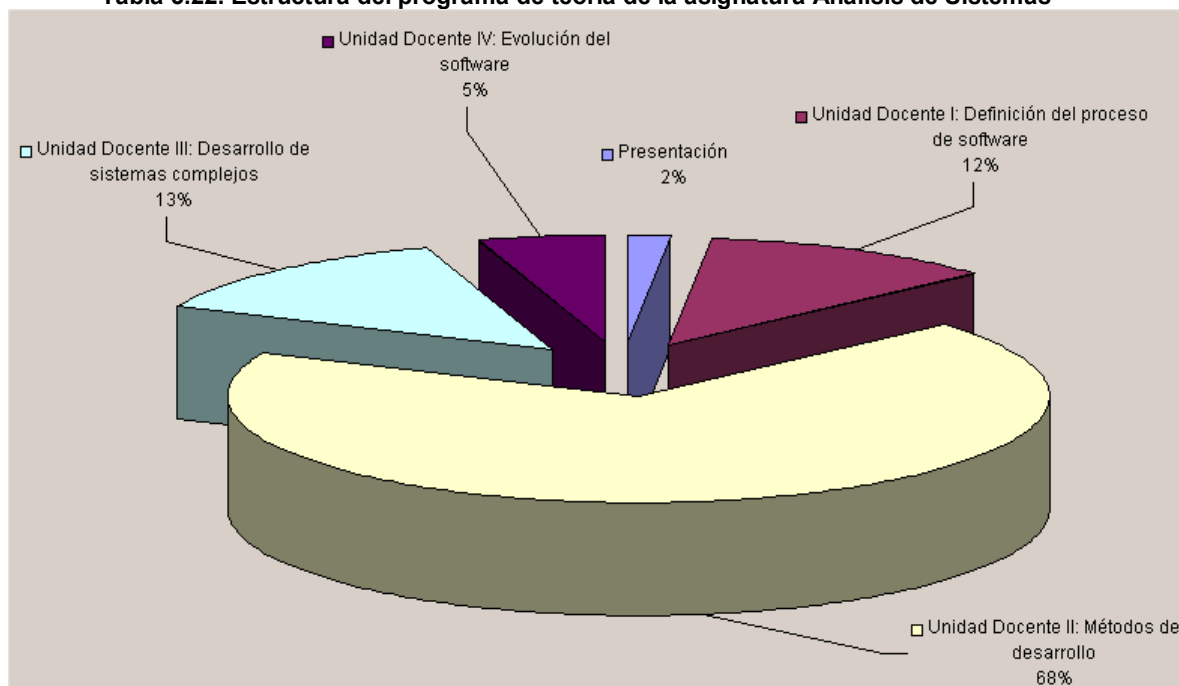
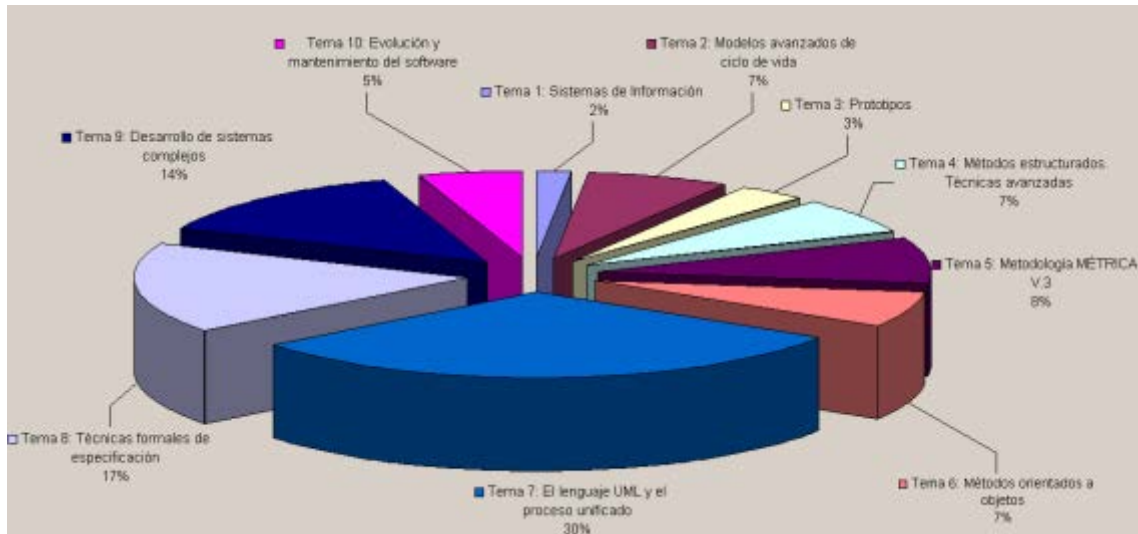
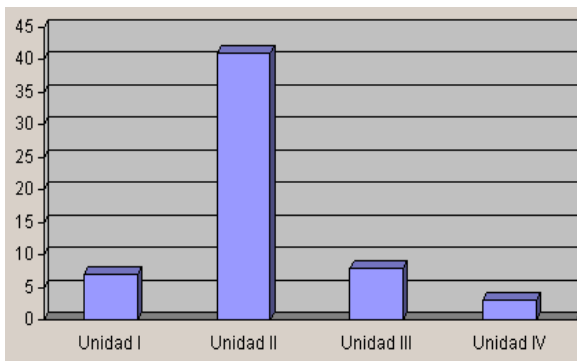


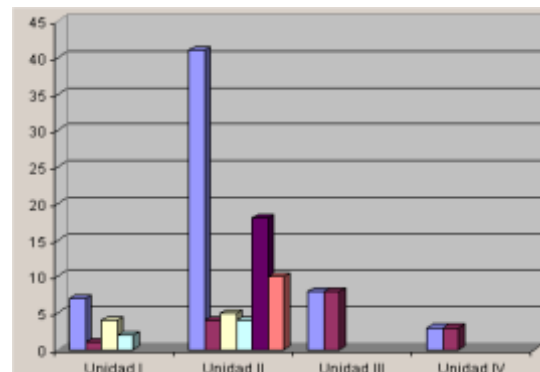
Figura 5.8. Reparto porcentual de las horas de teoría de Análisis de Sistemas entre las unidades didácticas



**Figura 5.9. Reparto de las horas de teoría de Análisis de Sistemas entre los temas**

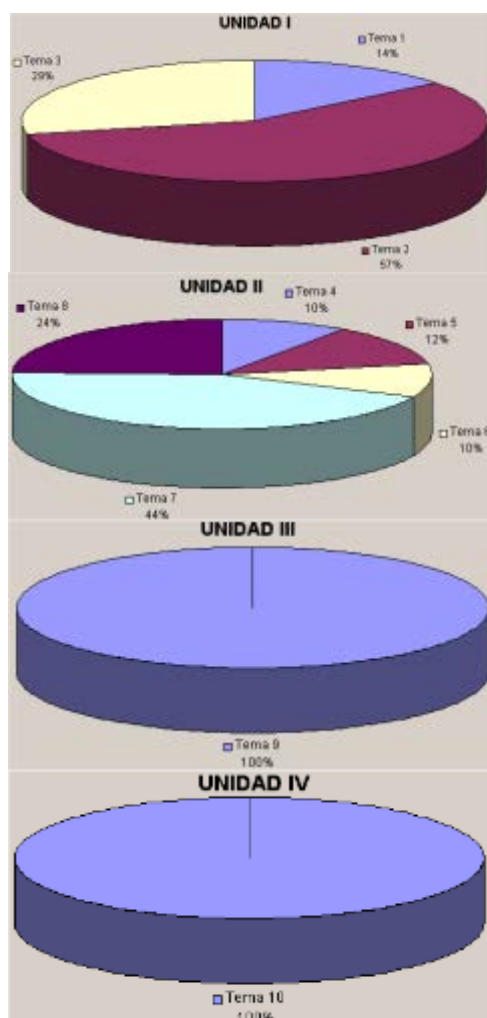


**Figura 5.10. Distribución temporal por unidades didácticas**



**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.

<p><b>Si</b> (<i>Teoría</i> <math>\geq 4,75</math>) y (<i>Práctica</i> <math>\geq 5.0</math>)</p> <p style="text-align: center;">Nota Final = ((<i>Teoría</i>*0,6) + (<i>Práctica</i>*0,3)) * 10/9 + Nota trabajos</p> <p><b>Sino</b></p> <p style="text-align: center;"><math>\emptyset</math></p> <p><b>Fin si</b></p>
--

**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., Lorenzen, 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. Pressman 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**

<b>1.1 Historia y evolución</b>
<b>1.2 Elementos de un sistema de información</b>
<b>1.3 Estructura de un sistema de información</b>
<b>1.4 Tipos de sistemas de información automatizados</b>
<b>1.5 Principios generales de sistemas</b>

**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

<b>2.1 Concepto de ciclo de vida</b>
<b>2.2 Procesos del ciclo de vida</b>
<b>2.3 Modelos de ciclo de vida</b>

**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. Salto 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*”. 2<sup>nd</sup> 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*”. 2<sup>a</sup> 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*”. 5<sup>th</sup> 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**

<b>3.1 ¿Qué es un prototipo?</b>
<b>3.2 Enfoques en la construcción de prototipos</b>
<b>3.3 Tipos de prototipos</b>
<b>3.4 Usos de los prototipos</b>
<b>3.5 Guías para el desarrollo de prototipos</b>
<b>3.6 Beneficios de los prototipos</b>
<b>3.7 Herramientas para la construcción de prototipos</b>

**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*”. 2<sup>nd</sup> 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*”. 2<sup>a</sup> 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*”. 5<sup>th</sup> 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**

<b>4.1 Introducción</b>
<b>4.2 Perspectiva histórica</b>
<b>4.3 Clasificación de los métodos</b>
<b>4.4 Principales métodos de desarrollo</b>
<b>4.5 Técnicas de modelado</b>

**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*”. 5<sup>th</sup> 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

<b>5.1 Introducción</b>
<b>5.2 Estructura de la metodología</b>
<b>5.3 Planificación de Sistemas de Información</b>
<b>5.4 Desarrollo de Sistemas de Información</b>
<b>5.5 Mantenimiento de Sistemas de Información</b>
<b>5.6 Procesos de interfaz</b>

**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.** “*Eurométodo 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

<b>6.1 El paradigma de la orientación al objeto</b>
<b>6.2 Clasificación de los métodos</b>
<b>6.3 Descripción de algunos métodos</b>

**Tabla 5.30. Contenidos del tema seis 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 el área de conocimiento **Software Engineering Tools and Methods** [Carrington, 2001] del **SWEBOK** (*Software Engineering Body of Knowledge*) [Abran et al., 2001].

### Resumen

El tercer tema de esta unidad didáctica constituye el primero de los dos dedicados al estudio de los 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 aquellas 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 [Rumbaugh 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/Díaz 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*”. 2<sup>nd</sup> 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., Övergaard, G.** “*Object Oriented Software Engineering: A Use Case Driven Approach*”. Addison-Wesley, 1992. Revised 4<sup>th</sup> 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*”. 2<sup>a</sup> 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*”. 5<sup>th</sup> 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., Opper, 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

<b>7.1 Génesis y evolución de UML</b>
<b>7.2 Visión general de UML</b>
<b>7.3 Elementos de modelado</b>
<b>7.4 Las vistas</b>
<b>7.5 Vista estática</b>
<b>7.6 Vista de casos de uso</b>
<b>7.7 Vista de máquina de estados</b>
<b>7.8 Vista de actividad</b>
<b>7.9 Vista de interacción</b>
<b>7.10 Vistas físicas</b>
<b>7.11 Vista de gestión de modelo</b>
<b>7.12 Proceso unificado de desarrollo de software</b>

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, 5 y 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*”. 2<sup>nd</sup> 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. (3<sup>rd</sup>-4<sup>th</sup> 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., Övergaard, G.** “*Object Oriented Software Engineering: A Use Case Driven Approach*”. Addison-Wesley, 1992. Revised 4<sup>th</sup> 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*”. 2<sup>nd</sup> 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., Lorenzen, 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 Technology 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; precondition; poscondition; 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

<b>8.1 Introducción</b>
<b>8.2 Base matemática de los métodos formales</b>
<b>8.3 Lenguajes formales de especificación</b>
<b>8.4 Lenguaje Z</b>
<b>8.5 OCL</b>
<b>8.6 Redes de Petri</b>
<b>8.7 Redes de Petri Coloreadas</b>

**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 contrapuestos [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 I*”. 2<sup>nd</sup> edition. Springer Verlag, 1996. Second corrected printing 1997.
- Lightfoot, D.** “*Formal Specification Using Z*”. 2<sup>nd</sup> 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*”. 5<sup>a</sup> 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*”. 6<sup>a</sup> 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 4<sup>th</sup> 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.
- Craig, 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 Formal 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*”. 2<sup>nd</sup> edition. Prentice-Hall, 1991.
- Johnson, S. D., Alexander, W. P., Chin, S.-K., Gopalakrishnan, G.** “*Report on the 21<sup>st</sup> 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*”. 5<sup>th</sup> 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

<b>9.1 Sistemas de tiempo real</b>
<b>9.2 Sistemas Cliente/Servidor (C/S)</b>
<b>9.3 Ingeniería Web</b>
<b>9.4 Sistemas Interoperativos</b>
<b>9.5 Sistemas para el soporte de decisiones</b>

**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 proyecto 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

gestión 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.
- Pressman, 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, M<sup>a</sup> 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., Övergaard, G.** "Object Oriented Software Engineering: A Use Case Driven Approach". Addison-Wesley, 1992. Revised 4<sup>th</sup> 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, N°85:40-45. 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*”. 2<sup>nd</sup> edition. John 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 reingeniería. 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**

<b>10.1 Introducción</b>
<b>10.2 Actividades de mantenimiento</b>
<b>10.3 El proceso de mantenimiento</b>
<b>10.4 Problemas del mantenimiento</b>
<b>10.5 Costes de mantenimiento</b>
<b>10.6 Herramientas y técnicas</b>
<b>10.7 Ingeniería inversa y reingeniería</b>

**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 Reengineering 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*”. 5<sup>th</sup> 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).

<b>P1</b>	Aplicar de forma práctica los conceptos teóricos sobre el desarrollo estructurado.
<b>P2</b>	Aplicar de forma práctica los conceptos teóricos de Orientación a Objetos.
<b>P4</b>	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.

<p><b>Si</b> (Teoría <math>\geq 4,75</math>) y (<b>Práctica</b> <math>\geq 5.0</math>)</p> <p style="padding-left: 40px;">Nota Final = ((Teoría*0,6) + (<b>Práctica</b>*0,3)) * 10/9 + Nota trabajos</p> <p><b>Sino</b></p> <p style="padding-left: 40px;"><math>\emptyset</math></p> <p><b>Fin si</b></p>
--

**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.

<b>Asignatura</b>	Administración de Proyectos Informáticos (troncal)
<b>Créditos</b>	6T + 3P
<b>Estudios</b>	Ingeniería Informática (2º ciclo)
<b>Plan</b>	B.O.E de 1-7-1999
<b>Curso</b>	2º
<b>Cuatrimestre</b>	1º y 2º (anual)
<b>Responsable</b>	María N. Moreno García ( <a href="mailto:mmg@usal.es">mmg@usal.es</a> )
<b>Página web de la asignatura</b>	<a href="http://lisisu02.usal.es/~mmoreno/api.html">http://lisisu02.usal.es/~mmoreno/api.html</a>

**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 prerequisite 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].

<b>T10</b>	Gestión de proyectos software: definición de objetivos, gestión de recursos, estimación de esfuerzo y coste, planificación y gestión de riesgos.
<b>T11</b>	Uso de métricas software para el apoyo a la gestión de proyectos software y aseguramiento de la calidad del software.
<b>T12</b>	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

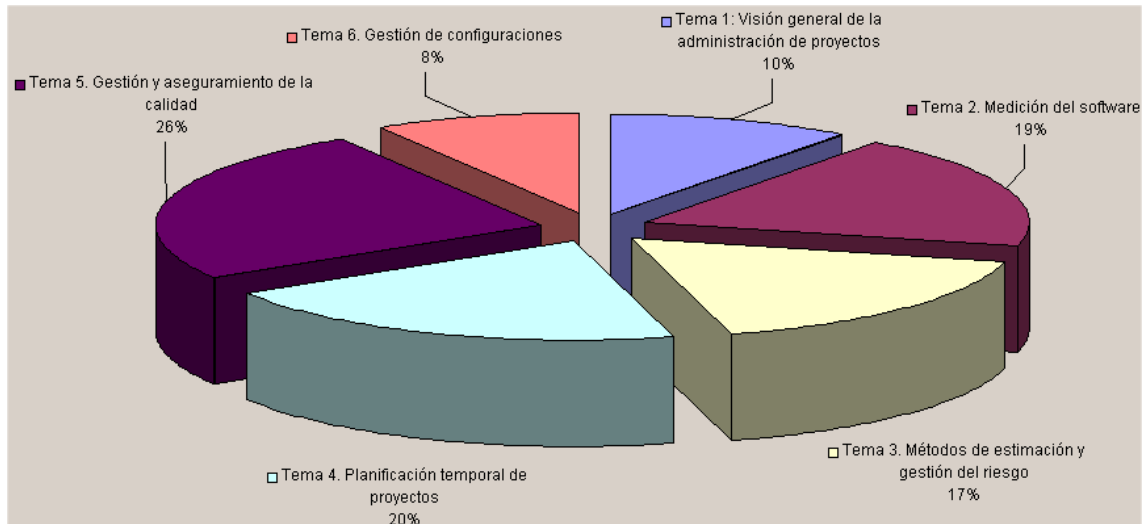
La primera hora de clase se dedica a la presentación de la asignatura, donde se realiza una breve presentación del temario de teoría y práctica de la misma. Los alumnos ya cuentan con el programa de la asignatura, resumido en la guía académica que se les entrega con la matrícula [GAFC-USAL, 2001], y 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.

<p><b>Si</b> (<i>Teoría</i> <math>\geq 4,75</math>) y (<i>Práctica</i> <math>\geq 5.0</math>)</p> <p>Nota Final = ((<i>Teoría</i>*0,6) + (<i>Práctica</i>*0,3)) * 10/9 + Nota trabajos</p> <p><b>Sino</b></p> <p><math>\emptyset</math></p> <p><b>Fin si</b></p>
--

**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, habiendo 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
<b>4.7 Método ROY (2 Horas)</b>
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
<b>Tema 5. Gestión y aseguramiento de la calidad (15 horas)</b>
<b>5.1 Concepto de calidad (15 Minutos)</b>
<b>5.2 Definición de calidad del software (15 Minutos)</b>
<b>5.3 Aspectos de la gestión de calidad (0,5 Horas)</b>
<b>5.4 Ámbitos de la gestión de calidad (2 Horas)</b>
5.4.1 Calidad en el ámbito de la organización
5.4.2 Calidad en el ámbito del proyecto
<b>5.5 Marco normativo (3 Horas)</b>
5.5.1 Estándares ISO 9000
5.5.2 Estándares IEEE
<b>5.6 Actividades de aseguramiento de la calidad (2 Horas)</b>
<b>5.7 Evaluación de la calidad (5 Horas)</b>
5.7.1 Modelos de calidad del software
5.7.2 Fiabilidad del software
5.7.3 Revisiones
<b>5.8 Métricas de calidad (2 Horas)</b>
<b>Tema 6. Gestión de configuraciones (5 horas)</b>
<b>6.1 Introducción (0,5 Horas)</b>
<b>6.2 Configuración de referencia (0,5 Horas)</b>
<b>6.3 Funciones de la gestión de la configuración (0,5 Horas)</b>
<b>6.4 Identificación de la configuración (1 Hora)</b>
<b>6.5 Control de la configuración (1,5 Horas)</b>
<b>6.6 Auditoría de la configuración (0,5 Horas)</b>
<b>6.7 Contabilidad del estado de la configuración (0,5 Horas)</b>

**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**

<b>1.1 Bases del desarrollo de software</b>
<b>1.2 Madurez del proceso de software</b>
<b>1.3 Organización de un proyecto</b>
<b>1.4 Herramientas de ayuda</b>
<b>1.5 Teoría W</b>

**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:

- Recaltar 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**

<b>2.1 Conceptos básicos</b>
<b>2.2 Medidas y modelos</b>
<b>2.3 Alcance de las métricas del software</b>
<b>2.4 Clasificación de las métricas de software</b>
<b>2.5 Recogida de datos métricos</b>
<b>2.6 Medición de atributos internos del producto</b>
<b>2.7 Medición de atributos externos del producto</b>
<b>2.8 Medición de recursos</b>
<b>2.9 Métricas para sistemas orientados a objetos</b>

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**

<b>3.1 Introducción</b>
<b>3.2 Precisión y exactitud de las estimaciones</b>
<b>3.3 Principios de estimación</b>
<b>3.4 Estimación de costes</b>
<b>3.5 Modelos de coste y esfuerzo</b>
<b>3.6 Las estimaciones en el CMM</b>
<b>3.7 Gestión de riesgos</b>

**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 (*CO*nstructive *CO*st *MO*del), 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 5<sup>th</sup> 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**

<b>4.1 Introducción</b>
<b>4.2 Notaciones gráficas</b>
<b>4.3 Grafos: conceptos básicos</b>
<b>4.4 Métodos de planificación temporal</b>
<b>4.5 Método PERT</b>
<b>4.6 Método CPM</b>
<b>4.7 Método ROY</b>

**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*”. 3<sup>rd</sup> 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**

<b>5.1 Concepto de calidad</b>
<b>5.2 Definición de calidad del software</b>
<b>5.3 Aspectos de la gestión de calidad</b>
<b>5.4 Ámbitos de la gestión de calidad</b>
<b>5.5 Marco normativo</b>
<b>5.6 Actividades de aseguramiento de la calidad</b>
<b>5.7 Evaluación de la calidad</b>
<b>5.8 Métricas de calidad</b>

**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.** “*Plan General 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**

<b>6.1 Introducción</b>
<b>6.2 Configuración de referencia</b>
<b>6.3 Funciones de la gestión de la configuración</b>
<b>6.4 Identificación de la configuración</b>
<b>6.5 Control de la configuración</b>
<b>6.6 Auditoría de la configuración</b>
<b>6.7 Contabilidad del estado de la configuración</b>

**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*”. 2<sup>nd</sup> 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](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).

<b>P3</b>	Aplicar de forma práctica los conceptos teóricos sobre gestión de proyectos.
<b>P8</b>	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 cabo 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*.

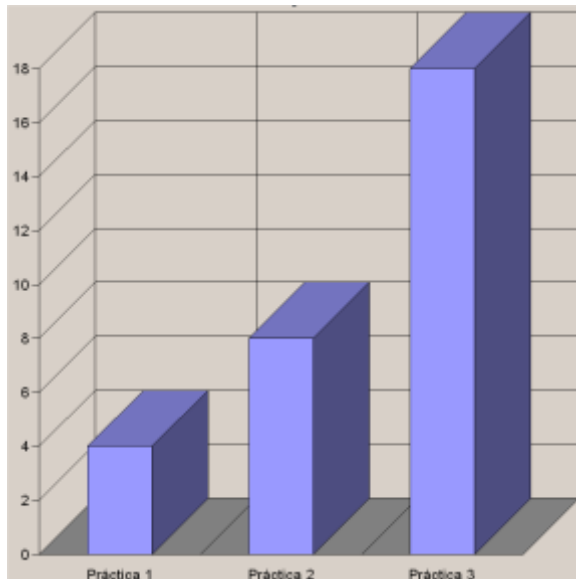


Figura 5.16. Reparto de las horas entre las prácticas de Administración de Proyectos Informáticos

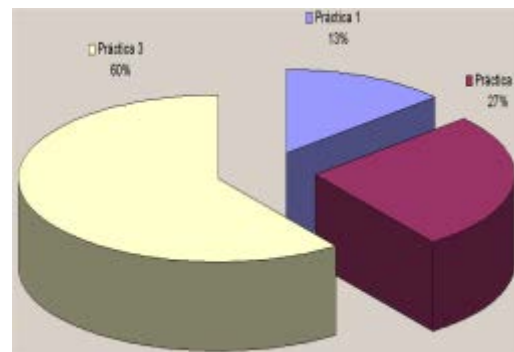


Figura 5.17. Reparto porcentual de las horas prácticas de Administración de Proyectos Informáticos

### 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  $\geq 4,75$ ) y (**Práctica**  $\geq 5.0$ )  
 Nota Final = ((Teoría\*0,6) + (**Práctica**\*0,3)) \* 10/9 + Nota trabajos  
**Sino**  
 $\emptyset$   
**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 propuesto 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](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., Övergaard, G.** “*Object Oriented Software Engineering: A Use Case Driven Approach*”. Addison-Wesley, 1992. Revised 4<sup>th</sup> printing, 1993.
- 📖 **Jacobson, I. Ericsson, M., Jacobson, A.** “*The Object Advantage-Business Process Reengineering 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, M<sup>a</sup> 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*”. 2<sup>nd</sup> edition. John 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-Executive Editor), Moore, J. W. (Co-Executive 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-Díaz, 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., MacLeod, G. J., Merritt, M. J. “*Characteristics of Software Quality*”. North-Holland, 1980.
- [Boehm et al., 1984] Boehm, B. W., Gary, T. E., Sewaldt, T. “*Prototyping versus Specifying: 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*”. 2<sup>nd</sup> 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 4<sup>th</sup> 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*”. 2<sup>nd</sup> 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 Augmented 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 4<sup>th</sup> 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](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](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](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 Process Improvement 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, M<sup>a</sup> N., García-Bermejo Giner, J. R., Luis Reboledo, 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, M<sup>a</sup> N., Moreno Montero, Á. M<sup>a</sup>, 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/iscdi01/informeTecnico.html>). Páginas 1-9. Diciembre, 2001.

[García et al., 2001b] **García, F. J., Moreno, M. N., Moreno, A. M<sup>a</sup>, 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-I 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.

[Gibb, 1988] **Gibb, 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*”. 5<sup>th</sup> 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*”. 2<sup>nd</sup> 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 Programming, 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. “*Information 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., Övergaard, G.** “*Object Oriented Software Engineering: A Use Case Driven Approach*”. Addison-Wesley, 1992. Revised 4<sup>th</sup> printing, 1993.
- [**Jacobson et al., 1994**] **Jacobson, I. Ericsson, M., Jacobson, A.** “*The Object Advantage- Business Process Reengineering with Object Technology*”. Addison Wesley, 1994.
- [**Jacobson et al., 1997**] **Jacobson, I., Griss, M., Patrik, J.** “*Software 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*”. 2<sup>nd</sup> edition. Springer Verlag, 1996. Second corrected printing 1997.
- [**Jensen, 1997b**] **Jensen, K.** “*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2*”. 2<sup>nd</sup> edition. Springer Verlag, 1996. Second corrected printing 1997.
- [**Jensen, 1997c**] **Jensen, K.** “*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3*”. 2<sup>nd</sup> 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*”. 3<sup>rd</sup> 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*”. Communications 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*”. 5<sup>th</sup> 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 d'ordonnement*”. 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., Lorenzen, 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 2<sup>nd</sup> 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 6<sup>th</sup> 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.