Capítulo 4
Definición del Proyecto Docente

El desarrollo de un Proyecto Docente para una determinada disciplina implica, en primer lugar, la identificación clara y precisa del conjunto de conceptos y conocimientos científico/técnicos que la misma engloba. En el caso de la Ingeniería del Software esta es una tarea que presenta algunas dificultades debido al amplio conjunto de materias que abarca y a la constante evolución a la que se ve sometida como consecuencia de los rápidos cambios que sufre la tecnología del software. Por este motivo, previo al desarrollo de la propuesta concreta de contenido, se presenta una visión global de la Ingeniería del Software como disciplina mediante un análisis de su marco histórico y conceptual. El objetivo no es realizar una exposición exhaustiva, sino mostrar aquellos conceptos, aportaciones y resultados que se consideran relevantes para el contenido de las asignaturas objeto de la propuesta docente. Sin duda, una característica de la Ingeniería del Software como disciplina universitaria, frente a otras disciplinas más establecidas, es el dinamismo con el que cambian sus conceptos y herramientas. Por ello es importante atender a las recomendaciones sobre la enseñanza de la Ingeniería del Software que realizan los organismos y organizaciones internacionales relacionados con la Ciencia de la Computación, los Sistemas de Información y la propia Ingeniería del Software que también se recogen en este capítulo. Atender a este tipo de recomendaciones permite que los conocimientos transmitidos y las habilidades adquiridas por los alumnos respondan a las necesidades profesionales reales y no queden obsoletos en poco tiempo. Finalizamos este capítulo analizando el contexto profesional en el que se van a mover los futuros titulados con el fin de tener presentes estos aspectos en el diseño de las asignaturas y atender de esa forma las demandas de la sociedad.

4.1 Introducción

Es un hecho ampliamente asumido que la Informática es hoy en día un factor social de gran relevancia. El objetivo de las titulaciones donde se circunscribe la plaza objeto de concurso, la Ingeniería Técnica en Informática de Sistemas y la Ingeniería Informática, es la formación de profesionales (ingenieros informáticos) que se puedan incorporar a un mercado laboral, actualmente en plena demanda de estos titulados, con unas garantías plenas de calidad en cuanto a la función que deberán desempeñar en sus puestos de trabajo.

Actualmente, y a consecuencia de la falta de madurez que todavía sufre la Informática, es difícil precisar qué se entiende por un ingeniero informático, ya sea técnico o superior. Esta bruma que rodea a la figura de los informáticos es aprovechada por multitud de personas, ya sean tituladas o no, para acogerse a puestos laborales donde se interacciona de una manera u otra con un computador. Es por este motivo por el que se debe hacer un esfuerzo por diferenciar a los *ingenieros informáticos*[18] de los informáticos en general.

En este sentido, y con un ámbito internacional, en 1998 se formó el Software Engineering Cordinating Committee (SWECC) bajo el auspicio de IEEE-CS y ACM. Su misión es cuidar la evolución de la Ingeniería del Software como una disciplina profesional dentro del mundo de la Informática. Para lo cual se pretende documentar el cuerpo de conocimientos de la disciplina, recomendar un criterio de acreditación de los titulados, desarrollar un modelo de currículo, mantener un código ético y definir un conjunto de estándares. Actualmente, los proyectos que se encuentran en marcha son:

- ***Software Engineering Body of Knowledge (SWEBOK) – Cuerpo de conocimiento de la Ingeniería del Software:*** Tiene como objetivos *clarificar y definir los límites de la ingeniería del software con respecto a otras disciplinas* y *ofrecer los fundamentos para el desarrollo de una propuesta curricular y el material para la certificación de los profesionales*. Este proyecto ha obtenido como resultado la guía con el cuerpo de conocimiento de Ingeniería del Software, que tras pasar por la denominada *Straw Man Version* (versión de hombre de paja) [Bourque et al., 1998], actualmente ha finalizado la *Stone Man Version v0.95* (versión del hombre de piedra) [Abran et al., 2001b]. En este momento se abre un período dos años para la utilización de esta guía, obteniéndose así la

[18] Como ya se ha comentado antes en este proyecto docente, cada vez existe una mayor tendencia en el ámbito internacional a denominar Ingeniería del Software a la titulación que en España equivaldría a la Ingeniería Informática, sin entrar en las diferenciaciones entre titulados de primer o segundo ciclo en nuestro país.

realimentación necesaria para comenzar en el tercer año la *Iron Man Versión* (versión del hombre de hierro).

- ***Software Engineering Education Project – Proyecto de Educación en Ingeniería del Software*** **[ACM/IEEE-CS, 1999a]*:*** Reúne dos subproyectos, *un modelo de acreditación para programas universitarios* [ACM/IEEE-CS, 1998] y *el plan para el proyecto de educación en Ingeniería del Software (Plan for the Software Engineering Education Project – SWEEP)* [ACM/IEEE-CS, 1999b].

- ***Software Engineering Code of Ethics and Professional Practice – Código de Ética y Práctica Procesional de Ingeniería de Software*** **[ACM/IEEE-CS, 1999c]*:*** El código ético de la profesión fue aprobado por las asociaciones ACM e IEEE-CS en su versión 5.2 [ACM/IEEE-CS, 1999c][19] como el estándar para la enseñanza y la práctica de la Ingeniería del Software. Este código ha sido desarrollado por un grupo dirigido por Donald Gotterbarn**,** siendo propuesto tras varias versiones[20] y después de revisar los códigos éticos de otras sociedades. El código ético contiene ocho principios relacionados con el comportamiento y las decisiones tomadas por los profesionales. Un breve resumen del mismo se presenta en el Cuadro 4.1.

Tras la presentación de los esfuerzos en pro de la definición de la profesión de *ingeniero informático* o *ingeniero del software* realizados por ACM e IEEE-CS, se vuelve a centrar la atención en el contexto nacional, y más concretamente en el de la Universidad de Salamanca, para definir de una forma más precisa el perfil de los titulados en Ingeniería Técnica en Informática de Sistemas e Ingeniería en Informática; de forma que éstos sean profesionales que *utilicen un sólido conjunto de fundamentos de Ciencias de la Computación para solventar problemas reales, interaccionando con clientes y usuarios, debiendo hacer uso de una capacidad de comunicación oral y escrita correcta y fluida, y actuando siempre de acuerdo al código ético marcado por su profesión*.

[19] En [Dolado, 1999] el Dr. D. Javier Dolado, de la Universidad de San Sebastián, ha traducido el código ético en su versión 5.2 al español.

[20] La versión del código ético más conocida, con anterioridad a la aprobación de la versión 5.2, fue la versión 3.0 [Gotterbarn et al., 1997a], [Gotterbarn et al., 1997b]. Precisamente en [Gotterbarn et al., 1999a] y en [Gotterbarn et al., 1999b] se comenta la versión 5.2 comparándola con la versión 3.0.

Los códigos éticos tienen una función esencial para caracterizar una profesión, y para que una disciplina adquiera el carácter de profesión debe poseer un código de conducta.

Se pueden resumir las principales funciones de los códigos éticos en los siguientes apartados [Bowyer, 1996]:

1) Simbolizar una profesión.
2) Proteger los intereses del grupo.
3) Inspirar buena conducta.
4) Educar a los miembros de la profesión.
5) Disciplinar a sus afiliados.
6) Fomentar las relaciones externas.
7) Enumerar los principios morales básicos.
8) Expresar los ideales a los que se debe aspirar.
9) Mostrar las reglas básicas de comportamiento.
10) Ofrecer guías de comportamiento.
11) Enumerar derechos y responsabilidades.

Los códigos de conducta van más allá de la pura normativa legal, ya que ayudan a guiar el comportamiento en multitud de situaciones para las que no existe referencia legal.

El código propuesto por ACM y IEEE-CS tiene como objetivo documentar las responsabilidades y obligaciones éticas y profesionales, en un intento de educar y aleccionar a los ingenieros del software, a la vez que informar al público sobre las responsabilidades que son importantes para la profesión. Este código ético prima el bienestar y la calidad de vida del público en general, en cuanto a todas las decisiones relacionadas con la Ingeniería del Software [Gotterbarn et al., 1999b].

Los ingenieros informáticos deben responsabilizarse de que al llevar a cabo sus actividades lo hagan con beneficio y respeto a la profesión que ejercen. De acuerdo a sus compromisos con la salud, la seguridad y el bien público, los ingenieros informáticos deben seguir los siguientes ocho principios [Gotterbarn, 1999]:

1. **Sociedad:** Los ingenieros del software actuarán de manera coherente con el interés general.
2. **Cliente y empresario:** Los ingenieros del software deberán actuar de tal modo que se sirvan los mejores intereses para sus clientes y empresarios, y consecuentemente con el interés general.
3. **Producto:** Los ingenieros del software deberán garantizar que sus productos y las modificaciones relacionadas con ellos cumplen los estándares profesionales de mayor nivel que sea posible.
4. **Juicio:** Los ingenieros del software deberán mantener integridad e independencia en su valoración profesional.
5. **Gestión:** Los gestores y líderes en Ingeniería del Software suscribirán y promoverán un enfoque ético a la gestión del desarrollo y el mantenimiento del software.
6. **Profesión:** Los ingenieros del software deberán progresar en la integridad y la reputación de la profesión, de forma coherente con el interés público.
7. **Compañeros:** Los ingenieros del software serán justos y apoyarán a sus compañeros.
8. **Persona:** Los ingenieros del software deberán participar en el aprendizaje continuo de la práctica de su profesión y promoverán un enfoque ético en ella.

Cuadro 4.1. Resumen del código ético de ACM/IEEE-CS, versión 5.2

4.2 El perfil de formación

Las actividades a realizar por la persona que ocupe la plaza a concurso se centran en impartir docencia en materia de Ingeniería del Software en las titulaciones de Ingeniería Técnica en Informática de Sistemas e Ingeniería Informática en la Universidad de Salamanca.

El siguiente paso es identificar qué asignaturas se ajustan al perfil de formación dentro de los Planes de Estudios vigentes en estas titulaciones, Plan de 1997 [BOE, 1997] y Plan de 1999 [BOE, 1999] para la Ingeniería Técnica y la Ingeniería Informática respectivamente.

En el Capítulo 2 ya se presentaron las asignaturas de dichos Planes de Estudios, concretamente, las tablas 2.17 y 2.18 muestran las asignaturas troncales/obligatorias y optativas respectivamente para la Ingeniería Técnica en Informática de Sistemas; mientras que las tablas 2.20 y 2.21 presentaban el mismo tipo de información para la Ingeniería Informática.

Una descripción más detallada de los Planes de Estudios vigentes para ambas titulaciones se puede encontrar en la *Guía Académica de la Facultad de Ciencias* para el Curso 2001-2002 [GAFC-USAL, 2001].

De las asignaturas que se incluyen en estos Planes de Estudios hay varias que podrían considerarse como relacionadas con la materia de Ingeniería del Software, y así se recoge en el Plan de Calidad de la Unidad Docente de Ingeniería del Software y Orientación a Objetos [García et al., 2000]. Pero de una manera directa hay tres asignaturas que se ajustan perfectamente al perfil de la plaza a concurso: **Ingeniería del Software** que se imparte en el primer cuatrimestre del tercer curso de la Ingeniería Técnica en Informática de Sistemas, **Análisis de Sistemas** y **Administración de Proyectos Informáticos**, ambas pertenecientes a la Ingeniería Informática, que con un carácter anual se imparten en el primer y segundo curso respectivamente.

En el caso de la Ingeniería Técnica, el Plan de Estudios vigente (Plan de 1997) comenzó su andadura en el curso 1997/1998 [GAFC-USAL, 1997], impartiéndose la asignatura de Ingeniería del Software por primera vez en el curso académico 1999/2000 [GAFC-USAL, 1999]. No obstante, existe una asignatura equivalente, con las mismas características de nombre, obligatoriedad y carga docente en el Plan de Estudios anterior (Plan de 1992), como se puede comprobar en [GAFC-USAL, 1998].

En la actualidad la asignatura Ingeniería del Software del Plan de 1992 aparece como asignatura sin docencia para aquellos alumnos que, matriculados en el Plan de 1992, no la hayan superado. Además, ha quedado establecido el mecanismo de convalidación oportuno entre las asignaturas del Plan de 1992 y sus homónimas del Plan de 1997.

4.3 Ingeniería del Software

Previamente al desarrollo de la propuesta concreta de los programas de las asignaturas que cubren la materia de Ingeniería del Software en las Ingenierías en Informática en la Universidad

de Salamanca, y que se desarrolla en el Capítulo 5, se presenta una visión global de la Ingeniería del Software como disciplina mediante un análisis de su marco histórico y conceptual. El objetivo no es realizar una exposición exhaustiva, sino mostrar aquellos conceptos, aportaciones y resultados que se consideran relevantes para el contenido de las asignaturas objeto de la propuesta docente. De esta forma los contenidos se adaptarán a las tendencias actuales consolidadas en la teoría, tecnología, práctica y aplicación del software a los sistemas basados en computadores.

El desarrollo de un Proyecto Docente para una determinada disciplina implica, en primer lugar, la identificación clara y precisa del conjunto de conceptos y conocimientos científico/técnicos que la misma engloba. En el caso de la Ingeniería del Software es una tarea que presenta algunas dificultades debido al amplio conjunto de materias que abarca y a la constante evolución a la que se ve sometida como consecuencia de los rápidos cambios que sufre la tecnología del software. Una primera aproximación a esta tarea puede venir dada por la definición de Ingeniería del Software.

4.3.1 Definición de Ingeniería del Software

Antes de hacer un repaso por algunas de las muchas definiciones que de Ingeniería del Software se han dado, se va comentar el origen y polémica que el propio término ha suscitado.

La introducción del término Ingeniería del Software se produce en la primera conferencia sobre Ingeniería del Software patrocinada por la OTAN, celebrada en Garmisch (Alemania) en octubre de 1968 [Naur y Randell, 1969], atribuyéndose la paternidad del término a Fritz Bauer [Randell, 1998].

Fue tal la aceptación de esta conferencia que se consiguió un nuevo patrocinio de la OTAN para una segunda conferencia sobre Ingeniería del Software, que tendría lugar un año más tarde en Roma (Italia) [Buxton y Randell, 1970], con unos resultados menos esperanzadores que los producidos en la primera conferencia. De hecho, no se produjo ninguna petición de que continuara la serie de conferencias de la OTAN, lo cual no influyó para que a partir de entonces se utilizara con gran profusión el nuevo término para describir los trabajos realizados, aunque quizás sin un consenso real sobre su significado.

En el contexto educativo, sin duda alguna, lo que más controversia ha levantado es el propio nombre del término, centrándose la discusión en la pregunta *¿es la Ingeniería del Software realmente una Ingeniería?* [Tomayko, 2000; Bryant, 2000; Lewerentz y Rust, 2000].

Los argumentos que se dan para sustentar una respuesta negativa se pueden resumir en dos categorías. La primera reuniría a aquéllos que se ciñen a la definición literal de Ingeniería dada por algunos diccionarios o sociedades profesionales, argumentando que en estas definiciones se hace mención a productos tangibles derivados del uso efectivo de materiales y fuerzas naturales, mientras que el software ni es tangible, ni utiliza materiales y/o fuerzas naturales para su concepción.

La segunda categoría estaría formada por los que arguyen que una disciplina ingenieril evoluciona desde una profesión, y la profesión relacionada con el software no ha evolucionado lo suficiente para ser considerada una Ingeniería.

Por el contrario, son muchos los que están a favor de la utilización y difusión del término *Ingeniería del Software*, tomando como un estándar de facto la utilización reiterada del término en la bibliografía especializada.

Quizás la defensa más fuerte y adecuada de la Ingeniería del Software como Ingeniería venga de la mano de Mary Shaw que justifica que si tradicionalmente se ha definido Ingeniería como "*la creación de soluciones rentables a problemas prácticos mediante la aplicación del conocimiento científico para la construcción de cosas al servicio de la humanidad*" [Shaw, 1990], entonces el desarrollo del software es un problema ingenieril apropiado, porque involucra "*la creación de soluciones rentables económicamente para problemas prácticos*" [Shaw y Tomayko, 1991].

Una vez hechas estas disquisiciones sobre el término y sus controversias, se va a proceder a exponer una muestra de las numerosas definiciones que de Ingeniería del Software se pueden encontrar en la bibliografía:

> "*Ingeniería del software es el establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en máquinas reales*"
>
> **Fritz Bauer**, *First NATO Software Engineering Conference*, Garmisch (Germany), 1968; en [Buxton et al., 1976][21]

[21] También en [Bauer, 1972].

"*Ingeniería del Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software*"

[Boehm, 1976]

"*Ingeniería del Software es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas software*"

[Zelkovitz et al., 1979]

"*La aproximación sistemática al desarrollo, operación, mantenimiento y retirada del software*"

IEEE "*Standard Glossary of Software Engineering Terminology*"

[IEEE, 1983]

"*Es la disciplina tecnológica y de gestión que concierne a la producción y mantenimiento sistemático de productos software que son desarrollados y modificados a tiempo y dentro de los costes estimados*"

[Fairley, 1985]

"*Tratamiento sistemático de todas las fases del ciclo de vida del software. Se refiere a la aplicación de metodologías para el desarrollo del sistema software*"

Asociación Española para la Calidad [AECC, 1986]

"*Construcción de software multi-versión por un equipo de varias personas*"

[Parnas y Weiss, 1987]

"*La aplicación disciplinada de principios, métodos y herramientas de ingeniería, ciencia y matemáticas para la producción económica de software de calidad*"

[Humphrey, 1989]

"*La utilización de metodologías, herramientas y técnicas para resolver los problemas prácticos que surgen en la construcción, desarrollo, soporte y evolución del software*"

Institute for Information Technology, NRC Canada, 1990

"*Una disciplina cuyo objetivo es la producción de software de calidad, que se entrega en plazo, se ajusta al presupuesto y que satisface sus requisitos*"

Vanderbilt University, "Software Engineering"
Aksen Assoc., 1990

"*(1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento del software; es decir, la aplicación de la Ingeniería al Software.*

(2) El estudio de las aproximaciones en (1)"

IEEE Std 610.12-1990 [IEEE, 1999]

"*Disciplina tecnológica y de gestión concerniente a la invención, producción sistemática y mantenimiento de productos software de alta calidad, desarrollados a tiempo y al mínimo coste*"

[Frakes et al., 1991]

"*Las actividades sistemáticas implicadas en el diseño, implantación y prueba de software para optimizar su producción y soporte*"

Canadian Standards Association.
"*CSA Information Technology Vocabulary*"**, 1992**

"*Aplicación de herramientas, métodos y disciplinas para producir y mantener una solución automatizada de un problema real*"

[Blum, 1992]

"*Aquella forma de ingeniería que aplica principios de informática y matemática a la resolución de problemas software de forma eficiente en cuanto al coste*"

[Humphrey, 1993]

"*Aplicación de principios científicos para la transformación ordenada de un problema en una solución software funcional, así como en el consiguiente mantenimiento del software hasta el final de su vida útil*"

[Davis, 1993]

"*Es la aplicación de herramientas, métodos y disciplinas de forma eficiente en cuanto al coste, para producir y mantener una solución a un problema de procesamiento real automatizado parcial o totalmente por el software*"

[Horan, 1995]

> "*La aplicación de métodos y conocimiento científico para crear soluciones prácticas y rentables para el diseño, construcción, operación y mantenimiento del software y los productos asociados, al servicio de las personas*"
>
> Adaptado de la definición de Ingeniería de
> **Mary Shaw y David Garlan** en **[Shaw y Garlan, 1996]**

En lugar de con definiciones escuetas, la *Sociedad Británica de Informática* y el *Instituto de Ingeniería del Software* presentan su visión de lo qué es la Ingeniería del Software con descripciones más elaboradas. La primera de ellas es la siguiente [BCS, 1989]:

"*La Ingeniería del Software requiere la comprensión y aplicación de principios de ingeniería, habilidades de diseño, buenas prácticas de gestión, fundamentos de la Ciencia de la Computación y formalismos matemáticos. Es tarea de la Ingeniería del Software juntar estas áreas de trabajo tan dispares y utilizarlas en las fases de obtención de los requisitos, especificación, diseño, verificación, implementación, prueba, documentación y mantenimiento de sistemas software complejos y de gran tamaño. El ingeniero del software debe cumplir el papel del arquitecto del sistema complejo, tomando en cuenta las necesidades y requisitos del usuario, la viabilidad, el coste, la calidad, la confianza, la seguridad y las restricciones temporales. La necesidad de ajustar la importancia relativa de estos factores de acuerdo a la naturaleza del sistema y de su aplicación confiere una fuerte dimensión ética a las tareas del ingeniero del software, sobre quien pueda depender la seguridad y bienestar de otros, y para quien, como en medicina o en derecho, un sentido de moralidad profesional se requiere para su trabajo.*

El ingeniero del software debe ser capaz de estimar el coste y la duración del proceso de desarrollo del software, así como determinar la consecución de corrección y confianza. Tales medidas y estimaciones pueden involucrar conocimientos de conceptos financieros y de gestión, al mismo nivel que el manejo de los fundamentos matemáticos. Se necesita el uso preciso de las notaciones formales y de las palabras para expresarlas con el grado de precisión requerido a otros ingenieros y a clientes formados. En la mayoría de las circunstancias las hebras técnicas, teóricas y de gestión de un proyecto de Ingeniería del Software no pueden separarse unas de las otras.

Para construir grandes productos y conseguir una alta productividad, el ingeniero requiere el uso de herramientas software de desarrollo y de elementos reutilizables que garanticen su subsiguiente modificación y mantenimiento con seguridad.

La actividad profesional del ingeniero del software abarca el rango de tareas involucradas en el ciclo de vida de un sistema software. La obtención de requisitos, especificación, diseño, verificación y construcción son tareas críticas para conseguir la calidad del producto y son todas ellas responsabilidad del ingeniero del software.

Dado que el software determina el comportamiento de un autómata, el ingeniero del software necesita tener conocimientos de hardware digital y de comunicaciones. Aunque la Ingeniería del Software como disciplina puede ser calificada como independiente del área de aplicación, su realización debe ser en el contexto de aplicaciones específicas. El ingeniero del software debe, por tanto, ser capaz de colaborar con otros profesionales que le brindarán capacidades complementarias en la labor de especificar, diseñar y construir sistemas hardware-software que se ajusten a las necesidades del cliente, haga uso de las soluciones hardware y software en una óptima combinación y ofrezca una interfaz de usuario con una calidad adecuada.

La mayoría del software se construye en equipo, frecuentemente con equipos interdisciplinarios. La habilidad para trabajar cerca los unos de los otros es esencial.

Algunos de los métodos y de las herramientas intelectuales de la Ingeniería del Software están en proceso de desarrollo y se espera que tengan que cambiar de forma rápida. Los ingenieros del software, por tanto, necesitan tener unos buenos fundamentos teóricos que les sirvan de base par aprender y usar nuevos métodos en el futuro, y la mentalidad que les permita actualizar de forma permanente los conocimientos que necesitan para su labor profesional"

El Instituto de Ingeniería del Software propone la siguiente definición [Ford, 1990]:

1. Definición central:
    - *Ingeniería es la aplicación sistemática de conocimiento científico para la creación y construcción de soluciones rentables a problemas prácticos al servicio de la humanidad.*
    - *La Ingeniería del Software es la forma de ingeniería que aplica principios propios de la Ciencia de la Computación y Matemáticas para conseguir soluciones rentables a problemas software.*
2. Elaboraciones e interpretaciones:
    - *La creación y construcción del software debe incluir el mantenimiento. Debe cubrirse el ciclo de vida del software completo.*

- *La rentabilidad implica no sólo dinero, sino tiempo, calendario y recursos humanos. También implica obtener buenos valores por los recursos invertidos; incluyendo la calidad cuando las medidas se consideren oportunas.*
- *La Ingeniería del Software no se limita a aplicar sólo principios de la Ciencia de la Computación y las Matemáticas, sino cualquier principio del que pueda sacar ventaja.*
- *La Ingeniería del Software necesita contar con principios y técnicas de gestión para llevar a cabo sus actividades de desarrollo.*

3. Distinción entre el uso actual del término "Ingeniería del Software" y la definición que se adecua a la misión del SEI:
   - *Actualmente, el término "Ingeniería del Software" tiene múltiples conjuntos de significados conflictivos y pobremente entendidos, que van desde la programación a la gestión del diseño del sistema.*
   - *Actualmente, el término "Ingeniería del Software" es más una aspiración que una descripción.*

Parece claro que hay un consenso en que el desarrollo del software necesita una base rigurosa que encuentra en la Ingeniería, e indirectamente en la Ciencia y en las Matemáticas. Pero concretamente, en lo referente a la palabra *Ingeniería*, hay diferentes opiniones, así J. McDermid destaca los fundamentos de Ciencia y Matemáticas [McDermid, 1991]; R. S. Pressman, en clara alusión a F. Bauer, habla simplemente de principios de ingeniería, que pueden incorporar principios teóricos y prácticos [Pressman, 2000]; Ian Sommerville incluye "teorías, métodos y herramientas" [Sommerville, 2001], aunque indica que la Ingeniería del Software es diferente a otras formas de Ingeniería debido a la propia naturaleza del software [Beizer, 2000]; Hoare en la década de los setenta cita los componentes clave de la Ingeniería, que según él son lo suficientemente valiosos y relevantes como para ser imitados por los desarrolladores de software, concretamente identifica cuatro aspectos de la Ingeniería que cubren tanto los elementos teóricos como los prácticos de la Ingeniería del Software [Hoare, 1975]: profesionalismo, vigilancia, conocimiento teórico y herramientas; A. Wasserman sugiere que existen ocho nociones fundamentales en la Ingeniería del Software que forman la base para una disciplina efectiva [Wasserman, 1996]: abstracción, métodos y notaciones de análisis y diseño, prototipado de la interfaz de usuario, modularidad y arquitectura del software, proceso y ciclo de vida, reutilización, métricas, y herramientas y entornos integrados.

No sólo hace falta decir lo qué es la Ingeniería del Software, sino que es conveniente recalcar lo qué no es; así la Ingeniería del Software no es el diseño de programas que se implementan en otras áreas ingenieriles, ni es simplemente una forma de programar más

organizada que la que prevalece entre aficionados, principiantes o personas con falta de educación y entrenamiento específico.

Esta variedad de definiciones refleja las diferentes concepciones existentes sobre la Ingeniería del Software. A modo de ejemplo, esta diferencia de alcance y concepción de la Ingeniería del Software como disciplina se aprecia revisando los contenidos del libro de Ian Sommerville [Sommerville, 1985], donde la Ingeniería del Software está limitada al desarrollo de la programación mientras que el libro de Roger S. Pressman [Pressman, 1987] de esa misma época ya considera el análisis y diseño de sistemas. En su quinta edición Sommerville [Sommerville, 1996] amplía su concepto de Ingeniería del Software a aquellas actividades relacionadas con la especificación, desarrollo, gestión y evolución del software, no incorporando ningún capítulo específico sobre la práctica o los lenguajes de programación. En la sexta edición [Sommerville, 2001], última edición hasta la fecha, se ha reorganizado en siete partes, se ha hecho un mayor énfasis en la orientación a objetos, ahora los fragmentos de código se expresan en Java [SUN, 2002] y los modelos objeto en UML (*Unified Modeling Language*) [OMG, 2001b].

El concepto de Ingeniería del Software surge de la distinción entre programación de pequeños proyectos (*programming in the small*) y programación de grandes proyectos (*programming in the large*) y el reconocimiento de que la Ingeniería del Software está relacionada con esta última. Este primer concepto fue rápidamente ampliado para incorporar a la Ingeniería del Software todas aquellas tareas relacionadas con la automatización de los Sistemas de Información y con la Ingeniería de Sistemas en general.

El enfoque de la Ingeniería del Software que se adopta en este Proyecto Docente es el relacionado con los problemas que se presentan en el desarrollo de grandes sistemas software bajo la perspectiva de los sistemas de información a los que dan servicio. Esto viene motivado por el hecho de que aquellos enfoques de la Ingeniería del Software más relacionados con el diseño y la programación de módulos o componentes software están asignados a otras asignaturas de los Planes de Estudios como **Programación**, **Laboratorio de Programación**, **Estructuras de Datos** o **Programación Orientada a Objetos**.

4.3.2 Marco histórico de la Ingeniería del Software

El contexto en el que se han desarrollado las técnicas del software está íntimamente ligado a la evolución solapada de los sistemas informáticos y de la programación, cuyos hitos o avances más importantes han configurado las eras o etapas que se detallan a continuación. No se intenta hacer una clasificación cronológica exacta ya que los desarrollos han estado muy solapados e

incluso ideas que surgieron en una fecha determinada no serian aceptadas ampliamente quizá hasta 10 años después [Goldberg, 1986; Pressman, 1992].

Ninguna de estas eras puede decirse que hayan terminado formalmente, sobre todo si se hace caso del estudio [Redwine, 1985] que afirma que el tiempo necesario para aceptar una idea que implique un cambio tecnológico es de 15 a 20 años. Esta afirmación ha sido corroborada más recientemente por otros muchos autores, como Klaus Dittrich del grupo del conocido "*manifiesto de Atkinson*" [Atkinson et al., 1989].

Durante la **primera era** (1.950-1.96x), la programación de ordenadores se concibe más como un arte que como una técnica sistematizada y compleja. En esta época la importancia se centra en el hardware, sometido a un cambio continuo, considerando al software como un *añadido*. El software se desarrolla utilizando únicamente la intuición, con escasos métodos sistematizados y prácticamente sin ningún control en su desarrollo. La mayoría de los sistemas utilizaban una orientación *batch* (excepciones a esto son el sistema de reserva de *American Airlines* y los sistemas americanos de defensa en tiempo real, como SAGE). El ordenador estaba dedicado a la ejecución de un programa simple que a su vez resolvía una situación específica.

En la **segunda era** (1.96x-197x) la multiprogramación y los sistemas multiusuario introdujeron nuevos conceptos en la interacción hombre-máquina. Los sistemas en tiempo real podían recoger, analizar y transformar datos procedentes de múltiples orígenes, controlando procesos y produciendo resultados en milisegundos en vez de en minutos. Los avances en las máquinas de memoria secundaria *on-line* llevaron a la primera generación de sistemas de gestión de bases de datos.

La segunda era también se caracterizó por el uso de *productos software* o paquetes de software y por el comienzo de las "*software house*" o "*casas de servicios*". El software empezó a ser desarrollado pensando en una distribución más amplia y para un mercado multidisciplinar.

Una de las primeras contribuciones a la Ingeniería del Software, en el sentido de mejorar la fiabilidad, dirección y productividad en el desarrollo del software, fue el artículo de E. W. Dijkstra "*Go to Statement Considered Harmful*", [Dijkstra, 1968a] que acentuaba los beneficios de utilizar los conceptos de la programación estructurada. Otra contribución importante de esta época fue la obra de Weinberg "*The Psychology of Computer Programming*" [Weinberg, 1971]; este clásico de la literatura del software introdujo las ideas de "*programación sin ego*", nombres mnemónicos de variables y en general la necesidad de la claridad y un buen estilo en la programación. Aunque quizás uno de los artículos de mayor influencia de la época fuera el de D. L. Parnas sobre la ocultación de la información [Parnas, 1972].

Según fue creciendo el número de sistemas basados en ordenador las bibliotecas de software se fueron extendiendo. Los proyectos produjeron cientos de miles de instrucciones fuente. Pero los problemas también empezaron a aparecer: todos estos sistemas informáticos y todas estas instrucciones fuente tenían que ser mantenidos para corregir errores "*oscuros*" (detectados tardíamente), adaptarse a los cambios en los requisitos de los usuarios o adaptarse al nuevo hardware que adquirían las organizaciones. El esfuerzo necesario para el mantenimiento de los sistemas informáticos, y sobre todo del software asociado, comenzó a absorber recursos de forma alarmante y, peor aún, la naturaleza personalizada y la ausencia o escasez de técnicas generales de diseño y análisis, hacía que muchos de estos sistemas fuesen prácticamente inmantenibles: había empezado una **crisis del software**, reconocida por primera vez oficialmente en la reunión de la OTAN de 1968 [Naur y Randell, 1969; Boehm, 1976; Goldberg, 1986; Randell, 1998].

En la **tercera era** (1.97x-1.98x) aparecen los sistemas distribuidos - varios computadores, realizando cada uno sus funciones concurrentemente y comunicándose unos con otros - dando lugar a un incremento importante de la complejidad de los sistemas informáticos. Esta época se caracteriza por el uso generalizado del microprocesador que, gracias al abaratamiento y aumento de potencia de éstos, permite la realización de funciones complejas a un coste excepcionalmente bajo.

Durante esta era, la crisis del software se acentuó, detectándose que aproximadamente el 50% de los presupuestos de los centros de procesamiento de datos se dedica a mantenimiento [Goldberg, 1986], de forma que la productividad en el desarrollo de nuevos sistemas se vio notablemente dañada. La intensificación de la crisis provoca como respuesta una mayor toma en consideración de la necesidad de un proceso de Ingeniería para el desarrollo del software. Como consecuencia de la importancia que adquiere la Ingeniería del Software hacen su aparición las primeras metodologías, destacando las propuestas por Jackson [Jackson, 1975], Warnier [Warnier, 1974] y DeMarco [DeMarco, 1979] por ser las que mayor difusión y utilización alcanzan. Por otra parte, este acercamiento del proceso de construcción del software a los procesos de ingeniería clásicos, conduce a la aplicación de técnicas de gestión de proyectos como PERT y CPM a los proyectos de desarrollo de software.

La consecuencia que se extrae de esta época es que es mejor utilizar alguna metodología disciplinada, no importa cual, que no utilizar ninguna [Basili, 1991].

Una **cuarta era** (1.98x-...) ha venido caracterizada por la introducción de sistemas de sobremesa, y la adopción de tecnologías y herramientas que proporcionan el soporte necesario para mejorar la calidad y la productividad en el desarrollo del software. Entre ellas se pueden

destacar: las herramientas CASE, los entornos de programación, el prototipado rápido (usado independientemente o con el ciclo de vida tradicional), la tecnología orientada a objetos, la reutilización sistemática del software, y los lenguajes de cuarta generación (4GL).

Es también en este período cuando empiezan a darse a conocer las aproximaciones formales al desarrollo de software a través de especificaciones algebraicas y lenguajes de especificación ejecutables [Goguen y Meseguer, 1988], como OBJ [Goguen et al., 1992] o ACT ONE [Ehrig y Mahr, 1985], técnicas y métodos orientados a modelos como Z [Spivey, 1989; Diller, 1990] o VDM (*Vienna Definition Method*) [Jones, 1991] y álgebras de procesos como CSP (*Communicating Sequential Processes*) [Hoare, 1985], aunque algunas de estas técnicas son muy anteriores (como VDM, desarrollado por los laboratorios de investigación de IBM de Viena, en 1973).

En los últimos años, han cobrado una gran importancia los modelos de proceso y la preocupación por la mejora en la calidad tanto del producto software como del proceso para mejorarlo. De estos modelos se puede destacar la norma ISO-9000, en su aplicación a la construcción de software [Layman, 1994; Schmauch, 1994], la iniciativa "BOOTSTRAP" [Lebsanft y Synspace, 1994], que consiste en un método para analizar y evaluar cuantitativamente determinados atributos de calidad del proceso (la evaluación permite conseguir un perfil detallado acerca de la calidad de la organización donde se aplica); el conocido modelo de madurez y capacidad de la Universidad Carnegie-Mellon (Pittsburg, PA - USA), CMM (*Capability Maturity Model*) [Paulk et al., 1993a; Paulk et al., 1993b] o el método SPICE (*Software Process Improvement & Capability Evaluation*) [Dorling, 1993; Konrad et al., 1995] que tiene como objetivo convertirse en un estándar ISO mundial que integre a las iniciativas anteriores.

Los sistemas basados en microprocesadores de 32 ó 64 bits, la computación paralela, el desarrollo de sistemas de Inteligencia Artificial (todavía hoy no excesivamente extendidos en el mercado) y las nuevas tecnologías (láser, fibra óptica...) en las comunicaciones (Internet), junto con herramientas de programación visual, desarrollo y ejecución de aplicaciones remotas, están conduciendo a **la transición hacia una quinta era**. En Ingeniería del Software se espera que esta nueva era venga marcada por el desarrollo y perfeccionamiento de los entornos de programación y herramientas integradas de apoyo a metodologías, por la continuación y mejora de las técnicas asociadas al prototipado [Davis, 1982; Gomaa, 1983; Davis, 1992] y a la reusabilidad del software [Marqués, 1995; García, 2000], por las técnicas de Ingeniería Web (*Web Engineering*) [Pressman, 2000, c.29; Ginige y Murugesan, 2001; IEEE-CS, 2001a; IEEE-CS, 2001b; WebE, 2001] y por la aplicación de técnicas de "Ingeniería del Conocimiento" al

desarrollo de software, especialmente como apoyo a la Ingeniería de Requisitos en la captura, estructuración y reutilización del conocimiento (requisitos) en dominios de aplicación [Sutcliffe y Maiden, 1998].

Estas eras tienen una correspondencia bastante inmediata en lo que podía denominarse la historia de la educación en Ingeniería del Software. Así, James E. Tomayko [Tomayko, 1998] identifica tres períodos significativos: la época de cursos de aislados y de libre contenido (antes de 1978), la época de los primeros programas para graduados (1978-1988), y por último una época de rápida proliferación de programas influidos por el trabajo realizado en el SEI (*Software Engineering Institute*).

4.3.3 Marco conceptual de la Ingeniería del Software

En este capítulo se ha señalado que el enfoque de la Ingeniería del Software que aquí se presenta es el relacionado con los Sistemas de Información a los que da servicio. Bajo esta perspectiva, un sistema software es una parte de un sistema mayor que lo engloba como componente. La Ingeniería del Software, por lo tanto, será solamente una parte del diseño del sistema en la que los requisitos del software han de ajustarse a los requisitos del resto de los elementos que constituyen este sistema. Por este motivo, el ingeniero del software ha de estar implicado en el desarrollo de los requisitos del sistema completo, comprendiendo el dominio de actividad en su totalidad.

Atendiendo a esta concepción de la Ingeniería del Software, es importante remarcar el papel que desempeña la Teoría General de Sistemas como antecedente conceptual en el que se apoya la teoría sobre los Sistemas de Información a los que la Ingeniería del Software intenta aportar soluciones. En el marco de la Teoría General de Sistemas, el análisis de sistemas tiene como objetivo general la comprensión de los sistemas complejos para abordar su modificación de forma que se mejore el funcionamiento interno para hacerlo más eficiente, para modificar sus metas... Las modificaciones pueden consistir en el desarrollo de un subsistema nuevo, en la agregación de nuevos componentes, en la incorporación de nuevas transformaciones... En general, el análisis de sistemas establece los siguientes pasos a seguir:

1. **Definición del problema**. En este paso se identifican los elementos de insatisfacción, los posibles cambios en las entradas y/o salidas al sistema y los objetivos del análisis del sistema.

2. **Comprensión y definición del sistema**. En este paso se identifica y descompone el sistema jerárquicamente en sus partes constituyentes o subsistemas junto con las relaciones existentes entre los mismos.
3. **Elaboración de alternativas**. En este paso se estudian las diferentes alternativas existentes para la modificación y mejora del sistema, atendiendo a los costes y perspectivas de realización.
4. **Elección de una de las alternativas definidas en el paso anterior.**
5. **Puesta en práctica de la solución elegida.**
6. **Evaluación del impacto de los cambios introducidos en el sistema.**

Muchas de las técnicas y métodos actuales de la Ingeniería del Software intentan dar respuesta a este tipo de cuestiones.

Los sistemas, de los que el software forma parte, se denominan sistemas informáticos o sistemas de computadora. En este sentido Roger S. Pressman [Pressman, 1997] considera que la Ingeniería del Software ocurre como consecuencia de un proceso denominado Ingeniería de Sistemas de Computadora. La Ingeniería de Sistemas de Computadora se concentra en el análisis, diseño y organización de los elementos en un sistema que pueden ser un producto, un servicio o una tecnología para la transformación de información o el control de información. De igual forma, este autor denomina al *Proceso de Ingeniería del Software* como **Ingeniería de la Información** cuando el contexto de trabajo de Ingeniería se enfoca a una empresa, y lo denomina **Ingeniería de Producto** cuando el objetivo es construir un producto. El término genérico de Ingeniería de Sistemas es el que utiliza para unificar estos dos tipos de ingenierías. La Ingeniería de Sistemas establece, por lo tanto, el papel que ha de asignarse al software y los enlaces que unen al software con otros elementos de un sistema basado en computadora.

Desde una perspectiva más general, J. L. Le Moigne [Le Moigne, 1973] concibe los sistemas formados por tres subsistemas interrelacionados: *el de decisión, el de información* y *el físico*. El sistema de decisión procede a la regulación y control del sistema físico decidiendo su comportamiento en función de los objetivos marcados. El sistema físico transforma un flujo físico de entradas en un flujo físico de salidas. En interconexión entre el sistema físico y el sistema de gestión se encuentra el sistema de información. El sistema de información está compuesto por diversos elementos encargados de almacenar y tratar las informaciones relativas al sistema físico a fin de ponerlas a disposición del sistema de gestión. El Sistema Automatizado de Información (SAI) es un subsistema del sistema de información en el que todas las

transformaciones significativas de información son efectuadas por máquinas de tratamiento automático de las informaciones.

Basándose en las ideas anteriores, se considera a la Ingeniería del Software como la disciplina que se ocupa de las actividades relacionadas con los sistemas informáticos o sistemas de información en los que el software desempeña un papel relevante. Estos sistemas de información han de ser fiables, es decir, que su realización se lleve a cabo de forma correcta conforme a unos estándares de calidad y, además, que su desarrollo se realice en el tiempo y coste establecidos. Este último aspecto es crucial, y existe una gran variedad de informes, publicaciones y datos que avalan la gran dependencia que las organizaciones tienen hoy en día de los sistemas software.

El término software de calidad aparece reiteradamente como un fin de la Ingeniería del Software. Una valoración general de la calidad de un sistema software requiere la identificación de atributos comunes que pueden esperarse de un buen producto de ingeniería. Asumiendo que el software proporciona la funcionalidad requerida, hay una serie de atributos que se deberían encontrar. Según I. Sommerville [Sommerville, 2001] existe un amplio rango de potenciales atributos de calidad del software a considerar (seguridad, protección, fiabilidad, flexibilidad, robustez, comprensión, experimentación, adaptabilidad, modularidad, complejidad, portabilidad, usabilidad, reutilización, eficiencia y aprendizaje), no siendo posible, en general, optimizarlos todos para un sistema software, por lo que una parte importante de la planificación de calidad es la selección de los atributos de calidad de mayor importancia, así como el establecimiento de las vías para alcanzarlos. Por su parte Mynatt [Mynatt, 1990] establece que la calidad de un producto software se mide en función de determinados aspectos que, básicamente, son distintos para el promotor, los usuarios y los encargados de mantenimiento. El promotor o cliente del proyecto, como entidad encargada de los costes, exigirá que el producto software aumente la productividad de su organización, tenga un bajo coste, sea eficiente, fiable y flexible. Los usuarios exigirán que el sistema les proporcione la funcionalidad adecuada, pero también fiabilidad, eficiencia, y que sea fácil de aprender, de usar y de recordar. Por último, a los encargados del mantenimiento les interesará que el sistema contenga el menor número de errores cuando se instale por primera vez, que tenga una buena documentación, que sea fiable y que esté bien diseñado. Por su parte, Bertrand Meyer [Meyer, 1997] distingue entre factores de calidad externos (tales como corrección, robustez, extensibilidad, reutilización, compatibilidad, eficiencia, portabilidad, facilidad de uso, funcionalidad y oportunidad) e internos (modularidad, legibilidad...), si bien los factores externos son los que importan en última instancia, debido a que son los únicos percibidos por el usuario, son los factores internos los que aseguran el cumplimiento de los primeros. En cualquier caso, la optimización de todos estos aspectos es

difícil ya que algunos son excluyentes. Además, la relación entre los costes y estos atributos no es lineal, y pequeñas mejoras o variaciones en cualquiera de ellos pueden ser demasiado caras. Boehm [Boehm, 1981] afirma que aplicar con éxito la Ingeniería del Software requiere una continua resolución de una variedad de metas importantes, pero conflictivas en la mayoría de sus casos.

En muchas ocasiones la Ingeniería del Software se ha querido limitar al desarrollo de grandes proyectos informáticos. Sin embargo, tal y como afirma Barry B. Boehm "*se hacen planos para una casa tanto si esta es grande como si es pequeña*". La filosofía que subyace en esta frase, es que cualquier desarrollo de software ha de seguir un proceso. Como afirma Roger S. Pressman [Pressman, 1997]: "*El fundamento de la Ingeniería del Software es la capa proceso. El proceso de la Ingeniería del Software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la Ingeniería del Software*". El proceso define un marco de trabajo en el que se establece el control de gestión de los proyectos software y el contexto en el que se aplican los métodos técnicos y se producen los resultados del trabajo.

En el proceso de construcción de sistemas informáticos se pueden distinguir dos fases genéricas, independientemente del paradigma de ingeniería elegido: ***la definición*** y ***el desarrollo***.

Durante la fase de *definición* se identifican los requisitos claves del sistema y del software. Durante la misma se desarrollan un **Análisis de Sistemas**, en el que se define el papel de cada elemento en el sistema automatizado de información, incluyendo el que jugará el software, y un **Análisis de Requisitos** en el que se especifican todos los requisitos de usuario que el sistema tiene que satisfacer. Esta fase está orientada al **QUÉ**: *qué información ha de ser procesada, qué función y rendimiento se desea, qué interfaces han de establecerse, qué ligaduras de diseño existen y qué criterios de validación se necesitan para definir un sistema correcto*. En la fase de definición existe un paso complementario que consiste en la planificación del proyecto software, en el que se asignan los recursos, se estiman los costes y se planifican las tareas y el trabajo.

Por el contrario la fase de *desarrollo* está orientada al **CÓMO**, y el primer paso de esta fase corresponde al **Diseño del Software**. En el diseño del software *se trasladan los requisitos del software a un conjunto de representaciones que describen la estructura de datos, arquitectura del software y procedimientos algorítmicos y que permiten la construcción física de dicho software*. Los otros dos pasos de la fase de desarrollo corresponden a la **Codificación** y a la **Prueba del Software**.

Además de las fases de definición y desarrollo del software, existe una tercera fase dentro de la construcción de los sistemas que corresponde a la fase de **Mantenimiento** de los mismos. Esta fase se enfoca *a los cambios asociados con la corrección de errores, con las adaptaciones requeridas por la evolución del entorno del software y las modificaciones debidas a los cambios de requisitos del usuario para mejorar el sistema*.

La fase de definición es crucial en el desarrollo de un sistema software pues en ella quedan establecidas y determinadas explícitamente las necesidades y limitaciones del usuario y del sistema. La especificación de requisitos ha de realizarse antes de que la construcción del sistema de comienzo, pues de lo contrario podría ocurrir que las necesidades se simplifiquen completamente, las limitaciones se olviden o las interdependencias se pasen por alto durante la fase de desarrollo.

Para la comprensión y validación del sistema en estudio, se elaboran modelos. Estos modelos han de separar las especificaciones conceptuales y lógicas (¿qué ha de cumplir el sistema?), de las especificaciones físicas, (¿cómo lo hará dependiendo de los recursos hardware y software?). Un concepto esencial en el desarrollo de este proceso es el de "*nivel de abstracción*". Aplicando este concepto, el desarrollo de un sistema de información consiste en definir una jerarquía apropiada de niveles de abstracción. Cada nivel produce un modelo del sistema que se describe mediante un lenguaje apropiado. El desarrollo comienza con niveles de abstracción altos, poco detallados, y termina con los de máximo detalle que sirven como base para la construcción directa del sistema ejecutable.

Una última consideración a tener en cuenta en el proceso de construcción de sistemas informáticos, es que la operatividad del producto final depende en gran medida de su conocimiento. Esto se consigue con la elaboración detallada y precisa de la documentación de apoyo: *documentación de sistema, manual de usuario, instrucciones de instalación, guías de entrenamiento, manual de operación...*

El uso de una **metodología** permite el dominio del proceso descrito, definición, desarrollo, implementación y mantenimiento, lo que asegurará el éxito de los proyectos informáticos. En general, una metodología es "*el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*" [DRAE, 1995]. Se puede decir que una metodología es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso particular correspondería a la Ingeniería del Software. A su vez, un método, es un procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla. Es un conjunto de técnicas, herramientas y tareas que, de acuerdo a un enfoque metodológico, se aplican para la resolución de un problema.

Desde el punto de vista específico de la Ingeniería del Software, la metodología describe como se organiza un proyecto, el orden en el que la mayoría de las actividades tienen que realizarse y los enlaces entre ellas, indicando asimismo cómo tienen que realizarse algunas tareas proporcionando las herramientas concretas e intelectuales. En concreto, se puede definir **metodología de Ingeniería del Software** como "*un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas*" [Rumbaugh et al, 1991].

En la actualidad se pueden distinguir seis escuelas principales de pensamiento en relación con las técnicas y métodos de desarrollo de Ingeniería del Software:

1. **Orientadas a procesos**: Si se parte de que la Ingeniería del Software se fundamenta en el modelo básico **entrada/proceso/salida** de un sistema[22]; de forma que los datos se introducen en el sistema y éste responde ante ellos transformándolos para obtener salidas. Estas metodologías se enfocan fundamentalmente en la parte de proceso y, por esto, se describen como un enfoque de desarrollo de software orientado al proceso. Utilizan un enfoque de descomposición descendente para evaluar los procesos del espacio del problema y los flujos de datos con los que están conectados. Este tipo de metodologías se desarrolló a lo largo de los años 70. Los creadores de este tipo de métodos fueron Edward Yourdon y Larry Constantine [Yourdon y Constantine, 1975; Yourdon and Constantine, 1979; Yourdon and Constantine, 1989; Yourdon, 1989]; Tom DeMarco [DeMarco, 1979]; Gane y Sarson [Gane y Sarson, 1977; Gane y Sarson, 1979]. Representantes de éste grupo son las metodologías de análisis y diseño estructurado como **Merise** [Tardieu et al., 1986], **YSM** (*Yourdon Systems Method*) [Yourdon Inc., 1993], **SSADM** (*Structured Systems Analysis and Design Method*) [Ashworth y Goodland, 1990] o **METRICA v.2.1** [MAP, 1995] y en parte **METRICA v3.0** [MAP, 2001a; MAP, 2001b].
2. **Orientadas a datos**: Al contrario que en el caso anterior, estas metodologías se centran más la parte de **entrada/salida** dentro del modelo básico **entrada/proceso/salida.** En estas metodologías las actividades de análisis comienzan evaluando en primer lugar los datos y sus interrelaciones para determinar la arquitectura de datos subyacente. Cuando esta arquitectura está definida, se definen las salidas a producir y los procesos y entradas necesarios para obtenerlas. Ejemplos representativos de este grupo son los métodos **JSP** (*Jackson*

[22] Este modelo básico es utilizado por todas las metodologías estructuradas.

Structured Programming) y **JSD** (*Jackson Structured Design*) [Jackson, 1975; Jackson, 1983; Cameron, 1989], la construcción lógica de programas **LCP** (*Logical Construction Program*) de [Warnier, 1974] y el **DESD** (Desarrollo de Sistemas Estructurados de Datos), también conocido como metodología **Warnier-Orr**, [Orr, 1977; Orr, 1981].

3. **Orientadas a estados y transiciones**: Estas metodologías están dirigidas a la especificación de sistemas en tiempo real y sistemas que tienen que reaccionar continuamente a estímulos internos y externos (eventos o sucesos). Las extensiones de las metodologías de análisis y diseño estructurado de Ward y Mellor [Ward y Mellor, 1985] y de Hatley y Pirbhai [Hatley y Pirbhai, 1987] son dos buenos ejemplos de estas metodologías.

4. **Diseño basado en el conocimiento**: Es una aproximación que se encuentra aún en una fase temprana de desarrollo. Utiliza técnicas y conceptos de Inteligencia Artificial para especificar y generar sistemas de información. El método **KADS** (*Knowledge Acquisition and Development Systems*) [Wielinga et al. 1991] y la metodología **IDEAL** [Gómez et al., 1998] son ejemplos de esta categoría.

5. **Orientadas a objetos**: Estas metodologías se fundamentan en la integración de los dos aspectos de los sistemas de información: datos y procesos. En este paradigma un sistema se concibe como un conjunto de objetos que se comunican entre sí mediante mensajes. El objeto encapsula datos y operaciones. Este enfoque permite un modelado más natural del mundo real y facilita enormemente la reusabilidad. Algunos representantes de este grupo son las metodologías **OOA/D** de Grady Booch [Booch, 1994], **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991; Rumbaugh, 1996], **OOSE** (*Object Oriented Software Engineering*) [Jacobson et al., 1993], **FUSION** propuesta por [Coleman et al., 1994], **MOSES** de [Henderson-Sellers y Edwards, 1994a; Henderson-Sellers y Edwards, 1994b], **UP** (*Unified Process*) [Jacobson et al., 1999] o, en parte, **METRICA v3.0** [MAP, 2001a; MAP, 2001b].

6. **Basadas en métodos formales**: Estas metodologías implican una revolución en los procedimientos de desarrollo, ya que a diferencia de todas las anteriores, estas técnicas se basan en teorías matemáticas que permiten una verdadera aproximación científica y rigurosa al desarrollo de sistemas de información y software asociado. Ejemplo de este tipo de metodologías puede ser **OO-Method** [Pastor et al., 1997], que está basada en el lenguaje de especificación formal OASIS (*Open and Active*

Specification of Information Systems) [Pastor y Ramos, 1995; Letelier et al., 1998; Sánchez et al., 1998].

Existen otras clasificaciones de las metodologías, por ejemplo en [Piattini et al., 1996] éstas se clasifican en función de tres parámetros *el enfoque, el tipo de sistema* y *la formalidad*.

4.3.4 Ingeniería del Software y Orientación a Objetos

La Orientación a Objetos forma parte de la Ingeniería del Software como un paradigma de desarrollo con su propio marco conceptual que involucra terminología, métodos, modelos, procedimientos, técnicas, prácticas y procesos.

Los métodos de la Orientación a Objetos han sido reconocidos en el ámbito de las tecnologías de la información, como la mejor filosofía para abordar la reutilización y la extensibilidad del software.

En una primera aproximación se puede decir que el término *orientado a objetos* significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como un comportamiento. Esto se opone frontalmente a la programación convencional, donde las estructuras de datos y la funcionalidad sólo se relacionan débilmente.

Tradicionalmente se asocia la Orientación a Objetos con la Programación Orientada a Objetos, es decir centrando la atención en los lenguajes de programación. Cierto es que los lenguajes de programación orientados a objetos son útiles para eliminar algunas restricciones propias de los lenguajes de programación tradicionales. Sin embargo, enfatizar la fase de codificación supone un retroceso de la Ingeniería del Software al priorizar los mecanismos de implementación frente al proceso de pensamiento subyacente al cual sirven de base [Rumbaugh et al., 1991]. Así, este paradigma se extiende a todas las fases del ciclo de vida con un modelo común subyacente a todas estas fases: *el modelo objeto*.

Se entiende, entonces, por *desarrollo orientado a objetos* la forma de pensar acerca del software basándose en abstracciones del mundo real; donde la palabra desarrollo hace alusión directa al bloque inicial de fases del ciclo de vida del software: *análisis, diseño* e *implementación.*

Aparecen así los conceptos de **Programación Orientada a Objetos** (POO), **Diseño Orientado a Objetos** (DOO) y **Análisis Orientado a Objetos** (AOO).

> "*La programación orientada a objetos es un método de implementación en el que los programas se organizan como colecciones cooperativas de*

> *objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia*"
>
> **Grady Booch**, [Booch, 1994]

Cabe destacar tres partes importantes en esta definición:

1. La POO utiliza *objetos*, no algoritmos, como sus bloques lógicos de construcción fundamentales.
2. Cada objeto es una *instancia* de alguna *clase*.
3. Las clases están relacionadas con otras clases por medio de relaciones de *herencia*.

Mientras que los métodos de programación ponen el énfasis en el uso correcto y efectivo de mecanismos particulares del lenguaje de programación que se utiliza, los métodos de diseño enfatizan la estructuración correcta y efectiva de un sistema complejo, definiéndose como sigue:

> "*El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para descubrir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña*"
>
> **Grady Booch**, [Booch, 1994]

Se destacan dos aspectos de esta definición:

1. El DOO da lugar a una descomposición orientada a objetos.
2. Utiliza diversas notaciones para expresar diferentes modelos del diseño lógico (*estructura de clases y objetos*) y físico (*arquitectura de módulos y procesos*) de un sistema, además de aspectos estáticos y dinámicos del sistema.

El modelo de objetos ha influido en las fases iniciales del ciclo de vida del desarrollo del software. El análisis orientado a objetos enfatiza la construcción de modelos del mundo real, utilizando una visión del mundo orientada a objetos, pudiéndose definir como:

> "*El análisis orientado a objetos es un método de análisis que examina los requisitos desde la perspectiva de las clases y los objetos que se encuentran en el vocabulario del dominio del problema*"
>
> **Grady Booch**, [Booch, 1994]

Toda la Orientación a Objetos gira en torno a dos conceptos fundamentales: *el objeto* y *la clase*. Ambos son dos conceptos estrechamente relacionados, pero que no deben confundirse

nunca [Holland et al., 1997]. Para terminar esta introducción se van a presentar algunas de las definiciones que, tanto de objeto como de clase, pueden encontrarse en la bibliografía especializada.

Un objeto modela una parte de la realidad. Con el concepto de objeto, se modela la permanencia e identidad de conceptos percibidos. Así un objeto puede definirse como:

> "*Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema*"
>
> **[Smith and Tockey, 1988]**
>
> "*Una colección de operaciones que comparten un estado*"
>
> **Peter Wegner,** [Wegner, 1990]
>
> "*Concepto, abstracción, o cosa con frontera y significado débil, perteneciente al problema que se trata; instancia de una clase*"
>
> **[Rumbaugh et al., 1991]**
>
> "*Un objeto es un encapsulado de un conjunto de operaciones o métodos que pueden ser invocados externamente y de un estado que puede recordar los efectos de los métodos*"
>
> **[Blair et al., 1991]**
>
> "*Entidad conceptual que es identificable, tiene características que comparten un estado interno y tiene unas operaciones que pueden cambiar el estado del sistema local, y que también pueden solicitar operaciones de objetos relacionados*"
>
> **[Champeaux et al., 1993]**
>
> "Un objeto tiene estado, comportamiento e identidad; la estructura y el comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables"
>
> **Grady Booch,** [Booch, 1994]

> "*Un concepto, abstracción o cosa que puede ser individualmente identificada y tiene significado en una aplicación. Un objeto es una instancia de una clase*"
>
> **[Blaha y Premerlani, 1998]**

> "*Una entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones, el comportamiento es representado por sus operaciones, métodos y máquinas de estados. Un objeto es una instancia de una clase*"
>
> **[OMG, 2001b]**

Por su parte las clases sirven como plantillas para la creación de objetos, especificando un comportamiento a todas sus instancias. Se puede definir como:

> "*Una clase es una plantilla desde la que sus objetos pueden ser creados. Contiene la definición de los descriptores de estado y los métodos de los objetos*"
>
> **[Blair et al., 1991]**

> "*Es un conjunto de objetos que comparten una estructura común y un comportamiento común*"
>
> **Grady Booch,** [Booch, 1994]

> "*Descripción abstracta de los datos y del comportamiento de una colección de objetos similares*"
>
> **Timothy Budd**, [Budd, 1991]

> "*Descripción de un grupo de objetos con propiedades similares, comportamientos comunes, interrelaciones comunes y semántica común*"
>
> **[Rumbaugh et al., 1991]**

> "*Una clase es un tipo abstracto de datos equipado con una posible implementación*"
>
> **Bertrand Meyer,** [Meyer, 1997]

4.3.4.1 Marco histórico de la Orientación a Objetos

Los pasos por los que ha pasado la evolución histórica de la Orientación a Objeto han sido similares a los de otros métodos de desarrollo; esto es, en primer lugar el énfasis estuvo centrado en las técnicas de programación, para posteriormente ir centrando la atención en las

fases de desarrollo de mayor nivel de abstracción, diseño y análisis, y actualmente en los procesos de negocio [Pancake, 1995].

Lo que no ha sido tan normal ha sido su evolución temporal, pues tras los primeros pasos dados a finales de la década de los sesenta, sufrió una parada significativa en su evolución hasta la segunda mitad de la década de los ochenta[23], que resurge con tremenda fuerza, especialmente a partir de la primera conferencia OOPSLA en 1986, dando lugar a una gran diversidad, que podría calificarse de caótica en algunas parcelas (como por ejemplo los métodos de análisis y diseño). En los últimos años de la década de los noventa se produjo una cierta madurez en la tecnología de objetos, caminando hacia la unificación y los primeros estándares.

El término objeto surge de forma independiente en varios campos de la informática, casi simultáneamente a principios de los setenta, para referirse a nociones que eran diferentes en su apariencia pero relacionadas entre sí. Todas estas nociones se inventaron para manejar la complejidad de los sistemas software de tal forma que los objetos representaban componentes de un sistema descompuesto modularmente o bien unidades modulares de representación del conocimiento [Yonezawa y Tokoro, 1987].

Levy añade que los siguientes acontecimientos contribuyeron a la evolución de los conceptos orientados a objetos [Levy, 1984]:

- Avances en la arquitectura de los computadores, incluyendo los sistemas de capacidades y el apoyo en hardware para conceptos de sistemas operativos.
- Avances en los lenguajes de programación, como se demostró en Simula [Dahl et al., 1970; Dahl y Hoare, 1972; Birtwistle et al., 1973; SIS, 1987], Smalltalk [Goldberg y Robson, 1983; Goldberg, 1985; Lalonde y Pugh, 1990; Lalonde y Pugh, 1990], CLU [Liskov, 1993] y Ada [Ada95-Web].
- Avances en metodología de la programación, incluyendo la modularización y la ocultación de la información [Parnas, 1972].

A esta lista de contribuciones podrían añadirse las tres siguientes [Booch, 1994]:

- Avances en los modelos de bases datos.
- Investigación en Inteligencia Artificial.
- Avances en Filosofía y Ciencia Cognitiva.

[23] Como dato significativo, entre junio de 1978 y diciembre de 1985, sólo aparecen nueve artículos – de unos cuatrocientos – en ACM SIGPLAN Notices, mencionando la tecnología orientada a objetos de una forma significativa [Sharp et al., 2000].

El concepto de un objeto tuvo sus inicios en el hardware con la aparición de arquitecturas basadas en descriptores y, posteriormente, arquitecturas basadas en capacidades [Ramamoorthy y Sheu, 1988]. Estas arquitecturas representaron una ruptura con las arquitecturas clásicas de Von Neumann, surgiendo como consecuencia de los intentos realizados para eliminar el hueco existente entre las abstracciones de alto nivel de los lenguajes de programación y las abstracciones de bajo nivel de la propia máquina. Según sus inventores estas arquitecturas ofrecen ventajas del tipo: mejor detección de errores, mejora de la eficiencia de ejecución, menos tipos de instrucciones, compilación más sencilla y reducción de los requisitos de almacenamiento. Algunos computadores con arquitectura orientada a objetos fueron el Burroughs 5000, el Plessey 250, el Intel 432, el IBM System/38 o el Rational R10000.

Muy relacionados con las arquitecturas orientadas a objetos están los sistemas operativos orientados a objetos. El trabajo de Dijkstra con el sistema de multiprogramación **THE** fue el primero que introdujo la idea de construir sistemas como máquinas de estados en capas [Dijkstra, 1968b]. Otros sistemas operativos pioneros en la tecnología de objetos fueron **UCLA Secure UNIX** (para el PDP 11/45 y 11/70), **StarOS** y **Medusa** (para Cm* de CMU) o el **iMAX** (para el Intel 432). Sistemas operativos actuales, como **Microsoft Windows NT**, parecen seguir el camino de los objetos.

Sin duda alguna la contribución más importante al modelo de objetos estriba en los denominados lenguajes de programación basados en objetos y orientados a objetos. Las ideas fundamentales de clase y objeto aparecieron por primera vez en el lenguaje Simula 67 [SIS, 1987].

En 1972 David L. Parnas introduce la idea de ocultación de la información [Parnas, 1972], y en la década de los setenta varios investigadores, destacando Liskov y Zilles [Liskov y Zilles, 1977], Guttag [Guttag, 1980] y Mary Shaw [Shaw, 1984], fueron pioneros en el desarrollo de mecanismos de tipos de datos abstractos. Hoare contribuyó a esos desarrollos con su propuesta de una teoría de tipos y subtipos.

Aunque la tecnología de Bases de Datos ha evolucionado un tanto independientemente de la Ingeniería del Software, también ha contribuido al modelo de objetos [Atkinson y Buneman, 1987], especialmente mediante las ideas de la aproximación entidad-relación al modelado de datos [Rumbaugh, 1987; Rumbaugh, 1988]. En el modelo entidad-relación, propuesto inicialmente por Peter Chen [Chen, 1976], el mundo se modela en términos de sus entidades, los atributos de éstas y las relaciones entre esas entidades.

En el campo de la Inteligencia Artificial, los avances en representación del conocimiento han contribuido a una comprensión de las abstracciones orientadas a objetos. En 1975, M.

Minsky propuso por primera vez una teoría de marcos para representar objetos del mundo real tal y como los perciben los sistemas de reconocimiento de imágenes y el lenguaje natural [Barr y Feigenbaum, 1981]. Desde entonces se han utilizado los marcos como fundamento arquitectónico para diversos sistemas inteligentes.

Por último la Filosofía y la Ciencia Cognitiva han contribuido al avance del modelo de objetos. La idea de que el mundo podía verse en términos de objetos o procesos procede de los filósofos griegos, más concretamente de la Teoría de las Ideas desarrollada por Platón, a partir de las enseñanzas de Sócrates, y estudiada y ampliada por Aristóteles (en la Tabla 4.1 se tiene una equivalencia entre los conceptos de la Teoría de las Ideas y la Orientación a Objetos, para una mayor información consultar [Alhir, 1998; Rayside y Campbell, 2000]). Así se puede decir que la Orientación a Objetos se acerca más al enfoque Aristotélico-Tomista[24], frente al enfoque Kantiano de los métodos estructurados. Asimismo, en el siglo XVII se tiene a Descartes defendiendo que los seres humanos aplican de forma natural una visión orientada a objetos del mundo [Stillings et al., 1987]. Ya en el siglo XX, Rand amplía estos conceptos en su filosofía de la epistemología objetivista [Rand, 1979].

TEORÍA DE LAS IDEAS	PARADIGMA OBJETUAL
Método Socrático (la recolección)	Abstracción (como forma de obtener conocimiento)
Ideas, universalidad y sustancias secundarias	Clases (y encapsulación)
Cosas, particulares y sustancias primarias	Objetos (y encapsulación)
Procedimiento de captura y división y el principio de uno sobre muchos	Herencia (y polimorfismo)
El argumento del tercer hombre	Abstracción (como marco conceptual para el modelado que involucra múltiples niveles de abstracción)

Tabla 4.1. Correspondencia entre la Teoría de las Ideas y la Orientación a Objetos

Al igual que sucede en el paradigma estructurado, la tecnología de objetos debe dar cobertura a todo el ciclo de desarrollo de los sistemas software, es decir, al análisis, al diseño y a la implementación; aunque se debe resaltar que estas fases se solapan y presentan unas formas más difuminadas que en el desarrollo estructurado.

Cuando a mediados de la década de los ochenta resurge la Orientación a Objetos, comienzan a surgir multitud de propuestas metodológicas con una orientación objetual, sólo en

[24] En la realidad no existen datos o procesos independientes. Santo Tomás de Aquino sostiene que el número, considerado abstractamente, no existe fuera de la mente humana; "*la verdad consiste en la adecuación del entendimiento con las cosas*" (Suma Teológica. I, c.16, a.3.).

Aristóteles indica que "*no se pueden separar, a los objetos en movimiento, por ejemplo*" puesto que "*hay una multitud de accidentes que son esenciales a las cosas, en tanto que cada uno de ellos reside esencialmente en ellas*" (Metafísica. 7ª edición, Madrid, Espasa-Calpe, 1972, p.284.).

el período comprendido entre 1989 y 1994 el número de lenguajes de modelado orientados a objetos pasa de 10 a más de 50. Precisamente esta diversidad de métodos y notaciones ha sido una de las mayores barreras con que se encontraron las empresas y los departamentos de informática para la adopción de la Orientación a Objetos, conociéndose a este fenómeno como la guerra de los métodos [García y Pardo, 1998].

La cantidad de métodos que surgen en esta primera *hornada* se denominan metodologías o métodos de primera generación entre los que cabe destacar **Synthesis** [Bailin, 1989]**, RDD** (*Responsibility Driven Design*) [Wirfs-Brock et al., 1990], **OMT** (*Object Modeling Technique*) [Rumbaugh et al., 1991], **OOD** (*Object Oriented Design*) [Booch, 1991], **OOSE** (*Object-Oriented Software Engineering*) – **Objectory** [Jacobson et al., 1993], o la metodología de **Shlaer y Mellor** [Shlaer and Mellor, 1992].

Hacia la mitad de década de los noventa aparecen en escena las nuevas versiones de los métodos más consolidados, así como nuevas incorporaciones que surgen como una acumulación de las características más destacadas de los métodos tradicionales junto con algunos añadidos: las metodologías o métodos de segunda generación. Pertenecientes a esta generación se tiene, entre otros, a **OMT-2** [Rumbaugh, 1996], **OOram** [Reenskaug et al., 1996], **OOA/D** (*Object-Oriented Analysis and Design*) [Booch, 1994], **Fusion** [Coleman et al., 1994]**, Moses** (*Methodology for Object-Oriented Software Engineering of Systems*) [Henderson-Sellers y Edwards, 1994a; Henderson-Sellers y Edwards, 1994b]**, Syntropy** [Cook y Daniels, 1994] o **Medea** [Piattini, 1994].

Ante esta situación de diversidad, en la parte final de la década de los noventa se empiezan a escuchar los primeros rumores sobre la necesidad de una unificación y una estandarización. Esto ha dado lugar a una tercera generación de metodologías, de entre las que destacan **UP** (*Unified Process*) [Jacobson et al., 1999], **OPEN** (*Object-Oriented Process, Environment and Notation*) [Graham et al., 1997; Henderson-Sellers et al., 1998; Firesmith et al., 1998] o **Catalysis** [D'Souza and Wills, 1999].

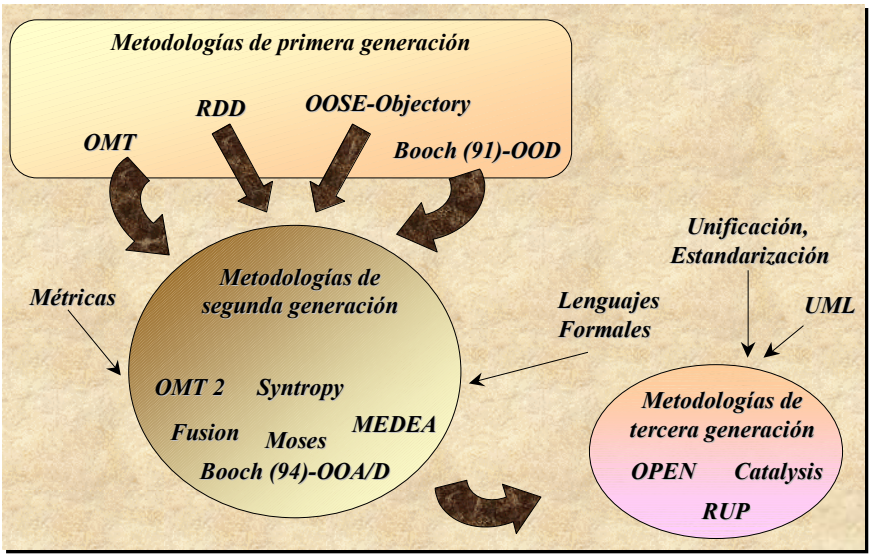

Figura 4.1. Generaciones de las metodologías orientadas a objetos

En la Figura 4.1 se presenta un esquema de la evolución sufrida por las metodologías y métodos de diseño orientados al objeto.

El catalizador principal de esta unificación ha sido sin lugar la aparición en escena del lenguaje de modelado UML (*Unified Modeling Language*) de Rational Software Corporation, que fue adoptado el 17 de noviembre de 1997 como lenguaje de modelado estándar por el OMG (*Object Management Group*) en su versión 1.1 [Rational et al., 1997]. OMG ha propuesto la especificación de UML para su estandarización internacional por parte de ISO (*International Organization for Standardization*).

UML se desarrolla en el seno de Rational Software Corporation con el apoyo de diversos colaboradores a lo largo de su historia, convirtiéndose en el elemento unificador de los lenguajes de modelado de los métodos **OOA/D** de Grady Booch [Booch, 94], **OMT-2** de James Rumbaugh [Rumbaugh, 1996] y **OOSE** de Ivar Jacobson [Jacobson et al., 1993].

Grady Booch, director científico de Rational Software Corporation desde prácticamente su creación en 1980, empieza a planificar su estrategia a favor de la unificación de métodos. Así, para la comunidad de los métodos orientados al objeto la gran noticia en el OOPSLA'94 fue que James Rumbaugh había abandonado General Electric para unirse a Grady Booch en Rational Software Corporation para fusionar sus métodos. De esta manera el desarrollo de UML comienza en octubre de 1994, cuando Booch y Rumbaugh empiezan a trabajar para unificar los dos métodos que más repercusión habían alcanzado en la escena del ADOO, OOA/D y OMT. Como primer fruto de esta colaboración aparece en octubre de 1995 la primera versión pública de la descripción de la unión de sus métodos. Esta versión se presenta en el OOPSLA'95 con el nombre de Método Unificado versión 0.8 [Booch y Rumbaugh, 1995].

El siguiente paso en el proceso de unificación se produce a finales de 1995 cuando Rational compra a la empresa Objectory, y su fundador, el prestigioso Ivar Jacobson, se une a Rational como vicepresidente de Ingeniería de Negocio para acoplar su metodología OOSE al método unificado de Booch y Rumbaugh. Tras la incorporación de Jacobson a Rational, se les conoce a Booch, Rumbaugh y Jacobson por *los tres amigos*.

El primer paso que se da después de la incorporación de Jacobson es la creación de un lenguaje de modelado unificado debido principalmente a dos razones. En primer lugar el lenguaje de modelado permite que los tres métodos puedan evolucionar hacia un punto común de forma independiente. Por otro lado, la unificación de la semántica y de la notación les permite colocarse en un lugar de privilegio en el mercado de los métodos orientados al objeto. Debe tenerse en cuenta que la mayoría de la gente que dice utilizar un método orientado al objeto realmente lo que usa es la notación asociada a dicho método, olvidándose de los procesos asociados al método, así pues, contar con una notación gráfica universal era fundamental para la unificación de sus métodos, y de hecho la notación de UML se convierte en el estándar de facto de las notaciones en tecnología de objetos antes de su aceptación como estándar oficial.

La primera versión de este lenguaje de modelado aparece en junio de 1996 con el nombre de UML 0.9 [Booch et al., 1996a]. En septiembre de este mismo año aparece la versión 0.91 de UML [Booch et al., 1996b].

Durante 1996 invitan a otros expertos a colaborar con ellos, entre los colaboradores se encuentran nombres muy ilustres y de reconocido prestigio dentro de la comunidad de la Ingeniería del Software y más particularmente de la Orientación al Objeto. Citarlos a todos sería largo, pero como muestra se pueden mencionar a personajes de tanto renombre como *Peter Coad, Derek Coleman, Ward Cunningham, David Ambly, Eric Gamma, David Harel, Richard Helm, Ralph Johnson, Stephen Mellor, Bertrand Meyer, Jim Odell, Kenny Rubin, sally Shlaer, JohnVlissides, Paul Ward, Rebbeca Wirfs-Brock, Edward Yourdon* entre otros.

Además, se crea OTUG (*Object Technology Users Group*) como foro de discusión; una lista de correo en la que se discute y se comparten ideas sobre la tecnología de objetos, teniendo como trasfondo a UML.

Tras la salida a la luz de UML la comunidad de ADOO queda divida en dos grupos principales, aquellos que se unen a la estela de UML y aquéllos que forman una "coalición anti UML". Esta situación la aprovecha OMG para en 1996 crear el OOAD SIG (*Object-Oriented Analysis and Design Special Interest Group*) que se encargara de canalizar los esfuerzos para conseguir un estándar. De esta forma se organiza la *OMG Task Force RFP* (*Request For Proposal*), es decir una petición de propuestas para la creación de un estándar del lenguaje de

modelado para los métodos de ADOO. Esta petición de propuestas sugería un lenguaje de modelado que contara con un metamodelo, una notación, una sintaxis y una semántica.

Rational va a responder a esta petición de propuesta con la versión 1.0 de UML en enero de 1997 [Booch et al., 1997a; Booch et al, 1997b]. Esta versión está avalada por un conjunto de empresas de mucho prestigio dentro del mundo de la informática: Digital Equipment Corporation, HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, y Unisys.

Como era de esperar, la respuesta de Rational no fue la única, OMG recibió en enero de 1997 las respuestas de IBM & ObjecTime [Cook et al., 1997], Platinum Technology [Rivas et al., 1997], Ptech, Taskon & Reich Technologies y Softeam.

Durante los primeros meses de 1997 se hicieron predicciones de todo tipo y para todos los gustos. La notación de UML se había convertido en un estándar de facto apoyado por empresas de especial relevancia en el sector informático, lo cual colocaba a UML en una posición de privilegio, pero su oscura semántica y su metamodelo parecían no estar a la altura de otras propuestas, lo cual hacía peligrar su adopción como estándar por OMG en detrimento del resto de las propuestas. Así todo apuntaba a dos posibles soluciones. La primera de ellas, y quizás la peor, daba como resultado dos estándares: UML de Rational y OML de Platinum Technology. La segunda de ella apuntaba por una solución mixta que contemplase aspectos de todas las propuestas, siendo la que se llevó a cabo; así se unen al equipo liderado por Rational el resto de los equipos que enviaron propuestas, dando lugar a la versión 1.1 de UML [Rational et al., 1997], que fue enviada a OMG el 1 de septiembre de 1997, contribuyendo todos con sus ideas a la generación de una respuesta única.

Esta propuesta es aceptada por OMG como estándar de lenguaje de modelado el 17 de noviembre de 1997, pasándose a denominar al lenguaje de modelado OMG UML 1.1. En julio de 1998 aparece la versión OMG UML 1.2 [OMG, 1998], presentando sólo cambios editoriales. Casi un año más tarde, en junio de 1999 aparece OMG UML 1.3 [OMG, 1999] con algunos cambios significativos, especialmente en lo tocante a la semántica. La versión OMG UML 1.4 [OMG, 2001b] aparece como versión definitiva en septiembre de 2001 presentando cambios menores, siendo ésta la versión en vigor a fecha de hoy. El camino seguido hasta la versión actual de OMG UML se puede resumir en la Figura 4.2.

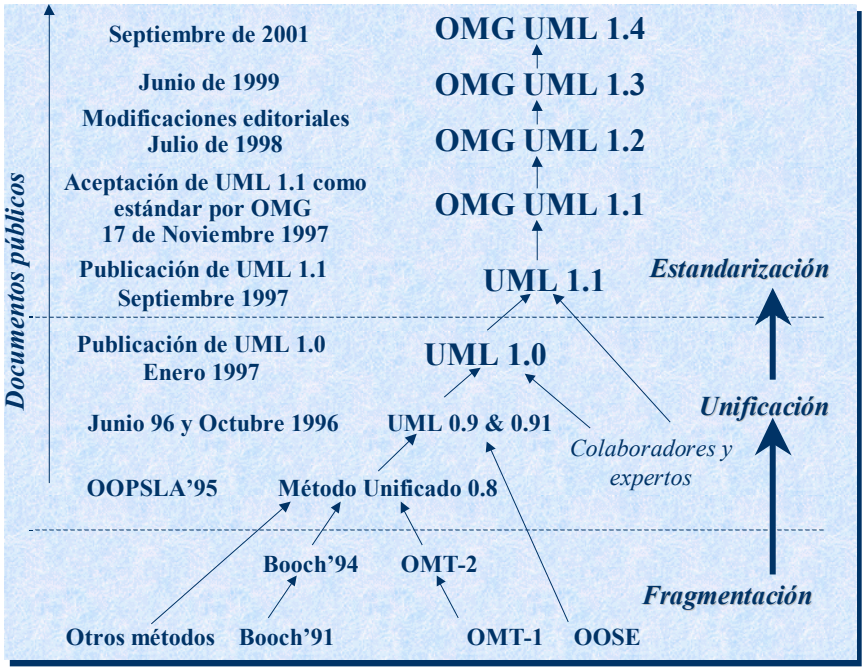

Figura 4.2. Evolución de UML

En los más de cuatro años que han pasado desde que UML fuera aceptado por OMG este lenguaje de modelado se ha convertido en el estándar de facto para especificar artefactos software. Durante este tiempo las modificaciones que ha sufrido UML han sido pequeñas en comparación con la evolución de los métodos que lo han adoptado como lenguaje de modelado. Por este motivo se está preparando una nueva versión que incorporará mayores cambios, y que recibirá el nombre de UML 2.0 [Kobryn, 1999; Kobryn, 2002], siendo el 2002 el año en que se prevé que vea la luz.

Consecuentemente, OMG ha puesto en marcha cuatro RFP para UML 2.0. Una RFP sobre la infraestructura de UML, en la que se busca reestructurar las construcciones básicas de UML y mejorar su configurabilidad [OMG, 2000a]. Una RFP dedicada a la superestructura de UML 2.0 para mejorar elementos avanzados tales como componentes, actividades e interacciones [OMG, 2000b]. Una RFP sobre OCL (*Object Constraint Language*) en pro de incrementar la precisión y expresividad de este lenguaje [OMG, 2000c]. Y por último una RFP para el intercambio de diagramas [OMG, 2001a].

Las referencias básicas de UML las constituyen los libros escritos por los autores principales del mismo [Rumbaugh et al., 1999; Booch et al., 1999; Jacobson et al., 1999], donde los dos primeros están dedicados por completo al lenguaje de modelado, mientras que el tercero se dedica al proceso unificado.

4.3.4.2 Marco conceptual de la Orientación a Objetos

Los sistemas y el pensamiento orientado a objetos se asientan sobre una colección de conceptos que, en su mayor parte, aparecen en diferentes campos y desarrollos de las Ciencias de la Informática con anterioridad a que la Orientación a Objetos fuera un paradigma aceptado como tal.

Desde que los primeros principios de la Orientación a Objetos fueran establecidos en el campo de los lenguajes de programación, fundamentalmente con Simula y Smalltalk, una gran diversidad de campos de la computación han utilizado la tecnología de objetos en el desarrollo de sus teorías; esto ha provocado la existencia de un número de polisemias y sinónimos en relación con el conjunto central de los conceptos inherentes a la Orientación a Objetos. En [Wirfs-Brock y Johnson, 1990] se hace un esfuerzo por presentar los conceptos básicos del paradigma, clarificando su significado con respecto a diferentes interpretaciones.

Para solucionar esta confusión conceptual se establece un *idioma común* que permita la comunicación entre equipos de desarrollo de software con tecnología de objetos. El conjunto de conceptos y propiedades, que configuran este *idioma común* es lo que se conoce como **Modelo Objeto**, y que puede definirse como sigue: "*Un Modelo Objeto es un marco de referencia conceptual, en el que se establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los sistemas software soportados por la tecnología orientada al objeto. Este conjunto básico de conceptos deberá incluir como mínimo, los de abstracción, encapsulación, jerarquía y modularidad; y deberá considerar el sistema de información como un conjunto de entidades conceptuales modeladas como objetos e interactuando entre ellas*" [Marqués, 1995].

Desde la perspectiva de la evolución de los modelos objetos, puede considerarse que el primero de ellos, reconocido como tal, es el propuesto por Anita K. Jones [Jones, 1987] en el año 1978, definiendo el modelo objeto como un concepto y una herramienta, que proporciona las líneas directrices para caracterizar las entidades abstractas, en los términos en los que se conciben.

A partir de esta primera propuesta formal de modelo objeto, y ante la ausencia de un modelo común de referencia y de estándares universalmente aceptados, para estos modelos, se ha asistido a una proliferación de propuestas de modelos objetos [Sernadas et al., 1989; Ward, 1989; Castellanos et al., 1991; Bertino y Martino, 1993; Marqués, 1995] entre otros. Además, se puede constatar que la mayoría de los trabajos en la Orientación a Objetos definen de forma implícita o explícita su propio modelo objeto, como por ejemplo el de UML [OMG, 2001b] o el de OML [Firesmith et al., 1998].

Según Grady Booch [Booch, 1994] todo modelo objeto está formado por cuatro elementos fundamentales: *abstracción, encapsulamiento, modularidad* y *jerarquía*; y por tres elementos secundarios: *tipos, concurrencia* y *persistencia*. "Fundamentales" significa que un modelo que carezca de cualquiera de estos elementos no es orientado a objetos. Con "secundarios" quiere decirse que cada uno de ellos es una parte útil del modelo objeto, pero no esencial.

La *abstracción* se utiliza para combatir la complejidad. Surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias. Una abstracción se centra en la visión externa de un objeto, sirviendo para separar el comportamiento esencial de un objeto de su implantación.

Se puede definir abstracción como: "*Representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes*" [Graham, 1994].

Mientras que la abstracción ayuda a las personas a pensar sobre lo que están haciendo, el encapsulamiento permite que los cambios realizados en los programas sean fiables con el menor esfuerzo. El encapsulamiento genera una cápsula, una barrera conceptual sobre la información y servicios de un objeto, haciendo que estos permanezcan juntos. El encapsulamiento es un medio para conseguir el principio de ocultación de la información [Parnas, 1972]. Se puede definir encapsulamiento como: "*Un principio de estado que agrupa datos y procesos permitiendo ocultar a los usuarios de un objeto los aspectos de implementación, ofreciéndoles una interfaz externa mediante la cual poder interaccionar con el objeto*" [Piattini et al., 1996].

La modularidad permite la fragmentación de un programa en componentes individuales, lo que puede reducir la complejidad en algún grado. Dicha fragmentación crea una serie de fronteras bien definidas y documentadas dentro del programa. Estas interfaces tienen una importancia incalculable para la comprensión del programa. La modularidad se puede definir como: "*La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados*" [Booch, 1994].

La abstracción es un concepto sumamente útil, pero demasiado extenso, excepto para los casos triviales. El encapsulamiento y la modularidad son medios para manejar las abstracciones. Con frecuencia, un conjunto de abstracciones forma una jerarquía. La jerarquía se puede definir como: "*La jerarquía es una clasificación u ordenación de abstracciones*" [Booch, 1994].

Los objetos se comunican a través del *paso de mensajes*. Esto elimina la duplicación de datos, garantizando que no se propaguen los efectos de los cambios en las estructuras de datos encapsuladas dentro del objeto sobre otras partes del sistema. Así, un objeto accede a otro

enviándole un mensaje, cuando esto ocurre, el receptor ejecuta el método correspondiente al mensaje. Un mensaje consiste en un nombre de un método más cualquier argumento adicional. El conjunto de mensajes a los que un objeto responde caracteriza su comportamiento. Se define mensaje como: "*Una comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerado como un evento*" [OMG, 2001b].

El concepto de tipo se deriva de las teorías sobre los tipos abstractos de datos. Se incluye el tipo como elemento separado de modelado de objetos porque el concepto de tipo pone énfasis en el significado de la abstracción en un sentido muy distinto. Se puede definir *tipo* como: "Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas" [Booch, 1994].

Un concepto muy ligado a la teoría de tipos es el *polimorfismo*, que indica que un solo nombre puede denotar objetos de muchas clases diferentes que se relacionan por una superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto de operaciones. Lo opuesto al polimorfismo es el *monomorfismo*, que se encuentra en todos los lenguajes con comprobación estricta de tipos y ligadura estática. Existe el polimorfismo denominado de *inclusión* cuando interactúan las características de la herencia y el enlace dinámico. Es una de las características más potentes de los lenguajes de programación orientados a objetos después de su capacidad para soportar la abstracción, y es lo que distingue a la programación basada en tipos abstractos de datos. Se puede definir polimorfismo como: "*La posibilidad de que una variable o una función adopte diferentes formas en tiempo de ejecución o, más específicamente, a la posibilidad de referirse a instancias de varias clases*" [Graham, 1994].

Un sistema que implique concurrencia puede tener muchos hilos de control (algunos transitorios y otros permanentes durante toda la ejecución del sistema). Mientras que la Programación Orientada a Objetos se centra en la abstracción de datos, encapsulamiento y la herencia, la concurrencia se centra en la abstracción de procesos y la sincronización; los objetos unifican los dos puntos de vista anteriores. Se define la concurrencia como: "*Concurrencia es la propiedad que distingue un objeto activo de uno que no está activo*" [Booch, 1994].

Todo objeto software ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo. Así pues, se puede definir la *persistencia* como: "*La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la*

posición del objeto varía con respecto al espacio de direcciones en el que fue creado)" [Booch, 1994].

4.3.5 El cuerpo de conocimiento de la Ingeniería del Software

Atendiendo a todo lo expuesto, parece imprescindible incluir en la asignatura de Ingeniería del Software los conceptos relacionados con los actores implicados en el desarrollo de proyectos, aspectos del desarrollo y la calidad de los productos finales, así como los procedimientos, herramientas y métodos de trabajo a disposición del analista para poder construir el software de calidad que el usuario necesita. Teniendo en cuenta que entre los enfoques metodológicos mencionados anteriormente la orientación a objetos es una de las líneas más prometedoras y que las orientadas a procesos y las orientadas a estados y transiciones siguen siendo las más utilizadas y difundidas en la actualidad, estos serán los enfoques metodológicos que se incluirán en los programas de las asignaturas presentados en este Proyecto Docente.

Una vez que se ha introducido cual es el marco histórico y conceptual de la materia, se va a perfilar de una manera más concreta el conjunto de conocimientos que entran dentro del área de influencia de la Ingeniería del Software: *su cuerpo de conocimiento*.

Una de las tareas básicas para la definición de una profesión es el establecimiento del conjunto de conocimientos que el profesional debe poseer para el adecuado ejercicio de su labor profesional. Este cuerpo de conocimiento es fundamental para constituir el resto de los elementos que conformarán la profesión, esto es, una propuesta curricular y una política de certificación de los estudios y de los profesionales.

Ante esta necesidad se han propuesto diferentes alternativas, aunque afortunadamente sólo la propuesta de IEEE-CS parece ser la que ha canalizado los resultados, y se ha convertido en la referencia al cuerpo de conocimiento de la Ingeniería del Software.

A continuación se va a describir este cuerpo de conocimiento coordinado por IEEE-CS, aunque se presentarán someramente, a modo informativo, algunas otras propuestas que nacieron casi en paralelo a la de IEEE-CS, pero que quedaron rápidamente olvidadas o absorbidas por ésta.

4.3.5.1 SWEBOK propuesto por IEEE-CS

De las diferentes propuestas para la creación de un cuerpo de conocimiento de la Ingeniería del Software, el proyecto SWEBOK (*Software Engineering Body of Knowledge*) [Abran et al., 2001c], coordinado por IEEE-CS y respaldado por otras importantes compañías y

organizaciones científicas como ACM, Mitre, Boeing o Racional Software Coorporation entre otras, es el que ha acabado acatándose como estándar internacional, aunque a fecha de hoy todavía no ha concluido, habiéndose finalizado en el año 2001 la segunda fase, versión del hombre de piedra [Abran et al., 2001b]. Queda pendiente la tercera y última fase de este proyecto, que se conoce como la versión del hombre de hierro, y que dará comienzo tras un período no inferior a dos años de utilización y maduración de la versión del hombre de piedra, posibilitando así la inclusión de una retroalimentación en el proyecto.

Las previsiones iniciales sobre la duración del proyecto se han visto bastante alteradas, pues en un principio se pensó que el proyecto estaría terminado completamente en el año 2001, tal y como se muestra en la Figura 4.3.

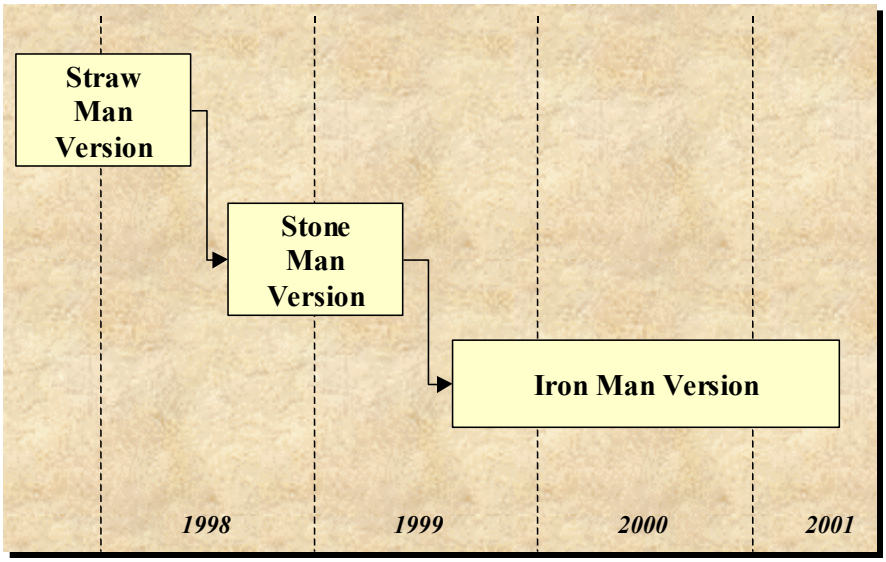

Figura 4.3. Duración inicialmente prevista para el proyecto SWEBOK [Dupuis et al., 1999]

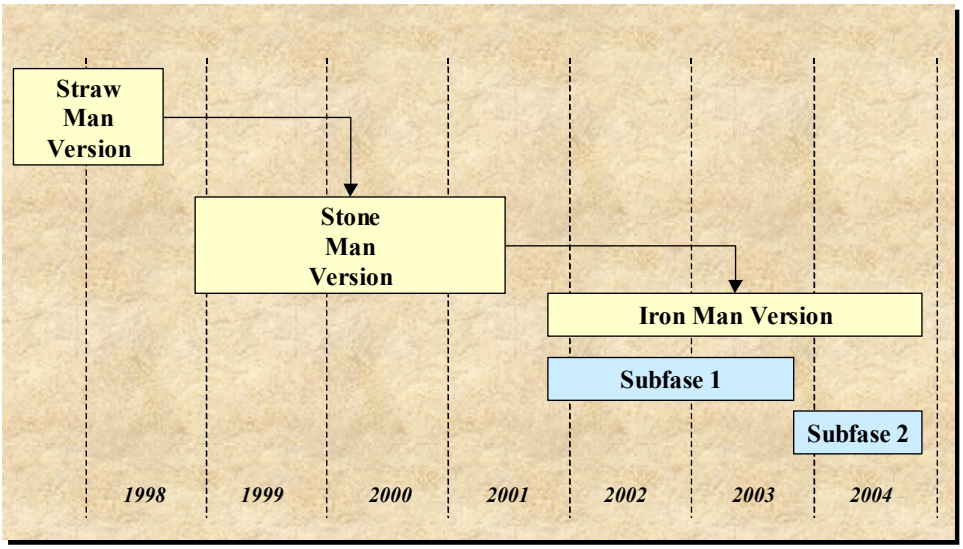

Figura 4.4. Duración prevista del proyecto SWEBOK al finalizar la versión del hombre de piedra (figura basada en [Dupuis et al., 2000])

Sin embargo, la realidad se desarrolló de forma diferente. Así mientras que la versión del hombre de paja [Bourque et al., 1998] concluyó en el tiempo previsto, septiembre de 1998, en la fecha aproximada de conclusión de la versión del hombre de piedra, finales de 1999, sólo se había terminado el primer ciclo de revisión de la misma por un grupo reducido de expertos, dando lugar a la versión 0.5 [Abran et al., 1999]. El segundo ciclo de revisión de la versión del hombre de piedra por un conjunto de usuarios seleccionados concluye en abril de 2000 con la versión 0.7 [Abran et al., 2000]. El tercer ciclo de revisión es llevado a cabo por la comunidad de Ingeniería del Software y concluye en febrero de 2001 con la versión 0.9 [Abran et al., 2001a], que aparece como versión final 0.95 de libre distribución en mayo de 2001 [Abran et al., 2001b] y como libro publicado por IEEE-CS en diciembre de 2001 [Abran et al., 2001c]. La versión del hombre de hierro cuenta de dos subfases, la primera de ellas con una duración estimada no inferior a los dos años monitorizará la utilización de la versión disponible en Internet [Abran et al., 2001b], para que al tercer año se comience el desarrollo propiamente dicho de la versión del hombre de hierro en la segunda subfase de ésta. El proceso seguido hasta la fecha en el desarrollo del SWEBOK se resume en la Figura 4.4.

Los objetivos de este proyecto se resumen en los siguientes cinco puntos [Bourque et al., 1999a; Abran et al., 2001b]:

1. Promover una vista consistente de la Ingeniería del Software para todo el mundo.
2. Clarificar el lugar, y establecer los límites, de la Ingeniería del Software con respecto a otras disciplinas, tales como la Ciencia de la Computación, la Gestión de Proyectos, la Ingeniería de Computadores o las Matemáticas.
3. Caracterizar los contenidos de la Ingeniería del Software como disciplina.
4. Ofrecer un acceso al cuerpo de conocimiento de la Ingeniería del Software.
5. Ofrecer las bases para el desarrollo de una propuesta curricular y una política de certificación, relacionadas ambas con la Ingeniería del Software.

El producto que ha resultado de este proyecto no es el cuerpo de conocimiento en sí, sino una guía de él. El conocimiento ya existe, lo que se busca es el consenso para determinar el subconjunto de conceptos esenciales que caracterizan a la Ingeniería del Software.

La guía que, como ya se ha mencionado, actualmente ha concluido de su segunda fase (versión del hombre de piedra), se divide en diez áreas de conocimiento, cada una de las cuales se estudia en un capítulo independiente de la guía y cuenta con una serie de especialistas responsables. Tanto las áreas como los especialistas para la versión actual quedan reflejados en la Tabla 4.2.

Área de Conocimiento	Especialistas
Requisitos del software	**P. Sawyer y G. Kotonya** (*Lancaster University, Reino Unido*)
Diseño del software	**G. Tremblay** (*Université du Québec à Montreal, Canadá*)
Construcción del software	**T. Bollinger** (*The MITRE Corporation, USA*), **P. Gabrini y L. Martin** (*Université du Québec à Montreal, Canadá*)
Prueba del Software	**A. Bertolino** (*National Research Council, Italia*)
Mantenimiento del software	**T. M. Pigoski** (*Techsoft, Inc., USA*)
Gestión de la configuración del software	**J. A. Scott y D. Nisse** (*Lawrence Livermore Laboratory, USA*)
Gestión de la Ingeniería del Software	**S. G. MacDonell y A. R. Gray** (*University of Otago, Nueva Zelanda*)
Proceso de Ingeniería del Software	**K. El Emam** (*National Research Council, Canadá*)
Herramientas y métodos de la Ingeniería del Software	**D. Carrington** (*The University of Queensland, Australia*)
Calidad del software	**D. Wallace y L. Reeker** (*National Institute of Standards and Technology, USA*)

Tabla 4.2. Las áreas de conocimiento del SWEBOK y sus responsables [Abran et al., 2001b]

El número de áreas de conocimiento no ha sido siempre de diez, ni han tenido siempre el mismo nombre. En la *Straw Man Version* [Bourque et al., 1998] fueron quince las áreas de conocimiento consideradas; éstas se obtuvieron del estudio de libros de texto sobre Ingeniería del Software, del estudio de programas de cursos universitarios y *masters* en Ingeniería del Software, y especialmente el ciclo de vida ISO/IEC 12207[25] [ISO/IEC, 1995] fue considerado como una entrada principal para esta versión de la guía, marcando las bases y el vocabulario para la clasificación de los diferentes temas relacionados con el ciclo de vida.

Además, se han considerado siete disciplinas relacionadas con la Ingeniería del Software: Ciencias cognitivas y factores humanos, Ingeniería de computadores, Ciencia de la Computación, Gestión y Ciencia de la gestión, Matemáticas, Gestión de proyectos e Ingeniería de Sistemas.

El proyecto SWEBOK especifica las unidades de conocimiento, así como los temas pertenecientes a dichas áreas de conocimiento, que se considerará el conocimiento esencial que debe poseer un ingeniero del software. Éstos deben también poseer ciertos conocimientos de las disciplinas relacionadas, pero no es cometido del SWEBOK especificar estos conocimientos.

La guía del SWEBOK se organiza de forma jerárquica, descomponiendo cada área de conocimiento en un conjunto de temas con nombres fácilmente reconocibles, como se puede

[25] Adoptado por IEEE/EIA y por ISO/IEC como un estándar.

apreciar en la Figura 4.5 y en la Figura 4.6. Una breve descripción de cada una de las diez áreas de conocimiento se presenta en el Cuadro 4.2.

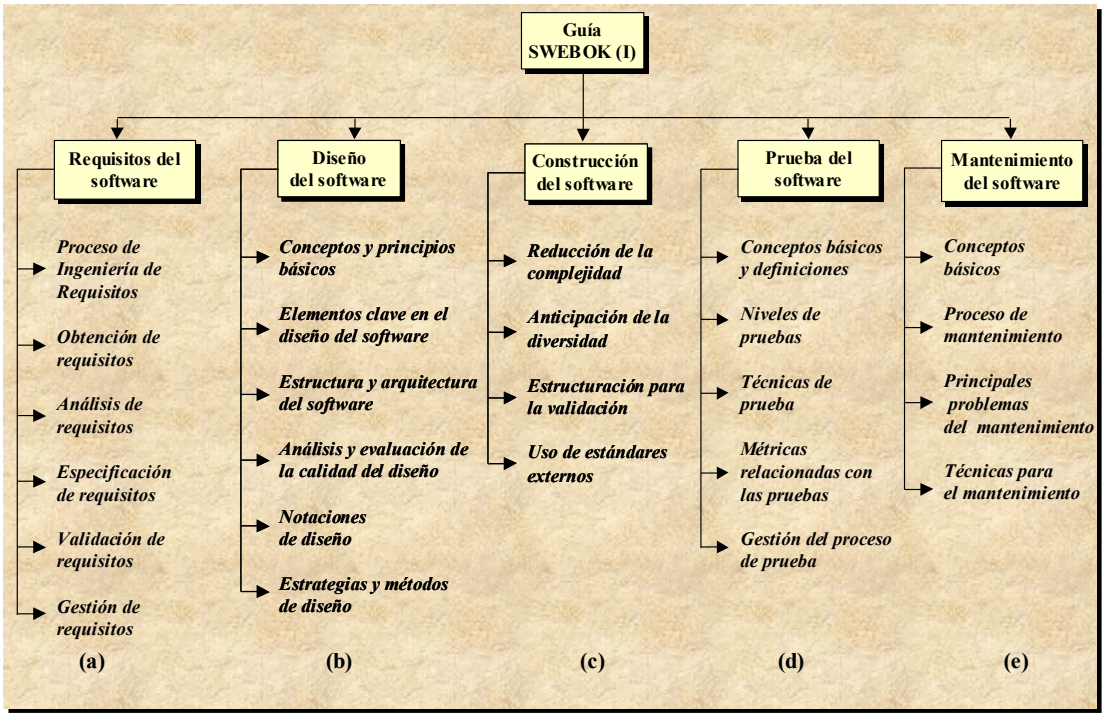

Figura 4.5. Estructura de la guía SWEBOK (parte I) [Abran et al., 2001b]

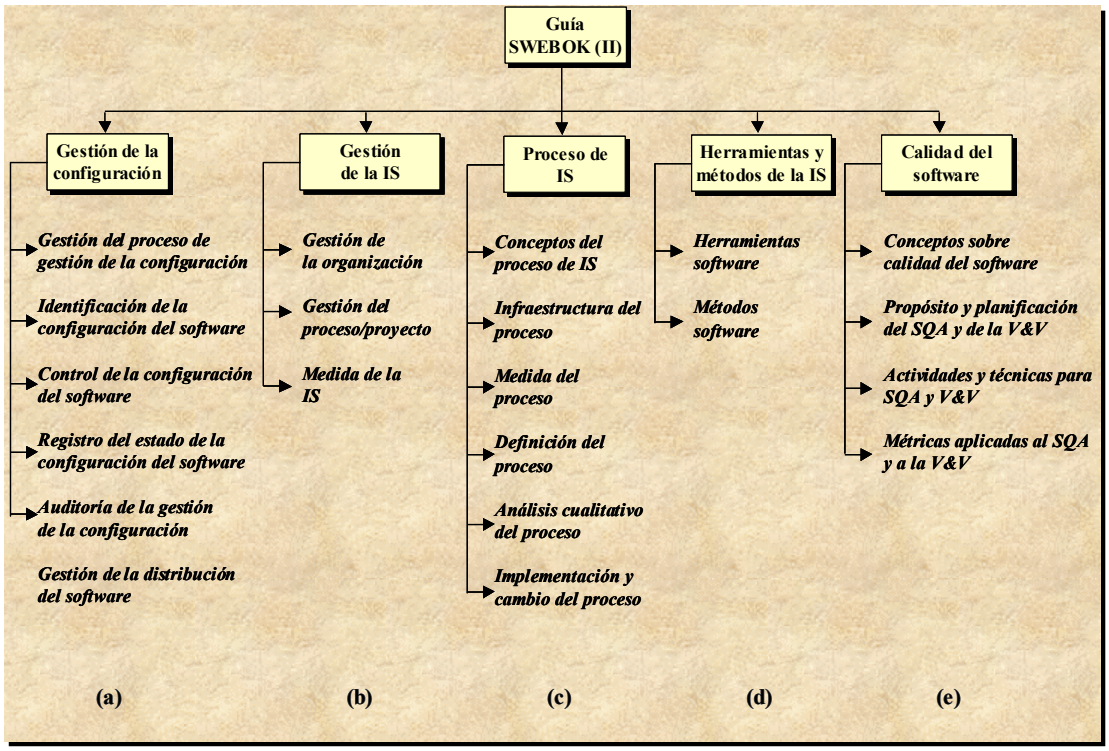

Figura 4.6. Estructura de la guía SWEBOK (parte II) [Abran et al., 2001b]

En la Tabla 4.3 se recoge la equivalencia entre las áreas de conocimiento propuestas en las dos versiones de la guía SWEBOK existentes hasta la fecha.

Requisitos del software: Se divide en seis subáreas que se corresponden aproximadamente con las actividades del proceso que se desarrollan iterativa y concurrentemente, más que de forma secuencial. La parcela del *proceso de ingeniería de requisitos* presenta e introduce las otras cuatro. La subárea de *obtención de requisitos* cubre la captura, descubrimiento o adquisición de éstos. La parte de *análisis* trata de solventar los conflictos entre los requisitos, así como la frontera del sistema. La especificación se refiere a la estructura, calidad y capacidad de verificación del documento de requisitos. La *validación de requisitos* busca conflictos, omisiones y ambigüedades; a la vez que asegura que los requisitos siguen un estándar de calidad. Por último, la subárea de *gestión de requisitos* permite mantener los requisitos presentes en todo el ciclo de vida, adaptándolos a los cambios sufridos por el proyecto.

Diseño del software: Transforma los requisitos, típicamente expresados en términos del dominio del problema, en descripciones que explican cómo resolver el problema. Describe cómo el sistema se descompone y se organiza en componentes, describiendo las interfaces entre ellos.

Construcción del software: Es el acto fundamental de la Ingeniería del Software. Requiere el establecimiento de un diálogo entre el ingeniero y el computador, entre representantes de dos mundos totalmente diferentes. Para establecer los temas de esta unidad de conocimiento se adoptan dos vistas complementarias de la construcción del software. La primera de ellas establece las tres interfaces principales para la creación de software: *lingüista, matemática* y *visual*. La segunda establece que, para cada uno de los estilos, los temas a tratar se organicen de acuerdo a cuatro principios de organización: *reducción de la complejidad, anticipación de la diversidad, estructuración de la validación* y *uso de estándares externos*.

Prueba del software: Consiste en la verificación dinámica del comportamiento de los programas ante un conjunto finito de casos de prueba.

Mantenimiento del software: Estudia los procesos relacionados con la modificación de un producto software después de su entrega, para corregir faltas, mejorar su rendimiento u otros atributos, o adaptar el producto a otro entorno.

Gestión de la configuración del software: Es la disciplina que identifica la configuración de puntos discretos en el tiempo para lograr un control sistemático de sus cambios y mantener la integridad y la trazabilidad a través del ciclo de vida del sistema.

Gestión de la Ingeniería del Software: Une la gestión del proceso y la parte de métricas del proceso. La gestión del proceso casa con la noción de "*gestión a la larga*", esto es, trata de la organización de las fases del ciclo de vida. La parte de métricas aborda *la medida de los objetivos del programa, la medida de la selección, la recolección de datos* y *el desarrollo de modelos*.

Proceso de la Ingeniería del Software: Cubre la definición, implementación, medida, gestión, cambio y mejora de los procesos software.

Herramientas y métodos de la Ingeniería del Software: Describe dos subáreas que discurren de forma horizontal a través de otras áreas de conocimiento: *las herramientas software* y *los métodos de desarrollo*.

Calidad del software: Discute sobre el aseguramiento de la calidad que todo producto resultado de un proceso de Ingeniería del Software debe tener.

Cuadro 4.2. Descripción de las áreas de conocimiento del SWEBOK en la *Stone Man Version*

Área de Conocimiento en la *Stone Man Version*	Áreas de Conocimiento correspondientes en la *Straw Man Version*	Notas
Requisitos del software	Análisis de requisitos♣	
Diseño del software	Diseño detallado♣	Se decidió cubrir todo el diseño desde el principio, no distinguiendo entre diseño arquitectónico y diseño detallado. Esta distinción aparece en el estándar 12207
Construcción del software	Codificación♣	
Prueba del Software	Prueba♣	
Mantenimiento del software	Proceso de mantenimiento♣	
Gestión de la configuración del software	Gestión de la configuración♣	
Gestión de la Ingeniería del Software	Proceso de gestión♣ Medida/Métricas	La definición de este proceso en el estándar 12207 hace mención del uso de datos cuantificables para la toma de decisiones, por eso la parte de *métricas* se ha agrupado con la de *proceso de gestión*
Proceso de Ingeniería del Software	Proceso de mejora♣	
Herramientas y métodos de la Ingeniería del Software	Métodos de desarrollo (orientados al objeto, formales, prototipado) Entornos de desarrollo de software	Los entornos de desarrollo (herramientas) y la reutilización se consideraron los dos elementos principales del proceso de infraestructura. Los métodos se incluyeron porque con frecuencia las herramientas se construyen para implementar métodos determinados
Calidad del software	Aseguramiento de la calidad♣ Verificación y validación♣ Confiabilidad del software	Todas las áreas relacionadas con la calidad se han agrupado en una sola área de conocimiento
	Presentación y definición de la Ingeniería del Software	Se incluyó en la *Straw Man Version* porque aparecía en todos los libros de texto sobre Ingeniería del Software, y se necesita algún tipo de introducción en la *Stone Man Version*, pero no ha sido incluida en ninguna área como tal.

Tabla 4.3. Equivalencia entre la lista de áreas de conocimiento de la *Stone Man Version* y de la *Straw Man Version* de la guía SWEBOK [Bourque et al., 1999b]

En la Tabla 4.4 se establece la correspondencia entre las áreas de conocimiento propuestas en la *Stone Man Version* v0.95 de la guía SWEBOK [Abran et al., 2001b] y el estándar ISO/IEC 12207 [ISO/IEC, 1995].

♣ Área de la *Straw Man Version* basada en el estándar ISO/IEC 12207.

Área de Conocimiento de la *Stone Man Version*	Estándar ISO/IEC 12207	
Requisitos del software	Análisis de requisitos	PROCESOS PRINCIPALES
Diseño del software	Diseño arquitectónico Diseño detallado	
Construcción del software	Codificación Integración	
Prueba del Software	Prueba Instalación Soporte a la aceptación Proceso de operación Explotación del software Soporte operativo a los usuarios	
Mantenimiento del software	Proceso de mantenimiento	
Gestión de la configuración del software	Gestión de la configuración	PROCESOS DE SOPORTE
Calidad del software	Aseguramiento de la calidad Verificación y validación Revisión conjunta Auditoría	
Gestión de la Ingeniería del Software	Procesos de gestión	PROCESOS DE LA ORGANIZACIÓN
Herramientas y métodos de la Ingeniería del Software	Proceso de infraestructura	
Proceso de Ingeniería del Software	Proceso de mejora	

Tabla 4.4. Correspondencia entre las áreas de conocimiento de la guía SWEBOK *Stone Man Version* y el estándar ISO/IEC 12207 [Bourque et al., 1999b]

4.3.5.2 SWE-BOK propuesto para la FAA

Este cuerpo de conocimiento (denotado por las siglas SWE-BOK) es el resultado de un trabajo patrocinado por la FAA (*Federal Aviation Administration*) de EEUU como parte de un proyecto para mejorar los procesos de adquisición, desarrollo y mantenimiento del software de dicha entidad.

Este cuerpo de conocimiento pretende contribuir al trabajo que está realizando el SWECC (*Software Engineering Coordination Committee*) bajo el patrocinio de ACM e IEEE-CS para el desarrollo de la Ingeniería del Software y su madurez como disciplina.

Este cuerpo de conocimiento se recoge en [Hilburn et al., 1999], donde el término *conocimiento* se utiliza para describir el espectro completo del contenido de la disciplina:

información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos y *bibliografía*.

Este cuerpo de conocimiento se estructura en tres niveles de abstracción: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*, para lograr así un balance entre la simplicidad y la claridad y el nivel apropiado de detalle en la descripción del conocimiento. En el Cuadro 4.3 se recogen las definiciones que se manejan en esta estructuración, mientras que en la Figura 4.7 se pueden apreciar estos niveles de abstracción de forma gráfica.

Conocimiento: Término utilizado para describir el espectro completo de los contenidos de la disciplina: *información, terminología, artefactos, datos, roles, métodos, modelos, procedimientos, técnicas, prácticas, procesos* y *bibliografía*.

Cuerpo de conocimiento: Descripción jerárquica del conocimiento sobre la Ingeniería del Software que organiza y estructura el conocimiento en tres niveles de jerarquía: *categorías de conocimiento, áreas de conocimiento* y *unidades de conocimiento*.

Categoría de conocimiento: Una subdisciplina de la Ingeniería del Software que es generalmente reconocida como una parte significativa de este cuerpo de conocimiento de la Ingeniería del Software. Son elementos estructurales de alto nivel, utilizados para organizar, clasificar y describir el conocimiento sobre Ingeniería del Software. Cada una de ellas está compuesta por un conjunto de áreas de conocimiento.

Área de conocimiento: Una subdivisión de una categoría de conocimiento que representa el conocimiento de la Ingeniería del Software que está lógicamente cohesionado y relacionado con la categoría de conocimiento mediante la herencia o la agregación. Cada una de ellas está compuesta de un conjunto de unidades de conocimiento.

Unidad de conocimiento: Una subdivisión de un área de conocimiento que representa un componente básico del cuerpo de conocimiento de la Ingeniería del Software que tiene una descripción explícita. Para el propósito de este cuerpo de conocimiento, cada una de estas unidades es atómica; esto es, no se subdivide en elementos más básicos.

Cuadro 4.3. Definiciones manejadas en el SWE-BOK de la FAA [Hilburn et al., 1999]

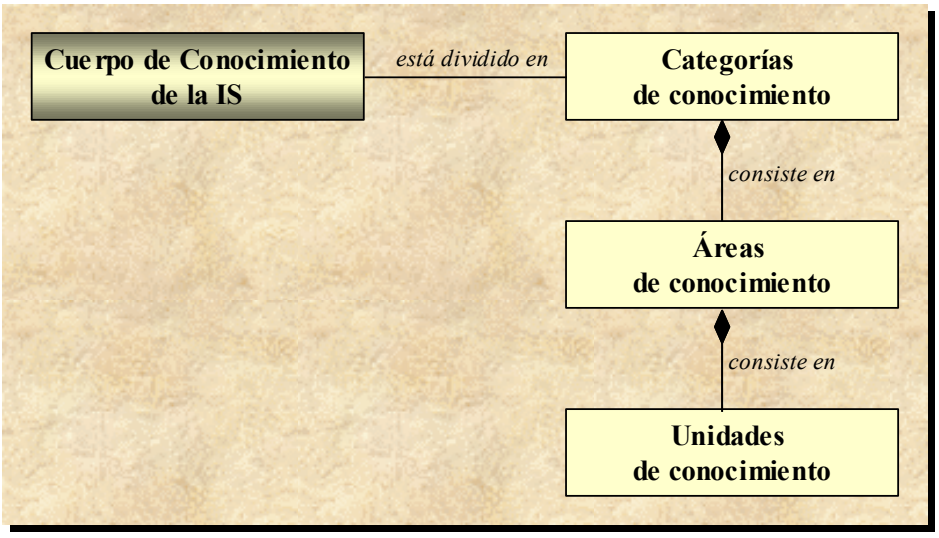

Figura 4.7. Niveles de abstracción en la arquitectura del SWE-BOK de la FAA [Hilburn et al., 1999]

Este cuerpo de conocimiento está formado por cuatro categorías de conocimiento: *Fundamentos de Informática, Ingeniería del Producto Software, Gestión del Software* y *Dominios Software*. En las siguientes tablas se resume este cuerpo de conocimiento.

1. FUNDAMENTOS DE INFORMÁTICA	
Descripción:	Esta categoría concierne al conocimiento, conceptos, teoría, principios, métodos, propiedades y aplicaciones de Informática que forman la base para el desarrollo de software y de la disciplina de la Ingeniería del Software.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
1.1 Algoritmos y estructuras de datos	1.1.1 Estructuras de datos básicas 1.1.2 Diseño de algoritmos 1.1.3 Análisis de algoritmos
1.2 Arquitectura del computador	1.2.1 Sistemas digitales 1.2.2 Organización de un sistema computacional 1.2.3 Arquitecturas alternativas 1.2.4 Comunicaciones y redes
1.3 Fundamentos matemáticos	1.3.1 Lógica matemática y prueba de sistemas 1.3.2 Estructuras matemáticas discretas 1.3.3 Sistemas formales 1.3.4 Combinatoria 1.3.5 Probabilidad y estadística
1.4 Sistemas operativos	1.4.1 Fundamentos de sistemas operativos 1.4.2 Gestión de procesos 1.4.3 Gestión de memoria 1.4.4 Seguridad y protección 1.4.5 Sistemas distribuidos y de tiempo real
1.5 Lenguajes de programación	1.5.1 Teoría de lenguajes de programación 1.5.2 Paradigmas de programación 1.5.3 Diseño e implementación de lenguajes de programación

Tabla 4.5. Categoría de Fundamentos de Informática

2. INGENIERÍA DEL PRODUCTO SOFTWARE	
Descripción:	Esta categoría se refiere a un conjunto de actividades bien definidas e integradas para producir productos software consistentes. Incluye actividades técnicas, que involucran la documentación de estos productos y el mantenimiento de la traza y la consistencia entre ellos. También se refiere al control de la transición entre las diferentes fases del ciclo de vida del software, así como a las actividades que se necesitan para ofrecer productos software de alta calidad a los clientes.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
2.1 Ingeniería de requisitos del software	2.1.1 Obtención de requisitos 2.1.2 Análisis de requisitos 2.1.3 Especificación de requisitos
2.2 Diseño del software	2.2.1 Diseño arquitectónico 2.2.2 Especificación abstracta

	2.2.3 Diseño de la interfaz 2.2.4 Diseño de las estructuras de datos 2.2.5 Diseño de algoritmos
2.3 Codificación del software	2.3.1 Implementación de código 2.3.2 Reutilización de código 2.3.3 Estándares de codificación y documentación
2.4 Prueba del software	2.4.1 Pruebas de unidad 2.4.2 Pruebas de integración 2.4.3 Pruebas del sistema 2.4.4 Pruebas de rendimiento 2.4.5 Pruebas de aceptación 2.4.6 Pruebas de instalación 2.4.7 Documentación de la prueba
2.5 Explotación y mantenimiento	2.5.1 Instalación y explotación del software 2.5.2 Operaciones de mantenimiento del software 2.5.3 Proceso del mantenimiento del software 2.5.4 Gestión del mantenimiento del software 2.5.5 Reingeniería del software

Tabla 4.6. Categoría de Ingeniería del Producto Software

3. GESTIÓN DEL SOFTWARE	
Descripción:	Esta categoría casa con los conceptos, métodos y técnicas para la gestión de los productos y proyectos software. Incluye actividades relacionadas con la gestión de proyectos, gestión de riesgos, calidad del software y gestión de la configuración.
ÁREAS DE CONOCIMIENTO	**UNIDADES DE CONOCIMIENTO**
3.1 Gestión del proyecto software	3.1.1 Planificación del proyecto 3.1.2 Organización del proyecto 3.1.3 Estimación del proyecto 3.1.4 Calendario del proyecto 3.1.5 Control del proyecto
3.2 Gestión de riesgos del software	3.2.1 Análisis del riesgo 3.2.2 Planificación de la gestión del riesgo 3.2.3 Monitorización del riesgo
3.3 Gestión de la calidad del software	3.3.1 Aseguramiento de la calidad del software 3.3.2 Verificación y validación 3.3.3 Métricas del software 3.3.4 Sistemas dependientes
3.4 Gestión de la configuración software	3.4.1 Identificación de la configuración del software 3.4.2 Control de la configuración del software 3.4.3 Auditoría de la configuración del software 3.4.4 Contabilidad del estado de la configuración del software
3.5 Gestión del proceso software	3.5.1 Gestión cuantitativa del proceso del software 3.5.2 Mejora del proceso del software 3.5.3 Evaluación del proceso software 3.5.4 Automatización del proceso software 3.5.5 Ingeniería del proceso software
3.6 Adquisición del software	3.6.1 Gestión de la obtención 3.6.2 Planificación de la adquisición 3.6.3 Gestión del rendimiento

Tabla 4.7. Categoría de Gestión del Software

4. DOMINIOS SOFTWARE	
Descripción:	Esta categoría tiene que ver con el conocimiento de dominios específicos que involucran la utilización y aplicación de la Ingeniería del Software.
ÁREAS DE CONOCIMIENTO	
4.1 Inteligencia artificial	
4.2 Sistemas de bases de datos	
4.3 Interacción hombre-máquina	
4.4 Computación numérica y simbólica	
4.5 Simulación por computadora	
4.6 Sistemas de tiempo real	

Tabla 4.8. Categoría de Dominios Software

4.3.5.3 SE-BOK propuesto por el WGSEET

El fin último del **WGSEET** (*Working Group on Software Engineering Education and Training*) es desarrollar un modelo de currículo para la Ingeniería del Software que pueda ser aplicado en todo, o en parte, en el desarrollo de programas educativos especializados en Ingeniería del Software. Esta propuesta se recoge en [Bagert et al., 1999] (y de forma más esquemática en [Hilburn et al., 1998]).

Como parte importante de este proyecto está la definición de un cuerpo de conocimiento de la Ingeniería del Software que sirva de base a la propuesta curricular. Este cuerpo de conocimiento (denotado por SE-BOK en este trabajo) se organiza en cuatro áreas de conocimiento: *el área central, el área de fundamentos, el área de conceptos recurrentes* y *el área de soporte*, áreas que se detallan en la Tabla 4.9.

SE-BOK [Bagert et al., 1999]	
ÁREAS DE CONOCIMIENTO	**COMPONENTES DE CONOCIMIENTO**
Área Central – incluye aquellos componentes que definen la esencia de la Ingeniería del Software	• Requisitos del software • Diseño del software • Construcción del software • Gestión de proyectos software • Evolución del software
Área de Fundamentos – incluye aquellos componentes que sirve de base a las áreas central y de conceptos recurrentes	• Fundamentos de Informática • Factores humanos • Dominios de aplicación
Área de Conceptos Recurrentes – son elementos que discurren por todos los componentes de conocimiento del área central	• Ética y profesionalismo • Procesos software • Calidad del software • Modelado de software • Métricas del software • Herramientas y entornos • Documentación
Área de Soporte – incluye otros campos de estudio que ofrecen los conocimientos necesarios para completar la educación de los ingenieros del software	• Educación general • Matemáticas • Ciencias naturales • Ciencias sociales • Empresariales • Ingeniería • ...

Tabla 4.9. SE-BOK propuesto por el WGSEET

4.3.5.4 Comparativa

De los diferentes cuerpos de conocimiento que se han presentado, el propuesto por IEEE-CS es el que cuenta con el respaldo internacional y, al igual que el propuesto por el WGSEET, tiene entre sus cometidos servir de base para la definición de una propuesta curricular para la disciplina de la Ingeniería del Software.

El propuesto para la FAA es el que abarca unos contenidos más amplios, buscando definir y evaluar las competencias software que se necesitan en una organización con unas dependencias altas de los sistemas software.

El SWEBOK es la única propuesta que establece la frontera de la Ingeniería del Software identificando otras disciplinas relacionadas, ya que las otras propuestas de una u otra forma introducen en el cuerpo de conocimiento temas relacionados con otras disciplinas. Una de las intersecciones que más se repite es con la parte de Gestión de Proyectos, que por otra parte tiene su propio cuerpo de conocimiento [Duncan, 1996].

4.3.6 La enseñanza de la Ingeniería del Software

Tras la revisión anterior sobre los conceptos relacionados con la disciplina de la Ingeniería del Software, se van a efectuar algunas consideraciones de tipo general sobre la enseñanza de la Ingeniería del Software.

Sin duda, una característica de la Ingeniería del Software como disciplina universitaria, frente a otras disciplinas más establecidas, es el dinamismo con el que cambian sus conceptos y herramientas. Por ello es importante atender a las recomendaciones sobre la enseñanza de la Ingeniería del Software que realizan los organismos y organizaciones internacionales relacionados con la Ciencia de la Computación, los Sistemas de Información y la propia Ingeniería del Software. Atender a este tipo de recomendaciones permite que los conocimientos transmitidos y las habilidades adquiridas por los alumnos respondan a las necesidades profesionales reales y no queden obsoletos en poco tiempo.

Del análisis realizado en el apartado anterior se puede deducir que la Ingeniería del Software consiste en un gran número de actividades interrelacionadas que resultan muy difíciles de conjugar bajo un único epígrafe. Por ello, y sobre todo con fines educativos, es imprescindible introducir una cierta organización en esos contenidos, que permita la definición de un programa educativo consistente y ordenado.

En este sentido en [Ardis y Ford, 1989; Ford y Gibbs, 1989; Ford, 1991a] se identifican para la formación en Ingeniería del Software los puntos de vista del proceso de ingeniería y de

los productos resultantes. A continuación se exponen brevemente las características más importantes de esta doble visión en formación en Ingeniería del Software.

❖ **Punto de vista del proceso**

El proceso de la Ingeniería del Software incluye un amplio rango de actividades realizadas por los ingenieros de software; pero a lo largo de este rango muchos aspectos de estas actividades son similares. Los elementos del proceso pueden considerarse en dos dimensiones: *la actividad* y *el aspecto*.

- *Actividad.* Las actividades del proceso se dividen en cuatro grupos: *desarrollo, control, dirección* y *operaciones*.
    - ***Actividades de Desarrollo.*** Creación o producción del software de los componentes del sistema, incluyendo análisis de requisitos, especificación, diseño, implementación y prueba.
    - ***Actividades de Control***. Estas actividades están más relacionadas con el control del desarrollo que con la producción del software. Las dos actividades principales del control hacen referencia a la evolución del software y a la calidad del software. En este apartado se consideran las actividades relacionadas con la dirección de la configuración, el control de los cambios, el mantenimiento, el control de la calidad, las pruebas, la evaluación, la verificación y la validación.
    - ***Actividades de Dirección***. Implica la dirección ejecutiva, administrativa y supervisora de un proyecto de software, incluyendo las actividades técnicas que soportan el proceso ejecutivo de decisión. Las actividades que se pueden considerar aquí son las de planificación del proyecto, localización de recursos, organización de equipos de trabajo, estimación de costes y aspectos legales.
    - ***Actividades de Operación***. Relacionado con el uso de los sistemas de software. Estas actividades incluyen formación del personal en el uso del sistema, planificación para la entrega e instalación de sistemas, cambio desde el sistema antiguo (manual o automático) al nuevo, operación del software y retirada del sistema.
- *Aspecto*. Las actividades de la Ingeniería se dividen tradicionalmente en actividades analíticas y sintéticas. Se consideran seis aspectos de estas actividades para recoger

esta distinción: *abstracción, representación, métodos, herramientas, medición* y *comunicaciones*.

- ***Abstracción.*** Incluye los principios fundamentales y los modelos formales (Modelos del proceso de desarrollo del software, máquinas de estados finitos y redes de Petri, modelo COCOMO...).
- ***Representación***. Incluye notaciones y lenguajes (Ada, Tablas de Decisión, diagramas de flujo, PERT).
- ***Métodos.*** Incluye métodos formales, prácticas actuales y metodologías (OOD, Programación estructurada...).
- ***Herramientas***. Incluye los conjuntos de herramientas software individuales e integradas (e implícitamente los sistemas hardware donde se ejecutan). Pueden mencionarse las de propósito general (correo electrónico y proceso de textos), las herramientas relativas al diseño e implementación (compiladores y editores sensitivos a la sintaxis) y las herramientas de control de proyectos.
- ***Medición***. Los aspectos de medición incluyen análisis de medidas y evaluación de los productos y el proceso software, así como el impacto del software en la organización; en esta categoría hay que incluir las métricas y los estándares. Esta área merece ser tenida en cuenta en la formación ya que los ingenieros de software, al igual que los ingenieros de los campos tradicionales, necesitan conocer qué medir, cómo medirlo y cómo utilizar los resultados para analizar y evaluar cómo progresan los procesos y productos.
- ***Comunicación***. La comunicación es el último aspecto. Todas las actividades de los ingenieros de software implican comunicación tanto oral como escrita, así como producción de documentación. Un ingeniero de software debe tener unas buenas habilidades en las técnicas generales de comunicación y una comprensión de las formas apropiadas de documentación para cada actividad [Levine et al., 1991; Michael, 2000; Pollock, 2001].

❖ **Punto de vista del producto**

A menudo es conveniente discutir las actividades y aspectos en el contexto de una determinada clase de sistema de software; por ejemplo la programación concurrente y la secuencial tienen características diferenciadoras. Así, se añaden dos nuevas dimensiones a la estructura

organizacional del contenido curricular: *las clases de sistemas software* y *los requisitos del sistema*.

- *Clases de sistemas software*. De las distintas clases que pueden ser consideradas, un grupo se define en función de las relaciones del sistema con su entorno, y sus elementos (partes) están descritos por términos tales como procesamiento por lotes, interactivo, reactivo, tiempo real. Otro grupo tiene elementos descritos en términos tales como distribuido, concurrente, o red. Otro está definido en función de las características internas, tales como orientado a tablas, orientado a procesos o basado en conocimientos. También, se incluyen áreas de aplicación genéricas o específicas, sistemas de aviónica, sistemas de comunicaciones, sistemas operativos, sistemas de base de datos.

- *Requisitos del sistema*. La discusión de los requisitos del sistema generalmente se centra en los requisitos funcionales, pero existen otras categorías que merecen atención. Identificar y reunir esos requisitos es el resultado de las actividades realizadas a lo largo del proceso de Ingeniería del Software. Ejemplos de estos requisitos son: accesibilidad, adaptabilidad, disponibilidad, compatibilidad, exactitud, eficiencia, tolerancia a fallos, integridad, interoperabilidad, mantenibilidad, rendimiento, portabilidad, protección, formalidad, reusabilidad, robustez, seguridad, comprobabilidad y usabilidad.

Otra forma de enfocar la enseñanza de la Ingeniería del Software es mediante un enfoque basado en un proyecto para que el alumno se aproxime al trabajo tal cual ocurre (o debiera ocurrir) en la realidad empresarial [Tomayko, 1987; Shaw y Tomayko, 1991].

En Psicología se distinguen dos tipos de conocimientos: *declarativo* y *procedural* [Norman, 1988]. El primero es fácil de transcribir y de enseñar; sin embargo, el segundo es imposible de transcribir y difícil de enseñar, siendo más sencillo de transmitir mediante demostración y de aprender por la práctica. Muchos de los procesos de la Ingeniería del Software dependen del conocimiento procedural. Por este motivo se recomienda una importante parte de experiencia adquirida mediante proyectos [Ardis y Ford, 1989].

En [Shaw y Tomayko, 1991] se presentan diferentes modelos de cursos de Ingeniería del Software que toman el proyecto como eje conductor de los mismos. Los modelos que se enuncian son:

- ***Modelo de Ingeniería del Software como producto***. Es el modelo al que se ajustan los cursos compuestos exclusivamente por clases teóricas. Son cursos que

concentran en, aproximadamente, un cuatrimestre los conceptos de la Ingeniería del Software. Su mayor desventaja es la ausencia de parte práctica y, por tanto, de experiencia. Estos cursos se adecuan a lo que se denomina *énfasis por el ciclo de vida*, que se ajusta a la forma de organizar los libros de texto sobre Ingeniería del Software, especialmente siguiendo el ciclo de vida en cascada, como claramente se aprecia en el libro de Richard Fairley [Fairley, 1985] o en ediciones anteriores de libros tan clásicos como el de Roger S. Pressman [Pressman, 1987] o Ian Sommerville [Sommerville, 1989].

- ***Modelo de la aproximación por seminarios.*** Es similar al anterior en el sentido de que la base del curso son las clases teóricas, pero se distingue en que en este modelo de curso, se reserva un tiempo para que los alumnos presenten trabajos que ellos mismos han realizado sobre algún tema concreto, manejando la bibliografía oportuna.

- ***Modelo de proyecto para grupos pequeños.*** Este modelo de curso incluye la realización de un proyecto de pequeñas dimensiones como parte del curso. Es muy seguido porque divide el curso en trabajo de clase y trabajo de proyecto. Los alumnos se dividen en grupos de entre tres y cinco personas, debiendo abordar un proyecto que les sea familiar y puedan terminar en el tiempo asignado al curso. Este curso permite obtener algunas experiencias derivadas de la aplicación de lo explicado en las clases teóricas, pero es deficiente en el sentido de que no les entrena para el trabajo en proyectos grandes.

- ***Modelo de proyecto para grandes equipos.*** Es la mejor opción para aprender las técnicas que se utilizan en los proyectos reales. La idea es realizar un proyecto con toda la clase. Típicamente se elige un proyecto consistente en el desarrollo de un producto software, idealmente destinado a un cliente real. Los alumnos se organizan en un solo equipo de desarrollo, asumiendo cada uno de ellos el rol que le sea asignado, y que coincidirá con los roles que aparecen en los entornos industriales reales. Cada alumno mantendrá el rol durante todo el curso, y aprenderá las actividades de los otros roles a través de la interacción con los otros miembros del equipo. Es inviable de llevar a cabo cuando el número de alumnos es alto y la asistencia no es obligatoria. Además, la relación con las actividades propias de otros roles no es tan intensa como las que uno lleva a cabo. Más opiniones sobre este tema se pueden encontrar en [Robillard y Robillard, 1998; Stevens et al., 2000].

- ***Modelo de proyecto único.*** El curso entero se dedica a la realización de un proyecto. Suele llevarse a cabo cuando la Ingeniería del Software se divide en dos asignaturas independientes, una dedicada por completo a la teoría y la otra a la práctica.

Con independencia del modelo que se siga existen varios tipos de proyectos. En [Shaw y Tomayko, 1991] se hace un repaso de ellos, encontrando:

- ***El proyecto de "juguete".*** La clase se divide en equipos de tres a cinco personas; cada equipo recibe una labor predefinida por el responsable de la asignatura. Puede existir una variante donde cada equipo crea su propia especificación. Las ventajas son que los alumnos aplican las enseñanzas de la Ingeniería del Software trabajando en equipo, aunque no se satisfagan los requisitos por completo o no se llegue a terminar la implementación, y los proyectos sean fáciles de gestionar. Se puede llegar a situaciones en las que alumnos de segundo ciclo actúen como gestores de los proyectos realizados por alumnos de primer ciclo. La mayor desventaja es que se omiten técnicas propias de los proyectos grandes como puede ser la gestión de la configuración. Es una práctica muy extendida.
- ***Proyecto basado en componentes.*** Los alumnos se organizan en grupos que desarrollan componentes software. Se establecen después una serie de proyectos que se llevarán a cabo con los componentes realizados, para ello debe haber una primera etapa de *adquisición de componentes*, una segunda de integración de los mismos y una tercera de *modificación* o *adaptación* a las necesidades del grupo. Es un tipo de proyecto en el que se manejan técnicas propias de proyectos grandes, y los alumnos se acostumbran a diseñar componentes reutilizables. Su mayor desventaja está en su gestión, muy parecida al caso anterior.
- ***Proyecto para un cliente externo.*** Los estudiantes trabajan en componentes que deberán integrarse para la realización de un producto para un cliente externo, que deberá pasar los criterios de aceptación oportunos acordados con el cliente. Se puede llevar a cabo en pequeños equipos o con un gran equipo. Es el que más se acerca a la realidad, debiendo hacer uso de todas las técnicas propias de un proyecto de grandes dimensiones, donde la relación con el cliente será uno de los puntos más sobresalientes. Un ejemplo de este tipo de experiencias se encuentra documentado en [Abi-Raad, 2000].

- ***Proyectos individuales.*** Cada equipo tiene un proyecto diferente. Los equipos pueden tener clientes externos, que con frecuencia son facilitados por el responsable de la asignatura. Puede convertirse en un caos desde la perspectiva de gestión de la globalidad de los proyectos.

En los últimos años son frecuentes las críticas realizadas a la manera de formar a los ingenieros informáticos, entre otras cosas porque los modelos de enseñanza y las propuestas curriculares de los Centros y Universidades no evolucionan al mismo ritmo que lo hace la disciplina, de forma que los nuevos ingenieros son educados de forma similar como se ha hecho durante años [Shaw, 1992; Shaw, 1998]. Además, esta evolución choca frontalmente con la falta de flexibilidad de los programas universitarios que presenta la Universidad Española, por ejemplo, y con el necesario balance con la estabilidad que necesita el profesorado para impartir una materia con garantías y poder asumir, al tiempo, la parte de investigación.

Por este motivo los estudios de Ingeniería del Software como tales se encuentran inmersos en programas más amplios sobre Ciencia de la Computación, como ocurre en España. En el informe FASE sobre los estudios superiores de Ingeniería del Software [Knoke y Bagert, 1998] identifican programas de Ingeniería del Software en 77 instituciones en todo el mundo, de forma que la mayoría de ellas ofrecen *masters* en algún área relacionada con la Ingeniería del Software, y nueve ofrecían doctorados con parte de Ingeniería del Software. Cabe destacar que programas de doctorado específicos en Ingeniería del Software han comenzado a aparecer en la Carnegie Mellon University (http://www.isri.cs.cmu.edu).

Para D. L. Parnas la educación de un ingeniero del software debe tener unas raíces sólidas en la educación de la ingeniería tradicional [Parnas, 1990; Parnas, 1998]. Mientras que Bertrand Meyer expone que las instituciones que enseñan software son las responsables de producir profesionales del mundo del software que construyan y mantengan los sistemas para satisfacción de sus beneficiarios [Meyer, 2001].

En este sentido Meyer propone que el currículo de un ingeniero informático debe incluir cinco elementos complementarios [Meyer, 2001]:

- **Principios**: Conceptos sobre los que se construye un campo de conocimiento.
    - Abstracción; Distinción entre especificación e implementación; Recursión; Ocultación de la información; Reutilización; Combate a la complejidad; Clasificación; Escalabilidad; Diseño para el cambio; Tipado; Contratos; Manejo de excepciones; Pruebas.

- **Prácticas**: Técnicas para la resolución de problemas que han de aplicarse consciente y regularmente.
    - Gestión de proyectos, Gestión de la configuración, Métricas, Ergonomía e Interfaces de usuario, Documentación, Interacción, Análisis de sistemas de alto nivel, Depuración.
- **Aplicaciones**: Áreas tradicionales y específicas de las técnicas software.
    - Algoritmos, Estructuras de datos, Compiladores, Sistemas Operativos, Bases de datos, Inteligencia artificial, computación numérica.
- **Herramientas**: Estado del arte de los productos que facilitan la aplicación de los principios y prácticas.
- **Matemáticas**: Base formal que hace posible la comprensión del resto de los conceptos. No se trata de que los alumnos tengan unos conocimientos sumamente profundos sobre formalismos, sino que tengan la habilidad de aplicar razonamiento matemático en el desarrollo del software [Meyer, 1990]. Además, en la Ingeniería del Software la abstracción es un concepto imprescindible, y si hay una disciplina que entrena a pensar en abstracto ésta es la Matemática [Devlin, 2001].

Retos	Aspiraciones
Identificar los distintos roles que aparecen en el desarrollo del software y ofrecer la educación apropiada a cada uno de ellos	Discriminar entre los distintos roles propios del desarrollo del software
	Hacer una educación del software para no graduados que sea válida a largo plazo
	Ofrecer especialización mediante práctica y educación superior
Instalar una actitud ingenieril en los programas educativos	Integrar un punto de vista ingenieril en los currículos para no graduados en Ingeniería Informática
Mantener los programas educativos preparados para afrontar el cambio en la disciplina	Hacer los currículos flexibles para responder al cambio
	Explotar la tecnología para soportar la educación
	Ofrecer mecanismos efectivos para que los ingenieros del software se mantengan al día en los conocimientos de la disciplina
Establecer credenciales que representen la habilidad con exactitud	Establecer credenciales distintas y apropiadas para los diferentes roles que aparecen en el desarrollo del software, cuando esto sea posible
	Establecer credenciales que reflejen con exactitud la práctica que pueden desarrollar

Tabla 4.10. Retos y aspiraciones de la educación de la Ingeniería del Software [Shaw, 2000]

Mary Shaw cuestiona la manera en que se está educando a los ingenieros del software porque no se distingue entre los diferentes roles que aparecen en el desarrollo del software, esto es, no se diferencia la formación de un ingeniero del software de la de un programador [Shaw, 2000]. En este mismo artículo Mary Shaw identifica los retos y las aspiraciones a conseguir en la educación de la Ingeniería del Software; éstos se recogen en la Tabla 4.10.

Resumiendo, el objetivo central que se debe perseguir con la educación en la disciplina de Ingeniería del Software es transmitir a los estudiantes todo lo que ellos van a necesitar en su futuro profesional como ingenieros del software, combinando los fundamentos científicos necesarios con una cierta dosis de pragmatismo, y todo aplicado con un sentido de la responsabilidad tanto individual como colectivo [Baber, 1998].

4.3.6.1 La enseñanza de la Ingeniería del Software Orientada a Objetos

La Orientación a Objetos es en sí misma una subdisciplina de la Ingeniería del Software, que requiere que sus conceptos básicos queden asentados de forma sólida para su correcta aplicación en el desarrollo de sistemas software, donde se quiere obtener ventaja de todo su potencial [D'Souza, 1996].

Para la enseñanza de la Ingeniería del Software Orientada a Objetos se aconseja seguir un modelo de aprendizaje constructivista, más que un modelo de aprendizaje objetivista [Hadjerrouit, 1999]. Esto es así porque en la aproximación objetivista se ve el proceso de aprendizaje como una *transmisión pasiva de conocimientos*, donde no se necesita ningún conocimiento previo; el resultado de este enfoque es que el alumno acaba sin una adecuada comprensión de la base conceptual del paradigma objetual, con malos hábitos de programación y con serios malentendidos sobre la tecnología de objetos. Por su parte, la aproximación constructivista presenta el aprendizaje como un *proceso activo de construcción*, donde los alumnos construyen su conocimiento sobre la base de su conocimiento previo, siendo necesario probar lo que se conoce y evaluar si entra en conflicto con los conceptos que están adquiriendo [Tobin y Tippins, 1993; Brandt, 1997].

El enfoque constructivo, se asemeja a la estructura de los ciclos de vida del desarrollo de software en la tecnología de objetos, ya que es iterativo e incremental. Esto es así, porque el conocimiento y el entendimiento se va adquiriendo, ampliando y madurando en una serie de etapas [Meyer, 1996].

Otro aspecto a tener en cuenta es el momento en el que se introduce la orientación a objetos en el currículo del alumno. Hay defensores de introducir la orientación a objetos desde el primer momento en la formación de los titulados en Informática [Tewari y Friedman, 1992; Decker y

Hirshfield, 1994; Adams, 1996; Woodman et al., 1996; Hadjerrouit, 1999]; aunque siendo conscientes de que el aprendizaje de los conceptos propios de la tecnología de objetos es difícil para los nuevos alumnos porque requiere una forma de pensar diferente y más profunda en términos de computación [Hadjerrouit, 1999].

Sobre los conocimientos que se requieren para iniciarse en la orientación a objetos, hay quien opina que aquéllos que tienen experiencia en el desarrollo de sistemas software, típicamente en el paradigma basado en procedimientos, aprenden fácilmente la sintaxis de los lenguajes orientados a objetos, pero esto no significa que hayan captado la esencia del diseño orientado a objetos y sean capaces de plasmar dichos diseños con los lenguajes aprendidos [Rosson y Carroll, 1996]; de hecho hay estudios que indican que el conocimiento de lenguajes de tipo procedimental puede llegar a ser una barrera conceptual para introducirse plenamente en la orientación a objetos [Hadjerrouit, 1999].

La forma más extendida para introducir la tecnología de objetos en los currículos de Informática es mediante los cursos de introducción a la programación, donde los lenguajes procedimentales y los métodos de diseño estructurado se verían desplazados por sus homónimos orientados al objeto.

Por supuesto, hay voces que no comporten la idea de que los conceptos del paradigma de la orientación a objetos se introduzcan por la programación, defendiendo que los principios del modelo de objetos para realizar análisis y diseño deben cubrirse con gran detalle antes de llegar a su implementación real en un lenguaje de programación [Northrop, 1992]; o quién no está de acuerdo con que la orientación a objetos se centre sólo en la programación, defendiendo un enfoque más amplio que cubra todo el ciclo vital, con una perspectiva de Ingeniería del Software [Bézivin et al., 1992].

La orientación a objetos se presta a la utilización de los patrones pedagógicos (*reusable pedagogical design patterns*) [Lilly, 1996; Proto-Patterns, 1999; Bergin, 1998a; Bergin, 1998b]. En concreto el proyecto de patrones pedagógicos [Proto-Patterns, 1999] recoge prácticas efectivas de docentes en tecnología de objetos. Estos patrones deben ser fáciles de repetir y de adaptar. Cada patrón debe ser descrito de manera que sea fácilmente instanciable para diferentes *lecciones* y por diferentes *educadores*. Muchos de estos patrones se refieren de forma explícita a temas de tecnologías de objetos, como por ejemplo [Bellin, 1999; Prieto y Victory, 1999; Vaitkevitchius, 1999], aunque otros muchos se pueden aplicar a la docencia de cualquier asignatura, como por ejemplo [Bergin, 1998c; Jalloul, 1999; Manns, 1999]. Los antipatrones son otra herramienta que se puede aplicar en la pedagogía [Dodani, 1999].

4.4 Revisión de los currículos internacionales

Se puede definir currículo como "un plan para educar estudiantes, ofreciéndoles las características y el conocimiento necesarios para vivir y practicar competentemente una profesión. El currículo debe anticiparse al mundo cambiante en que los alumnos graduados vivirán y trabajarán" [Denning, 1992].

En este apartado se analizan las propuestas más importantes en currículos internacionales en Informática, realizadas por instituciones de alto prestigio dentro del mundo informático. Para ello se han identificado tres categorías de programas que se estudiarán por separado: *Ciencia de la Computación (CS), Sistemas de Información (IS) o Gestión de Sistemas de Información (MIS)* e *Ingeniería del Software (SE)*.

Cada una de estas áreas presenta su propio enfoque, pero claramente tienen un núcleo de conocimiento común [Parrish et al., 1998].

La revisión que aquí se hace se centra en destacar la presencia de la materia objeto de este Proyecto Docente en las diferentes propuestas curriculares.

4.4.1 Currículos centrados en la Ciencia de la Computación

La educación en *Ciencia de la Computación e Ingeniería* ha sido un área activa durante toda la historia de la disciplina. En particular desde el establecimiento de los primeros Departamentos de Ciencia de la Computación a mediados de la década de los sesenta [Hopcroft, 1987], se puso una especial atención al reto de educar a los alumnos en un campo tan sumamente cambiante y que evoluciona con tanta rapidez como es la Informática [Tucker et al., 1996].

Las propuestas curriculares más relevantes que aparecen en el campo de la Ciencia de la Computación tienen como protagonistas a las dos asociaciones más prestigiosas en el mundo de la Informática, ACM y IEEE-CS. Estas organizaciones publican por separado diferentes propuestas curriculares entre los años 1968 y 1983, para terminar aunando esfuerzos para definir una propuesta curricular única en el campo de la Ciencia de la Computación e Ingeniería, la propuesta de ACM/IEEE-CS de 1991 (también conocida como *Computing Curricula 1991*) [Tucker et al., 1991a] convirtiéndose en una referencia internacionalmente aceptada para el establecimiento de los Planes de Estudio de las titulaciones de Informática en la Universidad, incluyendo a la Universidad en España.

La propuesta ACM/IEEE-CS de 1991 se ha visto reciente reemplazada (diciembre de 2001) por el denominado *Computing Curricula 2001* [ACM/IEEE-CS, 2001].

En los siguientes subapartados, y siempre desde el prisma de la Ingeniería del Software, se van a exponer las características de ambas propuestas curriculares, la de 1991 porque ha sido uno de los principales referentes a la hora de establecer los Planes de Estudio en vigor en la Universidad Española en general, y en particular en la Universidad de Salamanca; y la de 2001 porque establece las directrices a seguir de aquí en adelante.

No obstante, previamente al estudio de estos currículos se va a hacer un breve repaso por sus antecedentes históricos, para conocer mejor sus orígenes y las motivaciones que se han perseguido desde un principio.

4.4.1.1 Antecedentes de los Computing Curricula 1991 y 2001

Los esfuerzos por definir un modelo de currículo para los programas de Ciencia de la Computación e Ingeniería de Computadores comenzaron en la década de los sesenta, poco después de que se establecieran los primeros Departamentos en estas áreas. Así, en 1968, siguiendo una serie de estudios preliminares [ACM, 1965; COSINE, 1967; SAC, 1967], ACM publica el primer currículo informático, denominado *Curriculum '68* [ACM, 1968]. Esta propuesta propone los estudios básicos y troncales, agrupados en asignaturas, para una titulación de Ciencia de la Computación, correspondiente a un primer ciclo (*undergraduate* o *bachelor* en los países anglosajones y, en particular, en los EEUU); incluye recomendaciones detalladas para los programas académicos, a través de un conjunto de descripciones de cursos y una bibliografía exhaustiva para cada área descrita. Los propios autores del *Curriculum '68* manifiestan que iba destinado a los que pretendían dedicarse a la investigación dentro de la Informática.

En la década de los setenta, la Informática se desarrolla rápidamente, hasta el punto de que las recomendaciones del *Curriculum '68* quedan pronto obsoletas. En esta década, tanto ACM como IEEE-CS establecen comités para desarrollar un currículo revisado en Ciencia de la Computación. En 1977, el comité de IEEE-CS publica un informe sobre programas en Ciencia de la Computación e Ingeniería [EC, 1977]. Este informe fue significativo por el hecho de que presentaba una vista amplia de la disciplina, incorporando más Ingeniería al currículo y tendiendo un puente para salvar la distancia existente entre los programas orientados al software y los orientados al hardware. Respondiendo a las presiones generadas por el rápido desarrollo del campo, IEEE-CS actualiza su propuesta curricular en 1983 [EAB, 1983].

En 1978, ACM revisa su propuesta curricular, surgiendo así el *Curriculum '78* [Austing et al., 1979]. Esta propuesta fue bastante más compresible que su antecesor y tuvo mucho impacto en la educación de la Informática. Entre sus contribuciones el *Curriculum '78* propone un

programa estándar para un conjunto de cursos que forman el núcleo de conocimiento de la Ciencia de la Computación como disciplina. El *Curriculum '*78 se organiza en un conjunto de asignaturas troncales, complementado por un grupo de asignaturas optativas. Se añaden además un grupo de asignaturas de matemáticas, de las cuales las cinco tienen carácter obligatorio y dos optativo. Estas asignaturas se recogen en la Tabla 4.11.

Código	Asignatura	Carácter
CS1	Programación I	Troncal
CS2	Programación II	Troncal
CS3	Introducción a los computadores	Troncal
CS4	Introducción a la estructura de los computadores	Troncal
CS5	Introducción al procesamiento de ficheros	Troncal
CS6	Sistemas operativos y Arquitecturas de computadores I	Troncal
CS7	Estructuras de datos y análisis de los algoritmos	Troncal
CS8	Organización de los lenguajes de programación	Troncal
MA1	Cálculo elemental	Troncal
MA2	Análisis matemático I	Troncal
MA2A	Probabilidades	Troncal
MA3	Álgebra lineal	Troncal
MA4	Estructuras discretas	Troncal
MA5	Análisis matemático II	Optativa
MA6	Probabilidad y estadística	Optativa
CS9	Computadores y sociedad	Optativa
CS10	Sistemas operativos y arquitecturas de computadores II	Optativa
CS11	Diseño de sistemas de gestión de bases de datos	Optativa
CS12	Inteligencia artificial	Optativa
CS13	Algoritmos	Optativa
CS14	Diseño y desarrollo de software	Optativa
CS15	Teoría de los lenguajes de programación	Optativa
CS16	Autómatas, computabilidad y lenguajes	Optativa
CS17	Matemáticas numéricas: Análisis	Optativa
CS18	Matemáticas numéricas: Álgebra lineal	Optativa

Tabla 4.11. Asignaturas en *Curriculum '78*

La principal crítica al *Curriculum '78* se refiere principalmente al papel secundario atribuido a las matemáticas [Ralston y Shaw, 1980; Ralston, 1984; Berztiss, 1987]. De hecho, ninguna asignatura de matemáticas aparece como prerrequisito de ninguna de informática y el componente matemático del currículo no queda bien definido. Otros autores opinan que esta propuesta es un resumen de otros planes ya existentes, no presentando innovaciones [Aiken,

1980; Shaw, 1985], centrándose más en las aplicaciones que en los fundamentos de la informática [Berztiss, 1987].

El *Curriculum '78* con sólo unas pequeñas modificaciones [Koffman et al., 1984; Koffman et al., 1985], constituye un estándar para la educación en Informática, hasta 1991.

En 1981, y tras dos años de trabajo, el *ACM Committee on Curriculum in Computer Science*, presenta una propuesta curricular orientada a nivel de *master* [Magel et al., 1981]. Plantea como objetivo el desarrollo del pensamiento y de la intuición profesional crítica del estudiante, poniendo énfasis en los conceptos, la teoría y la práctica de la Informática. Las asignaturas que se incluyen en esta propuesta curricular se pueden agrupar en cinco áreas: *Lenguajes de programación*, *Sistemas operativos y arquitectura de computadoras*, *Informática teórica*, *Estructuras de datos y ficheros* y *Otras materias*.

En 1984 se presentan unas guías para la acreditación de los profesionales [Mulder y Dalphin, 1984], y en 1986 se presenta un currículo para las escuelas de artes liberales [Gibbs y Tucker, 1986].

Aunque ACM parte de una tradición más proclive a la Ciencia de la Computación, en la última década del siglo XX se ha acercado a la parte de Ingeniería, convergiendo con IEEE-CS, de orientación tradicionalmente ingenieril. De hecho, ACM se está abriendo incluso al mundo de los Sistemas de Información, cosa que nunca ha hecho IEEE.

Dado que en los trabajos publicados por las dos principales sociedades de Informática tenían bastantes partes en común, deciden aunar esfuerzos y elaborar una única propuesta avalada por ambas sociedades. Así, en 1985 se crea un grupo de trabajo dirigido por Peter Denning y formado por miembros de ACM e IEEE-CS; este grupo publica un informe en diciembre de 1988 [Denning et al., 1988]. Se decide continuar con el trabajo para desarrollar las recomendaciones para un currículo completo, para lo que se crea otro grupo de trabajo conjunto en febrero de 1988 (*The Joint ACM/IEEE-CS Curriculum Task Force*), que da lugar al *Computing Curricula 1991* [Tucker et al., 1991a]. Este currículo es más comprensible que sus predecesores, aunque con un enfoque diferente. Mientras que el *Curriculum '78* y el informe de IEEE-CS de 1983 se centraban en la identificación de programas estándares para cursos individuales, el *Computing Curricula 1991* divide el cuerpo de conocimiento asociado a la Ciencia de la Computación en unidades de conocimiento individuales. Cada una de estas unidades se corresponde con un tema que debe ser cubierto en algún momento en el currículo, aunque las organizaciones tienen la flexibilidad de componer las unidades en cursos como ellas requieran para satisfacer sus necesidades particulares.

A finales de 1998, la *ACM Education Board* y la *Educational Activities Board* de IEEE-CS establecieron un nuevo grupo de trabajo para revisar el *Computing Curricula 1991* [Roberts et al., 1999; Chang et al., 1999], dando lugar de forma definitiva el 15 de diciembre de 2001 al *Computing Curricula 2001* [ACM/IEEE-CS, 2001]. Esta última propuesta intenta reflejar el tremendo avance sufrido por la disciplina en los últimos diez años, llegando a la conclusión que debe haber diferentes propuestas curriculares para la Ciencia de la Computación, la Ingeniería de los Computadores, la Ingeniería del Software y los Sistemas de Información, de forma que el *Computing Curricula 2001* debiera ser la unión de los cuatro. De momento la propuesta recogida en [ACM/IEEE-CS, 2001] se refiere sólo a Ciencia de la Computación.

4.4.1.2 La propuesta conjunta ACM/IEEE-CS (Curricula 91)

La recomendación conjunta (ACM/IEEE-CS 91) [Tucker et al., 1991a; Tucker et al., 1991b; Tucker y Barnes, 1991] incluye un currículo general en Informática, adaptable a las necesidades concretas de cada institución que imparta titulaciones como "*Computer Science*", "*Computer Engineering*", "*Computer Science and Engineering*" y otras similares; pretende ser flexible y adaptarse a los nuevos cambios tecnológicos que se vayan produciendo. El informe constituye un conjunto de guías, más que una prescripción de cursos concretos, en una titulación universitaria de Informática. El contenido de este informe puede resumirse en los siguientes puntos:

- Una identificación de objetivos de un Plan de Estudios universitario en la disciplina de Informática.
- Una definición de qué es la Informática como disciplina vista en forma bidimensional "área/proceso", que comprende la definición de nueve áreas temáticas y tres procesos, uno para cada una de las áreas (Teoría, Abstracción y Diseño).
- Una colección de material curricular avanzado y suplementario que posibilita la profundización en el estudio en algunas de las nueve áreas identificadas. El desarrollo de estos materiales variará en función de los intereses y posibilidades de cada institución que los imparta.
- Un conjunto de consideraciones pedagógicas y curriculares que gobiernen la materialización de los anteriores requisitos comunes y materiales avanzados/suplementarios en una titulación universitaria completa. Se incluyen aspectos como *el papel de los laboratorios, la programación, las matemáticas, conceptos éticos, sociales y profesionales* y hace explícita la noción de los

"**conceptos recurrentes**" comunes a diferentes áreas, independientes de una tecnología en particular y repetidas por toda la disciplina Informática.

A lo largo del documento se insiste en que un currículo es algo más que un conjunto de cursos y por eso tiene que conocerse no solamente la materia básica, sino también tienen que comprenderse los tres procesos o puntos de vista: teoría, abstracción y diseño.

Influencias sobre el *Computing Curricula 91*

Los siguientes puntos resumen los aspectos más significativos de algunos de los trabajos anteriores que han tenido influencia sobre ACM/IEEE-CS91:

- El informe de **ACM de 1968** [ACM, 1968] supone una de las primeras tentativas de definir el ámbito de la Informática como disciplina. Además de incluir un currículo, define los principales campos de la Informática; concretamente identifica tres áreas: *estructuras de información y procesos, sistemas de proceso de información* y ***metodologías***. Propone un currículo central compuesto de cuatro cursos básicos (algoritmos y programación, estructura de computadores y sistemas, estructuras discretas y cálculo numérico) y de cuatro cursos más avanzados (estructuras de datos, lenguajes de programación, organización de computadores y programación de sistemas). Estos cursos enfatizaban el análisis numérico y el hardware, omitiendo la Ingeniería del Software [Tucker y Wegner, 1994].
- El informe de **ACM de 1978** [Austing et al., 1979], proporciona descripciones detalladas de cursos universitarios en Informática, subrayando el término "*Computer Science*" (al igual que el anterior de 1968) y la importancia de la programación. La visión aquí es muy dependiente de la máquina, solamente en uno de los cursos "*opcionales*" avanzados se hace referencia a un curso de "*Diseño y Desarrollo de Software*" que podría incluirse en el currículo; incluye técnicas *top-down* y estructuradas de diseño, formación de equipos de desarrollo y gestión de proyectos. En este momento, se critica el carácter que ha tomado el *Curriculum'78*, por pensarse que se está equiparando Ciencia de la Computación a Programación [Ralston y Shaw, 1980].
- El informe de **la Educational Activities Board de IEEE-CS de 1983**, que describe una serie de áreas para Informática, considerando la orientación *Computer Science and Engineering*. En este informe se hacen recomendaciones sobre material de laboratorio y sobre el propio laboratorio como soporte a las clases teóricas; utiliza una estructura modular para organizar los temas y construir los

cursos, proponiéndose como una base para configurar Planes de Estudios adaptados a distintas situaciones y centros particulares. En este informe se introduce la **Ingeniería del Software** en varios niveles: *como área temática básica*, *como Laboratorio específico de Ingeniería del Software* y *como área temática avanzada*, con un carácter dominado por las técnicas y métodos estructurados, propios de la época, pero subrayando ya la importancia de la fase de requisitos, modelado conceptual, prototipado y validación y verificación.

- El informe de 1989 "*Computing as a discipline*" [Denning et al., 1989], defiende la integración del trabajo de laboratorio con las clases teóricas, subrayando la importancia de introducir aspectos de *diseño* en el currículo y propone que en los cursos introductorios se dé una visión global de toda la carrera, incidiendo en los conceptos fundamentales, mientras que en los cursos superiores se debe ir profundizando en cada uno de esos temas. En este informe se proponen ya las nueve áreas temáticas que, posteriormente, aparecen en el documento ACM/IEEE-CS 91, distinguiendo para cada una de ellas ***Teoría*** (matemáticas subyacentes), ***Abstracción*** (modelado, análisis) y ***Diseño*** (especificar soluciones, estudiar alternativas).

Objetivos del programa de graduación. Perfil de los graduados

El programa debe preparar a los estudiantes para la comprensión de las materias relacionadas con la computación en dos aspectos: *como una disciplina académica* y *como una profesión dentro de su contexto social*. Los estudiantes deben adquirir las habilidades para poder mantenerse actualizados y evaluar las ideas nuevas. Entre estas habilidades se incluyen el leer publicaciones, la asistencia a seminarios y la evaluación de su contenido, así como el trabajo en equipo en proyectos relacionados con problemas de actualidad.

En consecuencia, el primer objetivo del programa de formación ha de ser el proporcionar una cobertura básica, amplia y coherente de la disciplina de la computación. Los estudiantes deben comprender las relaciones existentes entre las diferentes áreas de la computación.

El programa debe preparar a los estudiantes para aplicar sus conocimientos a problemas específicos y con restricciones para elaborar soluciones. Esto incluye la *capacidad de definir un problema claramente*, *determinar su posibilidad de tratamiento, determinar la oportunidad de consultar con expertos, evaluar y elegir una estrategia de solución adecuada; estudiar, especificar, diseñar, implementar, probar, modificar y documentar esa solución, evaluar alternativas y realizar análisis de riesgos del diseño, integrar tecnologías alternativas en la*

solución y comunicar la solución tanto a los compañeros, a los profesionales de otros campos como al público en general. Esto incluye la capacidad de trabajo en equipo durante todo el proceso de resolución de problemas.

El programa debe proporcionar la suficiente exposición del amplio cuerpo de teoría que subyace en el campo de la computación, de forma que los estudiantes aprecien la profundidad intelectual y los elementos abstractos que continuarán retando a los investigadores en el futuro.

Los graduados tienen que tener presente la alta tasa de cambio tecnológico, la tasa de crecimiento relativo en la teoría de la computación y la interacción delicada que tiene lugar entre las dos.

Principios subyacentes en el diseño curricular

En esta propuesta curricular se identifican nueve áreas de conocimiento y tres procesos que caracterizan las diferentes metodologías operativas utilizadas en la investigación y el desarrollo de la computación.

Cada una de las áreas identificadas tiene una base teórica significativa, abstracciones significativas y realizaciones significativas de diseño e implementación. Las nueve áreas son: *Algoritmos y estructuras de datos; Arquitectura; Inteligencia artificial y robótica; Bases de datos y recuperación de la información; Comunicación hombre-máquina; Computación numérica y simbólica; Sistemas operativos; Lenguajes de Programación* y ***Metodología e Ingeniería del Software***. De todas ellas las relacionadas con este Proyecto Docente son:

- ***Comunicación Hombre-Máquina***. El propósito principal de esta área es la transferencia eficiente de información entre el hombre y la máquina. Se incluyen gráficos, factores humanos que afectan a la interacción eficiente, así como la organización y visualización de la información para una utilización efectiva por parte de las personas.
- ***Metodología e Ingeniería del Software***. El propósito principal de esta área es la especificación, el diseño y la producción de grandes sistemas software. El interés se centra en los principios de programación y del desarrollo del software, la verificación y la validación del software y la especificación y producción de sistemas software que sean seguros y fiables.

La Informática es vista simultáneamente como una disciplina Matemática, Científica y de Ingeniería. Los diferentes profesionales en cada una de las áreas emplean diferentes

metodologías operativas, o procesos, en el curso de sus trabajos de investigación, desarrollo y aplicación.

El primero de estos procesos es el llamado **teoría**, está fundamentado en las Matemáticas, y se utiliza en el desarrollo de teorías matemáticas coherentes. Los principales elementos que componen este proceso son: *definiciones y axiomas*, *teoremas*, *pruebas* e *interpretación de resultados*.

El segundo proceso, denominado **abstracción**, está enraizado en las Ciencias Experimentales, constando de los siguientes elementos: *recolección de datos y formulación de hipótesis*, *modelado y predicción*, *diseño de un experimento* y *análisis de resultados*. Cuando las personas hacen abstracción están modelando algoritmos, estructuras de datos o arquitecturas, por ejemplo; prueban hipótesis acerca de esos modelos, toman decisiones de diseño alternativas... A los estudiantes hay que introducirlos en la abstracción a través de las clases y de los laboratorios.

El tercer proceso, llamado **diseño**, está enraizado en la Ingeniería y se utiliza en el desarrollo de un sistema o dispositivo para resolver un determinado problema. Consta de las siguientes partes: *requisitos*, *especificaciones*, *diseño*, *implementación* y *pruebas*. Cuando los profesionales de la computación diseñan, han de implicarse en la conceptualización y la realización de sistemas en el contexto de las restricciones del mundo real. Los estudiantes aprenden diseño mediante la experiencia directa y mediante el estudio de los diseños de otros. Los proyectos de laboratorio se orientan hacia el diseño, proporcionando a los estudiantes una experiencia de primera mano en el desarrollo de un sistema o un componente de un sistema para resolver un problema particular. Estos proyectos de laboratorio enfatizan en la síntesis de las soluciones prácticas a problemas y, por tanto, obligan a los estudiantes a evaluar las alternativas, costes y rendimientos en el contexto de las restricciones del mundo real. Los estudiantes desarrollan la capacidad de realizar estas evaluaciones viendo y discutiendo ejemplos de diseños, así como recibiendo información sobre sus propios diseños.

Conceptos recurrentes

Existen ciertos conceptos fundamentales que aparecen de forma recurrente en el diseño de los currículos de computación, los cuales representan un papel importante en el diseño de los cursos individuales. Estos conceptos son *ideas, materias, principios* y *procesos* que ayudan a unificar una disciplina académica en su substrato. En la propuesta curricular ACM/IEEE-CS91 se han identificado doce conceptos, a saber:

- **Ligadura**. El proceso de concretar abstracciones mediante la asociación de propiedades adicionales a dicha abstracción. Por ejemplo, la asociación de procesos a procesadores o la creación de instancias concretas a partir de descripciones abstractas.
- **Complejidad de los grandes problemas**. Los efectos del crecimiento no lineal de la complejidad cuando crece el tamaño del problema.
- **Modelos conceptuales y formales**. Diferentes modos de formalizar, caracterizar, visualizar y concebir ideas o problemas. Los ejemplos incluyen modelos conceptuales del tipo de los tipos abstractos de datos y los modelos semánticos, y lenguajes visuales utilizados en la especificación y diseño de sistemas, tales como flujos de datos y diagramas entidad-relación.
- **Consistencia y compleción**. Incluye la consistencia de un conjunto de axiomas que sirven como especificación formal, la consistencia de la teoría con los hechos observados y la consistencia interna de un lenguaje. La compleción incluye la suficiencia de un conjunto de axiomas dados para capturar todos los comportamientos que se desea, la suficiencia funcional de los sistemas software y hardware, así como la capacidad de un sistema para comportarse adecuadamente en condiciones de error o situaciones no previstas.
- **Eficiencia**. La medida de los costes relativos de los recursos tales como espacio, tiempo, dinero o personal.
- **Evolución**. El hecho del cambio y sus implicaciones.
- **Niveles de abstracción**. La naturaleza y uso de la abstracción en computación; la utilización de abstracciones para manejar la complejidad, estructurar sistemas, ocultar detalles y capturar patrones recurrentes. La capacidad de representar una entidad o sistema por abstracción tiene diferentes niveles de detalle y especificidad.
- **Ordenación en el espacio**. De los conceptos de localización y proximidad en la disciplina de la Informática. Además de la localización física, como en las redes o en la memoria de un computador, incluye el emplazamiento de la organización (por ejemplo, de procesadores y procesos; definiciones de tipo y las operaciones asociadas) y la localización conceptual (por ejemplo, el ámbito del software, la cohesión y el acoplamiento).
- **Ordenación en tiempo**. El concepto de tiempo en la ordenación de los eventos. Esto incluye el tiempo *como un parámetro en los modelos formales* (como por

ejemplo en la lógica temporal), *como sinónimo de sincronización de procesos* o *como una parte esencial en la ejecución de los algoritmos*.

- **Reutilización**. La capacidad de una técnica, concepto o componente de sistema para ser reutilizado en un nuevo contexto o situación.
- **Seguridad**. La capacidad de los sistemas software y hardware para responder y defenderse de las peticiones no autorizadas, inapropiadas o imprevistas.
- **Decisiones y consecuencias**. El fenómeno de las decisiones en Informática y de las consecuencias que acarrean; los efectos técnicos, económicos, culturales que se derivan de la selección de una determinada alternativa de diseño.

El papel de los laboratorios

Un currículo de formación en Informática se compone, idealmente, de un programa integrado de clases teóricas y experiencias de laboratorio.

El papel de los laboratorios es muy importante en el diseño de un currículo en formación Informática, presentando las siguientes características y ventajas:

- Los laboratorios permiten demostrar la aplicación de los principios en el diseño, implantación y prueba de los sistemas software.
- Los laboratorios enfatizan las técnicas que utilizan las herramientas actuales y conducen hacia métodos experimentales adecuados incluyendo la presentación oral y escrita de los hallazgos.
- Los ejercicios de laboratorio han de estar diseñados para mejorar la experiencia del estudiante en las metodologías del software mediante el desarrollo de diseños e implantaciones.
- Las experiencias del laboratorio incrementan la capacidad para resolver problemas, las habilidades analíticas y la capacitación profesional.

Se diferencian dos tipos de laboratorios, *laboratorios abiertos* y *laboratorios cerrados*. Los primeros consisten en una asignación de trabajo que se llevará a cabo sin supervisión. Un laboratorio cerrado conlleva la asignación de trabajo de forma planificada, estructurada y supervisada. La finalización de un trabajo de laboratorio debe de estar acompañada de un informe oral y/o escrito por parte del estudiante.

La unidad de conocimiento de Metodología e Ingeniería del Software

El currículo en Informática se organiza sobre la base de unidades del conocimiento. Se entiende por unidad de conocimiento *la designación de una colección coherente de materias dentro de una de las nueve áreas principales de la disciplina de la computación* [Tucker et al., 1991a].

Para cada unidad de conocimiento se especifican los contenidos bajo el epígrafe de materias, los laboratorios propuestos, la posible relación con otras unidades de conocimiento, las unidades de conocimiento que son prerrequisito y si la unidad del conocimiento es requisito para otras unidades de conocimiento.

Las unidades de conocimiento que se definen en la propuesta curricular ACM/IEEE-CS91, relacionadas con el presente Proyecto Docente caen dentro del área de Metodología e Ingeniería del Software; cuyas unidades de conocimiento a su vez se recogen en la Tabla 4.12.

SE: Metodología e Ingeniería del Software (se recomiendan unas 44 horas teóricas)
SE1: Conceptos fundamentales para la resolución de problemas
SE2: El proceso de desarrollo del software
SE3: Requisitos y especificaciones del software
SE4: Diseño e implementación del software
SE5: Verificación y validación

Tabla 4.12. Unidades de conocimiento para el área de Metodología e Ingeniería del Software

De las unidades de conocimiento de esta área se describen aquéllas que se estima que deben estar contempladas en las asignaturas que se ajustan al perfil y al entorno en el que se va a desarrollar la docencia objeto de la plaza a concurso.

- **SE2.- El proceso de desarrollo del software.**

  Introducción a los modelos y elementos relacionados con el desarrollo de software de calidad. Utilización de entornos y herramientas que faciliten el diseño y la implantación de grandes sistemas software. El papel y la utilización de los estándares.

  1) Materias.

  1. Modelos del ciclo de vida del desarrollo del software.
  2. Objetivos del diseño del software.
  3. Documentación.
  4. Gestión y control de configuraciones.
  5. Elementos de la fiabilidad del software.
  6. Mantenimiento.

7. Herramientas de especificación y diseño, herramientas de implementación.

2) Laboratorios (abiertos).

a. Implementar un prototipo para una especificación determinada.
b. Dado un diseño de software y una implementación intermedia de un desarrollo iterativo, completar la implementación correspondiente a la siguiente iteración.
c. Crítica de un conjunto de documentación determinado.
d. Dada una implementación, unas especificaciones y un conjunto de nuevas especificaciones, modificar el código de acuerdo a las nuevas especificaciones.

3) Relacionado con: Especificaciones y requisitos del software.

4) Prerrequisitos: Tipos abstractos de datos.

5) Requisito para: Diseño e implementación del software.

- **SE3.- Especificaciones y requisitos del software.**

Introducción al desarrollo de especificaciones formales e informales para la definición de los requisitos de un sistema software.

1) Materias.

a. Especificaciones informales.
b. Especificaciones formales, especificaciones algebraicas de los TAD, precondiciones y poscondiciones.

2) Laboratorios.

a. Producir un documento de análisis de requisitos.
b. Producir un documento con las especificaciones formales correspondientes a un conjunto de especificaciones informales.

3) Relacionado con: El proceso de desarrollo del software, control de tipos, semántica de los lenguajes.

4) Prerrequisitos: Tipos abstractos de datos.

5) Requisito para: Diseño e implementación del software, verificación y validación del software.

- **SE4.- Diseño e implementación del software.**

Introducción a los paradigmas principales que rigen el diseño y la implementación de grandes sistemas software.

1) Materias.

a. Diseño dirigido por funciones/procesos.

b. Diseño ascendente, soporte para reutilización.
c. Estrategias de implementación (por ejemplo, ascendente, descendente, equipos)
d. Elementos de la implementación, mejora de rendimientos, depuración.

2) Laboratorios (abiertos).

a. Realizar un diseño objeto para un conjunto de especificaciones.
b. Implementar el diseño anterior.
c. Dado una especificación de problema y un conjunto de módulos ejecutables con sus especificaciones, realizar un diseño ascendente, reutilizando lo más posible.
d. Realizar una implementación descendente para un diseño software dado.

3) Relacionado con: Bases de datos, paradigmas de programación.

4) Prerrequisitos: IS2, IS3.

Otras consideraciones

La propuesta curricular ACM/IEEE-CS91 propone un curso optativo de nivel avanzado en Ingeniería del Software, centrado en los métodos y herramientas necesarios para incrementar la calidad, además de reducir el coste y la complejidad de mantenimiento de los sistemas software. Este curso tiene como prerrequisitos, además de diversas unidades de las áreas de algoritmos y estructuras de datos, cálculo numérico, lenguajes de programación y asuntos tanto éticos como sociales de la profesión, así como todo el área de metodología e Ingeniería del Software.

Se incluye una propuesta detallada de un programa universitario en Informática con énfasis en la Ingeniería del Software (*Implementation G: A Program in Computer Science – Software Engineering Emphasis*) [Tucker et al., 1991a].

En cuanto a la tecnología de objetos, cabe decir que tiene muy poca presencia en esta propuesta curricular (unas diez horas teóricas divididas en cuatro áreas [Osborne, 1992]), quizás porque ésta se centra en describir de una forma general el esqueleto básico de la Informática como disciplina, más que en establecer su cuerpo de conocimiento.

Aunque no es perfecta, esta propuesta curricular ha sido la que mayor influencia ha tenido en un plano internacional, incluyendo a España. Durante todos los años que ha estado en vigor el *Computing Curricula 91* se han propuesto diferentes modificaciones para incorporar o enfatizar diferentes aspectos, como la tecnología de objetos, la Ingeniería del Software o los Sistemas de Información; algunas de estas propuestas son: [Scragg et al., 1994; Knight et al., 1994; Shackelford y LeBlanc, 1994; Hirmanpour et al., 1995; Reynolds y Fox, 1996; Pham, 1997; Jackson et al., 1997].

4.4.1.3 La propuesta conjunta ACM/IEEE-CS (Curricula 2001)

Reconocidas las limitaciones del *Computing Curricula 91* y el tremendo avance de la Informática en estos años, de nuevo ACM e IEEE-CS se unen para crear un nuevo grupo de trabajo que defina una nueva propuesta curricular en el campo de la Ciencia de la Computación [ACM/IEEE-CS, 2001].

Principios

Sobre la base del análisis de pasadas propuestas curriculares y de los cambios técnicos y culturales sufridos por la Informática, el equipo de trabajo del *Computing Curricula 2001* ha formulado los siguientes principios que guían su trabajo:

1. La Informática es un campo extenso que se extiende más allá de los límites de la Ciencia de la Computación.
2. La Ciencia de la Computación basa sus fundamentos en una amplia variedad de disciplinas.
3. La rápida evolución de la Ciencia de la Computación requiere una revisión continua del currículo.
4. El desarrollo de un currículo propio de la Ciencia de la Computación debe ser sensitivo a los cambios en la tecnología, a los nuevos desarrollos en Pedagogía y a la importancia del aprendizaje continuo durante toda la vida.
5. El *Computing Curricula 2001* debe ir más lejos de unas unidades de conocimiento para ofrecer una guía significativa en términos de diseño de cursos individuales.
6. El *Computing Curricula 2001* debe identificar las propiedades fundamentales y el conocimiento que todos los estudiantes deben poseer.
7. El cuerpo de conocimiento requerido debe hacerse lo más pequeño posible.
8. El *Computing Curricula 2001* debe tener un ámbito internacional.
9. El desarrollo del *Computing Curricula 2001* debe ser lo más abierto posible.
10. El *Computing Curricula 2001* debe incluir la práctica profesional como un componente integrado en el currículo de los no graduados.
11. El *Computing Curricula 2001* debe incluir discusiones de estrategias y tácticas para la implementación de recomendaciones de alto nivel.

El cuerpo de conocimiento de la Ciencia de la Computación

En el *Computing Curricula 2001* el número de áreas pasa de las nueve del *Computing Curricula 91* a catorce (Tabla 4.13), habiendo sido asignada cada una de ellas a un grupo de trabajo distinto para que definiera el cuerpo de conocimiento asociado a cada área.

El cuerpo de conocimiento se organiza jerárquicamente en tres niveles. El nivel más alto es el **área**, que representa un campo disciplinar particular. Casa área se identifica por dos letras. Las áreas se dividen en divisiones más pequeñas llamadas **unidades**. Cada unidad representa un módulo temática individual dentro de un área. Cada unidad está dividida en **temas** o **tópicos** que son el nivel más bajo de la jerarquía.

Como uno de los objetivos era que el cuerpo de conocimiento fuera tan pequeño como fuera posible, se ha definido un núcleo mínimo consistente en aquellas unidades en las que había un amplio consenso en que el material que representan es esencial para cualquiera que quiera obtener el grado en el campo. Las unidades que forman parte del programa pero que quedan fuera del núcleo son opcionales. Esta división en unidades troncales y opcionales lleva a las siguientes conclusiones:

1. El núcleo se refiere a aquellas unidades que se requieren para que los estudiantes adquieran un grado en Informática.
2. El núcleo no es un currículo completo.
3. El núcleo debe ser completado con material adicional.
4. Las unidades troncales no son necesariamente aquéllas que aparecen en un conjunto de cursos introductorios al comienzo de los estudios.

Para tener una métrica de la cobertura de las unidades se ha elegido la **hora**, correspondiéndose con el tiempo de clase requerido para presentar el material en un formato de clase tradicional. Las unidades troncales tienen asignadas un mínimo de horas lectivas. Esta métrica se ha elegido por motivos de consistencia con anteriores informes. Esta asignación temporal no incluye el tiempo de dedicación fuera de clase, que se considera que es el triple del asignado a cada unidad. Ahora bien, no existe ninguna relación entre las horas de las unidades troncales y la distribución que se puede hacer de éstas para conformar una determinada asignatura o diseñar una propuesta curricular concreta.

El informe considera que una asignatura cuatrimestral puede estar formada por unas 45 sesiones, cada una con una duración entre 50 minutos y una hora. Restando el tiempo para exámenes, las asignaturas durarían unas 40 horas.

En la Figura 4.8 se recoge el cuerpo de conocimiento del *Computing Curricula 2001* expresado en áreas y unidades troncales (subrayadas) y optativas.

Área	Código
Estructuras Discretas	DS
Fundamentos de Programación	PF
Algoritmos y Complejidad	AL
Arquitectura y Organización	AR
Sistemas Operativos	OS
Redes	NC
Lenguajes de Programación	LP
Interacción Persona-Ordenador	HC
Gráficos y Computación Visual	GV
Sistemas Inteligentes	IS
Gestión de la Información	IM
Aspectos Sociales y Profesionales	SP
Ingeniería del Software	**SE**
Ciencia de la Computacional y Métodos Numéricos	CN

Tabla 4.13. Áreas de interés en el *Computing Curricula 2001* [ACM/IEEE-CS, 2001]

Finalmente las unidades se empaquetan en cursos. Los cursos se dividen en el *Computing Curricula 2001* en tres categorías de acuerdo al nivel. Los cursos introductorios son típicamente cursos de nivel básico ofertados el primero y el segundo año de carrera. Los cursos intermedios construyen los fundamentos para un mayor y posterior estudio en el campo, suceden típicamente el segundo y tercer año. Los cursos avanzados se dan en los últimos años y se centran en aquellos temas que requieren una preparación significativa en términos de cursos preliminares.

Las principales diferencias en lo tocante a las áreas entre el *Computing Curricula 1991 y el Computing Curricula 2001*, a parte del número, se pueden resumir en los siguientes puntos:

- *Se han añadido las Estructuras Discretas (DS) como un área separada.* En el *Computing Curricula 1991*, la Matemática Discreta aparece sólo como un prerrequisito para los tópicos que requieren una madurez matemática. El *Computing Curricula 2001* ha decidido enfatizar la dependencia de la Informática de la Matemática Discreta, incluyéndola como una nueva área.
- *La necesidad de incluir la instrucción en el uso de un lenguaje de programación se ha explicitado añadiendo un área específica sobre Fundamentos de Programación. (PF)*. El *Computing Curricula 1991* incluye un área denominada "Introducción a un Lenguaje de Programación", pero se identificaba como un componente opcional en el currículo.

Se esperaba que los alumnos adquiriesen los conocimientos básicos sobre programación de ordenadores en el Instituto, pero esto no ha llegado a ocurrir, lo que ha provocado la creación de esta área.

- *El área sobre los Aspectos Sociales, Éticos y Profesionales (SP) se ha integrado en el currículo, dándole la misma importancia que las demás áreas.* Esto se ha producido debido a que, desde la publicación del *Computing Curricula 1991*, se ha producido un consenso creciente en el hecho de que todos los estudiantes deben ser conscientes del impacto social de su trabajo y de las responsabilidades éticas de ser un profesional de la Informática.
- *Gráficos y Computación Visual (GV) y Redes (NC) se han añadido como áreas separadas.* Muchas de las áreas se han expandido en gran medida desde 1991. Como resultado, algunas de ellas que antes simplemente eran temas de un área más general, han crecido tanto que ya no cuadran adecuadamente en la estructura anterior.
- *Cuatro áreas, la de Algoritmos y estructuras de Datos (AL), Inteligencia Artificial y Robótica (AI), Bases de Datos y Recuperación de Información (DB) y Computación Numérica y Simbólica (NU), cambian significativamente de nombre, pasando a denominarse Algoritmos y Complejidad (AL), Sistemas Inteligentes (IS), Gestión de la Información (IM) y Ciencia Computacional (CN)*. La justificación de este cambio de nombres es la de identificar mejor los nuevos contenidos que se han añadido al área a lo largo de los diez años transcurridos.

DS. Discrete Structures (43 core hours)
- DS1. Functions, relations, and sets (6)
- DS2. Basic logic (10)
- DS3. Proof techniques (12)
- DS4. Basics of counting (5)
- DS5. Graphs and trees (4)
- DS6. Discrete probability (6)

PF. Programming Fundamentals (38 core hours)
- PF1. Fundamental programming constructs (9)
- PF2. Algorithms and problem-solving (6)
- PF3. Fundamental data structures (14)
- PF4. Recursion (5)
- PF5. Event-driven programming (4)

AL. Algorithms and Complexity (31 core hours)
- AL1. Basic algorithmic analysis (4)
- AL2. Algorithmic strategies (6)
- AL3. Fundamental computing algorithms (12)
- AL4. Distributed algorithms (3)
- AL5. Basic computability (6)
- AL6. The complexity classes P and NP
- AL7. Automata theory
- AL8. Advanced algorithmic analysis
- AL9. Cryptographic algorithms
- AL10. Geometric algorithms
- AL11. Parallel algorithms

AR. Architecture and Organization (36 core hours)
- AR1. Digital logic and digital systems (6)
- AR2. Machine level representation of data (3)
- AR3. Assembly level machine organization (9)
- AR4. Memory system organization and architecture (5)
- AR5. Interfacing and communication (3)
- AR6. Functional organization (7)
- AR7. Multiprocessing and alternative architectures (3)
- AR8. Performance enhancements
- AR9. Architecture for networks and distributed systems

OS. Operating Systems (18 core hours)
- OS1. Overview of operating systems (2)
- OS2. Operating system principles (2)
- OS3. Concurrency (6)
- OS4. Scheduling and dispatch (3)
- OS5. Memory management (5)
- OS6. Device management
- OS7. Security and protection
- OS8. File systems
- OS9. Real-time and embedded systems
- OS10. Fault tolerance
- OS11. System performance evaluation
- OS12. Scripting

NC. Net-Centric Computing (15 core hours)
- NC1. Introduction to net-centric computing (2)
- NC2. Communication and networking (7)
- NC3. Network security (3)
- NC4. The web as an example of client-server computing (3)
- NC5. Building web applications
- NC6. Network management
- NC7. Compression and decompression
- NC8. Multimedia data technologies
- NC9. Wireless and mobile computing

PL. Programming Languages (21 core hours)
- PL1. Overview of programming languages (2)
- PL2. Virtual machines (1)
- PL3. Introduction to language translation (2)
- PL4. Declarations and types (3)
- PL5. Abstraction mechanisms (3)
- PL6. Object-oriented programming (10)
- PL7. Functional programming
- PL8. Language translation systems
- PL9. Type systems
- PL10. Programming language semantics
- PL11. Programming language design

Note: The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.

HC. Human-Computer Interaction (8 core hours)
- HC1. Foundations of human-computer interaction (6)
- HC2. Building a simple graphical user interface (2)
- HC3. Human-centered software evaluation
- HC4. Human-centered software development
- HC5. Graphical user-interface design
- HC6. Graphical user-interface programming
- HC7. HCI aspects of multimedia systems
- HC8. HCI aspects of collaboration and communication

GV. Graphics and Visual Computing (3 core hours)
- GV1. Fundamental techniques in graphics (2)
- GV2. Graphic systems (1)
- GV3. Graphic communication
- GV4. Geometric modeling
- GV5. Basic rendering
- GV6. Advanced rendering
- GV7. Advanced techniques
- GV8. Computer animation
- GV9. Visualization
- GV10. Virtual reality
- GV11. Computer vision

IS. Intelligent Systems (10 core hours)
- IS1. Fundamental issues in intelligent systems (1)
- IS2. Search and constraint satisfaction (5)
- IS3. Knowledge representation and reasoning (4)
- IS4. Advanced search
- IS5. Advanced knowledge representation and reasoning
- IS6. Agents
- IS7. Natural language processing
- IS8. Machine learning and neural networks
- IS9. AI planning systems
- IS10. Robotics

IM. Information Management (10 core hours)
- IM1. Information models and systems (3)
- IM2. Database systems (3)
- IM3. Data modeling (4)
- IM4. Relational databases
- IM5. Database query languages
- IM6. Relational database design
- IM7. Transaction processing
- IM8. Distributed databases
- IM9. Physical database design
- IM10. Data mining
- IM11. Information storage and retrieval
- IM12. Hypertext and hypermedia
- IM13. Multimedia information and systems
- IM14. Digital libraries

SP. Social and Professional Issues (16 core hours)
- SP1. History of computing (1)
- SP2. Social context of computing (3)
- SP3. Methods and tools of analysis (2)
- SP4. Professional and ethical responsibilities (3)
- SP5. Risks and liabilities of computer-based systems (2)
- SP6. Intellectual property (3)
- SP7. Privacy and civil liberties (2)
- SP8. Computer crime
- SP9. Economic issues in computing
- SP10. Philosophical frameworks

SE. Software Engineering (31 core hours)
- SE1. Software design (8)
- SE2. Using APIs (5)
- SE3. Software tools and environments (3)
- SE4. Software processes (2)
- SE5. Software requirements and specifications (4)
- SE6. Software validation (3)
- SE7. Software evolution (3)
- SE8. Software project management (3)
- SE9. Component-based computing
- SE10. Formal methods
- SE11. Software reliability
- SE12. Specialized systems development

CN. Computational Science (no core hours)
- CN1. Numerical analysis
- CN2. Operations research
- CN3. Modeling and simulation
- CN4. High-performance computing

Figura 4.8. Cuerpo de conocimiento del *Computing Curricula 2001* [ACM/IEEE-CS, 2001]

La unidad de conocimiento de Ingeniería del Software

En la Tabla 4.14 se recogen las unidades para el área de conocimiento de Ingeniería del Software en el *Computing Curricula 2001*. Se sigue el convenio de subrayar aquellas unidades que se consideran troncales.

SE: Ingeniería del Software (31 horas troncales)
SE1: Diseño del software
SE2: Uso de APIs
SE3: Herramientas y entornos software
SE4: Proceso software
SE5: Requisitos software y especificaciones
SE6: Validación del software
SE7: Evolución del software
SE8: Gestión de proyectos software
SE9: Desarrollo basado en componentes
SE10: Métodos formales
SE11: Confiabilidad del software
SE12: Desarrollo de sistemas especializados

Tabla 4.14. Unidades de conocimiento para el área de Ingeniería del Software

En el *Computing Curricula 2001* la Ingeniería del Software se entiende como la disciplina relacionada con la aplicación de la teoría, conocimiento y práctica a la construcción efectiva y eficiente de sistemas software que satisfagan los requisitos de los usuarios y clientes. La Ingeniería del Software es aplicable a cualquier sistema, independientemente de su tamaño. La Ingeniería del Software comprende todas las fases del ciclo de vida de un sistema software. La Ingeniería del Software emplea método de ingeniería, procesos, técnicas y métricas. Se beneficia del uso de herramientas para la gestión del desarrollo del software.

Las unidades del área de conocimiento se reparten por los diferentes cursos introductorios e intermedios propuestos en el *Computing Curricula 2001*, y además se proponen (pero no se detallan) una serie de cursos avanzados sobre Ingeniería del Software, los cuales se recogen en la Tabla 4.15.

A continuación se van a exponer los tópicos recomendados en cada una de las unidades del área de Ingeniería del Software.

- **SE1.- Diseño del software.**

  - Mínimo de tiempo: 8 horas.

  - Tópicos:

1. Conceptos y principios de diseño.
2. Patrones de diseño.
3. Arquitectura software.
4. Diseño estructurado.
5. Análisis y diseño orientado a objetos.
6. Diseño de componentes.
7. Diseño para reutilización.

- **SE2.- Uso de APIs.**

  - Mínimo de tiempo: 5 horas.

  - Tópicos:

    1. Programación con APIs.
    2. Inspectores (navegadores) de clases y herramientas relacionadas.
    3. Programación por ejemplo.
    4. Depuración en el entorno de API.
    5. Introducción a la programación basada en componentes.

- **SE3.- Herramientas y entornos software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Entornos de desarrollo.
    2. Herramientas para el análisis de requisitos y el modelado en diseño.
    3. Herramientas de prueba.
    4. Herramientas para la gestión de la configuración.
    5. Mecanismos para la integración de herramientas.

- **SE4.- Proceso software.**

  - Mínimo de tiempo: 2 horas.

  - Tópicos:

    1. Ciclo de vida del software y modelos de procesos.
    2. Modelos de seguimiento de procesos.
    3. Métricas para proceso software

- **SE5.- Requisitos software y especificaciones.**

  - Mínimo de tiempo: 4 horas.

  - Tópicos.

1. Obtención (elicitación) de requisitos.
2. Técnicas de modelado en el análisis de requisitos.
3. Requisitos funcionales y no funcionales.
4. Prototipado.
5. Conceptos básicos de las técnicas de especificación formal.

- **SE6.- Validación del software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Planificación de la validación.
    2. Fundamentos de la prueba del software.
    3. Técnicas de caja negra y de caja blanca.
    4. Pruebas de unidad, de integración, de validación y de sistema.
    5. Pruebas orientadas a objetos.
    6. Inspecciones.

- **SE7.- Evolución del software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Mantenimiento del software.
    2. Características del software mantenible.
    3. Reingeniería.
    4. Sistemas legados.
    5. Reutilización del software.

- **SE8.- Gestión de proyectos software.**

  - Mínimo de tiempo: 3 horas.

  - Tópicos:

    1. Gestión de equipos.
    2. Planificación.
    3. Técnicas de estimación y medida del software.
    4. Análisis de riesgos.
    5. Aseguramiento de la calidad del software.
    6. Gestión de la configuración del software
    7. Herramientas de gestión de proyectos.

- **SE9.- Desarrollo basado en componentes [opcional].**

  - Tópicos:

    1. Fundamentos.
    2. Técnicas básicas.
    3. Aplicaciones.
    4. Arquitectura de los sistemas basados en componentes.
    5. Diseño orientado a componentes.
    6. Manejo de eventos.
    7. *Middleware*.

- **SE10.- Métodos formales [opcional].**

  - Tópicos:

    1. Conceptos sobre métodos formales.
    2. Lenguajes de especificación formal.
    3. Especificaciones ejecutables y no ejecutables.
    4. Pre y post aserciones.
    5. Verificación formal.

- **SE11.- Confiabilidad del software [opcional].**

  - Tópicos:

    1. Modelos de confiabilidad del software.
    2. Redundancia y tolerancia a fallos.
    3. Clasificación de defectos.
    4. Métodos probabilísticas de análisis.

- **SE12.- Desarrollo de sistemas especializados [opcional].**

  - Tópicos:

    1. Sistemas en tiempo real.
    2. Sistemas cliente-servidor.
    3. Sistemas distribuidos.
    4. Sistemas paralelos.
    5. Sistemas basados en la web.
    6. Sistemas altamente integrados.

Cursos avanzados en el área de Ingeniería del Software
CS390 Desarrollo de software avanzado
CS301 Ingeniería del Software
CS392 Diseño de software
CS393 Ingeniería del Software y especificación formal
CS394 Ingeniería del Software empírica
CS395 Mejora del proceso software
CS396 Desarrollo basado en componentes
CS397 Entornos de programación
CS398 Sistemas de seguridad crítica

Tabla 4.15. Propuesta de cursos avanzados en Ingeniería del Software

El área de conocimiento de Interacción Persona-Ordenador (HC) y la unidad de Modelado de datos (IM3) perteneciente al área de conocimiento de Gestión de la Información (IM) guardan una estrecha relación con el área de conocimiento de Ingeniería del Software (SE).

Concretamente el campo de la Interacción Persona-Ordenador ha sufrido un avance considerable desde que en 1995 Sutcliffe en el prefacio de la segunda edición de su libro dedicado a la Interacción Persona-Ordenador [Sutcliffe, 1995] argumentaba que: "*en 1988 había muy pocos libros dedicados a la Interacción Persona-Ordenador... y el tema es tenido en cuenta en muy pocos cursos de Ciencia de la Computación*". En 1995, cuando aparece su libro existe una oferta más considerable de libros en esta parcela, lo que denota su reconocimiento, y hoy en día aparece como un área de conocimiento independiente en el *Computer Curricula 2001*. Pero además, el campo de la Interacción Persona-Ordenador cuenta con varias propuestas curriculares [Hefley et al., 1992; Kirby et al., 1994; Kirby et al., 1995; BCS, 1995] y con voces que piden una mayor integración de ésta con la Ingeniería del Software, presentando la figura del *ingeniero en usabilidad* [Dowell y Long, 1989; Faulkner y Culwin, 2000].

Plan de Estudios y la Ingeniería del Software

Para organizar las unidades y materias en una propuesta curricular concreta, el *Computing Curricula 2001* establece tres niveles de asignaturas para una titulación: asignaturas de iniciación, correspondientes al primero o segundo cuatrimestre de la titulación; asignaturas intermedias, correspondientes al segundo o tercer cuatrimestre y que establecen los fundamentos para estudios más avanzados de especialización; y las asignaturas avanzadas, que se imparten en cuatrimestres posteriores y se centran en las materias que requieren un estudio más exhaustivo en relación con el trabajo realizado en cursos anteriores.

La organización de los contenidos en estos tres tipos de asignaturas se puede hacer de diversas maneras dependiendo de los objetivos de la titulación, la Universidad, el ámbito

social... El Comité que ha elaborado el documento asume seis estrategias, como las más aceptadas, para la definición de las asignaturas de iniciación.

Según estas recomendaciones, la Ingeniería del Software es un área para la que sólo se debería cubrir, en las asignaturas de iniciación, un subconjunto de sus materias troncales:

SE1. Diseño del Software. Materias para las asignaturas de iniciación:

- Principios y conceptos fundamentales del diseño.
- Análisis y diseño orientado a objetos.
- Diseña para reutilización.

SE2. Utilización de APIs. Materias para las asignaturas de iniciación:

- Programación con APIs.
- Inspectores y visualizadores de clases y herramientas relacionadas.
- Programación a través de ejemplo.
- Depuración en el entorno de API.

SE3. Herramientas y entornos de software. Materias para las asignaturas de iniciación:

- Entornos de programación.
- Herramientas para pruebas.

SE5. Requisitos y especificaciones del software. Materias para las asignaturas de iniciación:

- Importancia de la especificación en el proceso software.

SE6. Validación del software. Materias para las asignaturas de iniciación:

- Fundamentos de pruebas, incluyendo la creación de un plan de pruebas y la generación de casos de prueba.

El *Computing Curricula 2001* recomienda un esquema de tres cuatrimestres para cubrir las asignaturas de iniciación. Hay que destacar que, cuando se revisa la manera en que se construyen las asignaturas, se observa una discrepancia profunda con el sistema educativo español. Para cualquiera de las seis estrategias de construcción para las asignaturas de iniciación, las recomendaciones realizan cada asignatura con materias de diferentes áreas. Así, un esquema de *anchura primero* – que pretende dar una visión global de la Informática en las asignaturas de iniciación – estaría constituido por una secuencia de tres asignaturas para los tres primeros cuatrimestres de la titulación ($CS101_B$, $CS102_B$ y $CS103_B$), más otra asignatura de

introducción $CS100_B$. La troncalidad del área de Ingeniería del Software se cubriría en un 30% en los cuatrimestres segundo y tercero (asignaturas $CS102_B$ y $CS103_B$).

Además, e independientemente de la estrategia elegida, el informe propone cinco asignaturas de iniciación adicionales (CS105, CS106, CS115, CS120 y CS130) en las que se cubre un 6% adicional de la troncalidad de Ingeniería del Software. Estas nueve asignaturas, repartidas en los tres primeros cuatrimestres de la titulación dan como resultado tres asignaturas a la semana en cada cuatrimestre, en las que se cubre entre un 36% y un 64% de la troncalidad del área de Ingeniería del Software.

Para las asignaturas intermedias, también se proponen varias estrategias que, en este caso, son temáticas: por materias (8 asignaturas), intensivo (5 asignaturas), sistemas (9 asignaturas) y orientado a web (8 asignaturas). De las 19 asignaturas propuestas, la mayoría comunes para varias de las estrategias temáticas, sólo cuatro contienen materias troncales de Ingeniería del Software: $CS255_{\{S,W\}}$, $CS290_{\{T\}}$, $CS291_{\{S\}}$ y $CS292_{\{C,W\}}$. Sin embargo, la tasa de cobertura del área de Ingeniería del Software en estas asignaturas es muy alto. Por ejemplo, la asignatura $CS292_{\{C,W\}}$ contiene un 50% de la troncalidad de Ingeniería del Software.

En cuanto a las asignaturas avanzadas, el *Computing Curricula 2001* propone para el área de Ingeniería del Software las asignaturas recogidas en la Tabla 4.15.

Por último, el *Computing Curricula 2001* hace un especial hincapié en la necesidad de que todos los estudiantes realicen un proyecto, con trabajo en equipo, como parte indispensable de su Plan de Estudios.

4.4.1.4 Modelo de currículo para las artes liberales

Este modelo de currículo [Gibbs y Tucker, 1986] presenta una aproximación alternativa a la disciplina de la Informática, pone un gran énfasis en que la Ciencia de la Computación y tiene un cuerpo coherente de principios científicos. Define la Informática como un estudio sistemático de propiedades formales, implementación y aplicación de algoritmos y estructuras de datos.

La Ingeniería en esta propuesta curricular está ausente; en una posterior revisión [Walker y Schneider, 1996] se le asignan trece horas a la Ingeniería del Software (frente a las cuarenta y cuatro que recomendaba el *Computing Curricula 91*). Se presenta también un curso optativo que lleva por nombre *Ingeniería del Software*.

Existe alguna propuesta para la creación de un programa de estudios centrado en la Ingeniería del Software para su desarrollo en las Escuelas de Artes Liberales, como por ejemplo [Tymann et al., 1994].

4.4.2 Currículos centrados en los Sistemas de Información

Los Sistemas de Información son parte esencial de las organizaciones. Son sistemas complejos que requieren tanto experiencia técnica como de organización para el diseño, el desarrollo y la gestión.

Hay una estrecha relación entre los Sistemas de Información y la Ciencia de la Computación y la Ingeniería del Software. Sin embargo, hay grandes diferencias ya que los Sistemas de Información se concentran en la parte organizativa y en la aplicación de las tecnologías de la información para sus objetivos.

Ninguna de las propuestas curriculares conjuntas de ACM/IEEE-CS entra en los Sistemas de Información, ya que son ajenos a la aplicación de las tecnologías de la información a los Sistemas de Información en las empresas u organizaciones; es decir, estarían cercanos a lo que en España se denomina *Ingeniería Técnica en Informática de Gestión*.

Dentro de un Plan de Estudios centrado en los Sistemas de Información, la Ingeniería del Software es una disciplina instrumental y su importancia relativa dependerá del grado de orientación tecnológica que se le quiera dar a la titulación, así hay titulaciones basadas en Sistemas de Información más orientadas al mundo empresarial, mientras otras están más orientadas al desarrollo de aplicaciones; estas últimas son las que utilizan conocimientos de Ingeniería del Software.

El desarrollo de currículos para Sistemas de Información comienza a principios de la década de los setenta, con el Plan de Estudios definido por la ACM [Ashenhurst, 1972]. Las razones de esta incursión en terrenos tradicionalmente ajenos fueron [Camps, 1999]:

1. El sector de los Sistemas de Información era y es el sector de aplicación de la Informática que más técnicos solicita y emplea.
2. En EEUU algunos departamentos de Ciencia de la Computación impartían enseñanzas propias de los Sistemas de Información.

4.4.2.1 Antecedentes de los modelos de currículos para los Sistemas de Información

La cronología de los diferentes currículos definidos en el campo de los Sistemas de Información comienza a principios de la década de los setenta con una propuesta curricular liderada por ACM. Los hechos más relevantes se presentan a continuación:

- Mayo de 1972: *ACM Graduate Professional Programs in Information Systems* [Ashenhurst, 1972].
- Diciembre de 1973: *ACM Undergraduate Programs in Information Systems* [Couger, 1973].
- Marzo de 1981: *ACM Educational Programs and Information Systems* [ACM, 1981; Nunamaker, 1981].
- 1981 *DPMA Curriculum for Undergraduate Information Systems Education* [DPMA, 1981].
- En el año 1982, **ACM** elabora un nuevo informe para la enseñanza de los Sistemas de Información, el currículo **ACM-IS-82** [Nunamaker et al., 1982]. En él se hacen las siguientes distinciones entre el currículo en Ciencia de la Computación y el currículo para Sistemas de Información:
    - El currículo de Sistemas de Información enseña conceptos y procesos de Sistemas de Información, en dos contextos; conocimientos de organización y gestión, y conocimientos técnicos sobre Sistemas de Información. Por el contrario, la Ciencia de la Computación tiende a ser enseñada en un entorno de matemáticas, algoritmos y tecnología.
    - En cuanto a conocimientos técnicos, el currículo de Sistemas de Información pone substancial énfasis en la capacidad para desarrollar la estructura de un Sistema de Información para una organización (institución/empresa) y para diseñar e implementar aplicaciones. Al titulado en Ciencia de la Computación se le suele hablar menos de análisis de requisitos de información y de consideraciones organizativas, pero adquiere mayores conocimientos en desarrollo de algoritmos, programación, software de sistemas y hardware.
- Durante 1990 se desarrolla el **IS'90** propuesto por **DPMA** (*Data Processing Management Association*) [Longenecker y Feinstein, 1991].
- En 1994 aparece el **IS'94** de **DPMA** [Longenecker et al., 1994].
- Quince años después del currículo **ACM-IS-82**, en 1997 ACM vuelve a tratar los Planes de Estudio para los Sistemas de Información, cuando conjuntamente con la **AITP** (*Association of Information Technology Professionals*), antes **DPMA**, y la **AIS**

(*Association for Information Systems*) propusieron un nuevo currículo para los Sistemas de Información, el **IS'97** [Davis et al., 1997], de amplia repercusión.

- Actualmente se está trabajando una nueva actualización denominada **IS'2000**.
- Tanto el **IS'97** como el **IS'2000** son Planes de Estudios para estudiantes no graduados. Existe una propuesta de *master* especializado en Sistemas de Información, denominado **MSIS2000** [Gorgone et al., 1999].

4.4.2.2 Ingeniería del Software en los currículos para los Sistemas de Información

La presencia de la Ingeniería del Software en estos modelos de currículo empieza a ser significativa a partir del modelo IS'90 definido por la **DPMA** [Longenecker y Feinstein, 1991].

	ÁMBITO	TEMAS
SI'97.7-Análisis y diseño lógico	Ofrece una comprensión del proceso de desarrollo y modificación de los sistemas software. Permite a los alumnos evaluar y escoger una metodología. Enfatiza la comunicación entre las partes interesadas. ADOO. Modelado de datos. Ciclo de vida estándar.	Fases del ciclo de vida; técnicas de entrevista; JAD; DOO; prototipado; diseño de bases de datos; análisis de riesgos; gestión de proyectos; métricas de calidad del software; evaluación y adquisición de paquetes software; código de ética profesional.
SI'97.8-Diseño físico e implementación con SGBD	Diseño e implementación de Sistemas de Información con un SGBD.	Modelado de datos: herramientas y técnicas; paradigma estructurado y OO; modelos de bases de datos; CASE; repositorios; *datawarehouse*; IGU; cliente-servidor; conversiones; mantenimiento; formación de usuarios.
SI'97.9-Diseño físico e implementación con entornos de programación	Diseño físico, programación, prueba e implantación de un sistema. Implementación OO y cliente-servidor utilizando entornos de desarrollo.	Selección del entorno de programación cliente-servidor; construcción de software: estructurado, orientado a eventos y OO; pruebas; calidad del software; formación de usuarios; gestión de la configuración; mantenimiento; ingeniería inversa y reingeniería.
SI'97.10-Gestión de proyectos y práctica	Cubre los factores necesarios de la gestión del desarrollo de sistemas. Cubre tanto los aspectos técnicos como los de comportamiento. Se centra en la gestión del desarrollo de sistemas de nivel empresarial.	Gestión del ciclo de vida; integración de sistemas y bases de datos; gestión de redes y cliente-servidor; métricas para la gestión de proyectos y para la evaluación del rendimiento de sistemas; gestión de recursos humanos; análisis de coste-beneficio; técnicas de presentación; gestión de cambios.

Tabla 4.16. Descripción de los cursos más relacionados con la Ingeniería del Software del IS'97

En el último modelo de currículo **IS'97** definido por la **ACM**, la **AIS** y la **AITP** (formalmente **DPMA**) [Davis et al., 1997] presenta veinte subáreas significativas, de las que las áreas de **Análisis, diseño e implementación de Sistemas de Información** y de **Gestión de Proyectos** serían las más directamente relacionadas con la Ingeniería del Software, aunque muchas otras tendrían una relación más colateral. Por su parte, el curso que más directamente se relaciona con la Ingeniería del Software es el que lleva el epígrafe **SI'97.7-Análisis y diseño lógico**, estando también relacionados los cursos con epígrafes **SI'97.8, SI'97.9** y **SI'97.10**; la descripción de estos cursos se encuentra en la Tabla 4.16**.**

En el modelo curricular para graduados **MSIS2000**, definido por la ACM y la AIS [Gorgone et al., 1999] presenta una línea profesional directamente relacionada con la Ingeniería del Software, **Analista y diseñador de sistemas**, que trataría temas de: *metodologías de diseño avanzadas (ADOO, RAD, Prototipado), Gestión de proyectos avanzada, Integración de sistemas* y *Consultoría de Sistemas de Información.*

4.4.3 Currículos centrados en la Ingeniería del Software

En su mayor parte, los currículos centrados en la Ciencia de la Computación están más orientados a la formación de científicos en computación que ingenieros, mientras que la realidad empresarial e industrial demanda profesionales que sean capaces de afrontar con éxito los proyectos que paulatinamente quedan inconclusos o con carencias significativas, suponiéndoles graves pérdidas económicas.

Para poder formar profesionales que afronten con garantías los problemas reales de las empresas se necesitan programas que hagan hincapié en la Ingeniería del Software, dado que los programas existentes centrados en la Ciencia de la Computación prestan poca atención a la Ingeniería del Software [Jalics y Golden, 1995; Vaughn, 2000].

Existen diferentes propuestas curriculares que se centran en la Ingeniería del Software, ya sea en el contexto de los estudios de graduación o postgrado, cuya influencia está derivando en la definición de un cuerpo de conocimiento propio de la Ingeniería del Software [Abran et al., 2001c] que sienta la base para el establecimiento de una propuesta curricular de ámbito internacional que tenga a la Ingeniería del Software como el objetivo central de la misma.

Se presentan a continuación las propuestas curriculares elaboradas por el **Instituto de Ingeniería del Software** (SEI – *Software Engineering Institute*) en la **Universidad Carnegie-Mellon** (CMU) en USA y por el **WGSEET** (*Working Group on Software Engineering Education and Training*).

4.4.3.1 Las propuestas del SEI-CMU

En 1985 la Universidad Carnegie Mellon presenta un currículo general en Informática [Shaw, 1985] donde la *Ingeniería del Software* aparece como una asignatura en el nivel intermedio, denominada "*Organización de programas*" ("*programming in the small*", ampliación de TAD, POO, reusabilidad, especificaciones formales e informales...) y en los cursos avanzados en forma de dos asignaturas: "*Ingeniería del Software*" ("*programming in the large*", diseño avanzado y especificación, descomposición en módulos, CASE, prototipado, modelado...) y "*Laboratorio de Ingeniería del Software*" (desarrollo de proyectos en grupo, colaboración con la Industria). En este currículo aparece el concepto de "*nociones recurrentes*" que después aparecerían como novedad en el currículo ACM/IEEE-CS91.

Con posterioridad el Instituto de Ingeniería del Software, SEI a partir de ahora, también en la Carnegie Mellon, realiza diferentes currículos específicos en Ingeniería del Software para estudios de graduación y de postgrado.

Programas de postgrado

El SEI establece primeramente estudios de postgrado, *masters*, centrados en Ingeniería del Software, que se detallan en diferentes informes y artículos entre 1989 y 1991 [Gibbs, 1989; Ardis y Ford, 1989; Ford, 1991a], siendo este último el de mayor difusión.

En el informe de 1991 se realiza un desarrollo muy completo y detallado de las áreas temáticas de Ingeniería del Software. El citado informe incluye un modelo de currículo, numerosa bibliografía sobre Ingeniería del Software y descripciones de revistas de investigación de la disciplina. Incluye también descripciones detalladas de los contenidos de la serie de audiovisuales (cintas de vídeo) que constituyen a su vez un ejemplo de una implementación del modelo de currículo. Se describen, además, los programas para postgraduados en Ingeniería del Software de 23 universidades de todo el mundo.

El material se organiza en cursos universitarios, e incorpora el concepto de "*unidad de conocimiento*", al igual que el ACM/IEEE-CS91. También como esta última propuesta, el currículo se organiza en una serie de materias fundamentales optativas avanzadas o complementarias (20-30% de un currículo), y proyectos prácticos de desarrollo (al menos un 30% del trabajo total del estudiante).

La secuencia que sugiere (aunque en esto no es restrictivo) es la de una aproximación docente, comenzando con una visión de proceso para colocar cada actividad individual en su contexto, para seguir con aspectos de gestión del software y actividades de control, finalizando con las actividades concretas de desarrollo y la visión producto.

En el currículo se incluyen materias tanto de Ingeniería del Software, como, de una forma más amplia, de Ingeniería de Sistemas. Las materias fundamentales se distribuyen en 21 unidades, sin recomendación temporal, describiendo para cada unidad sus contenidos, aspectos de la actividad que son más importantes y objetivos educativos de la unidad. A continuación se ofrece una relación de dichas unidades.

- El proceso de Ingeniería del Software.
- Evolución del software.
- Generación de software.
- Mantenimiento de software.
- Comunicación técnica.
- Gestión de configuraciones software.
- Conceptos de calidad del software.
- Aseguramiento de la calidad del software.
- Organización y gestión de proyectos de software.
- Economía de proyectos software.
- Conceptos de operación del software.
- Análisis de requisitos.
- Especificación.
- Diseño de sistemas.
- Diseño de software.
- Implementación de software.
- Pruebas del software.
- Integración de sistemas.
- Sistemas de tiempo real "empotrados".
- Interfaces hombre-máquina.
- Asuntos de la profesión.

El *master* en Ingeniería del Software que se imparte en la Universidad Carnegie Mellon consta de tres elementos básicos [Garlan et al., 1997]:

1. **Un núcleo curricular:** Formado por los cursos sobre los que recaen los fundamentos de Ingeniería del Software, haciendo un énfasis especial en el análisis, diseño y gestión de grandes sistemas software. Los cursos que conforman este currículo son:

a. Modelos de sistemas software.

b. Métodos de desarrollo de software.

c. Gestión del desarrollo del software.

d. Análisis de elementos software.

e. Arquitecturas de sistemas software.

2. **Desarrollo de un proyecto:** Durante la duración del *master*, los alumnos planifican e implementan un proyecto software de tamaño significativo para un cliente externo. El trabajo se hace en equipo supervisado por profesores.

3. **Cursos de especialización:** De carácter optativo, que permiten que los estudiantes desarrollen una experiencia más profunda en una de las siguientes especialidades, entre las que se incluyen *sistemas de tiempo real, interfaces hombre-máquina* y *mejora del proceso software*.

Programas de graduación

En el SEI se llega a la conclusión de que la incesante demanda de ingenieros del software no se puede cubrir sólo con los estudiantes de postgrado, lo que hace necesaria la creación de programas de graduación en Ingeniería del Software. Estos programas quedan documentados en [Ford, 1990; Ford, 1991b; Ford, 1994].

El esquema de este currículo es:

1. **Matemáticas y Ciencia.** Con los objetivos de preparar a los alumnos para participar en una sociedad cada día más tecnológica, así como de ofrecerles los fundamentos necesarios para afrontar el resto de las asignaturas de la titulación. Se recomiendan dos asignaturas de matemáticas discretas, una de estadística y probabilidad, dos de cálculo, una de métodos numéricos, una de física, una de química y una de biología.

2. **Ciencia de Ingeniería y Diseño de Ingeniería.** Con los siguientes cursos:

   a. **Análisis.** Ofrece al alumno el conocimiento para realizar modelos y razonar sobre el proceso software. Algunos de los temas a tratar son: *desarrollo formal de algoritmos y programas (incluyendo la verificación formal de los mismos); técnicas de abstracción y modelado; sistemas formales y su aplicación a la Ingeniería del Software; métricas, análisis de algoritmos; análisis de rendimiento...*

b. **Arquitecturas software.** Abordan la solución de problemas recurrentes a un alto nivel. Algunos temas pueden ser: *representación de datos, información y conocimiento; gestión de recursos; sistemas expertos; sistemas de tiempo real embebidos; sistemas concurrentes, paralelos o distribuidos...*

c. **Hardware.** En la Ingeniería del Software no todo es software, debiendo un ingeniero del software comprender el hardware que está presente en los sistemas informáticos. Se tratan temas relacionados con *lenguajes ensambladores; arquitectura de computadores; sistemas digitales; redes...*

d. **Proceso software.** Se encarga de transmitir el conjunto de herramientas, métodos y prácticas que se utilizan en la creación de software. Temas de este curso son: *análisis de requisitos; especificación y métodos formales; diseño; técnicas de implementación y lenguajes; verificación y validación; evolución del software; evaluación de productos y procesos software; gestión y organización de equipos de trabajo; temas éticos y profesionales.*

3. **Humanidades, Ciencias Sociales y Optativas.** Para completar la formación del alumno.

4.4.3.2 La propuesta del WGSEET

Cuando se presentaron los cuerpos de conocimiento sobre Ingeniería del Software, ya se comentó la iniciativa del WGSEET para la definición de un modelo de currículo para esta disciplina, que se recoge en [Bagert et al., 1999].

Esta propuesta curricular se centra en la generación de nuevos graduados que tengan en la Ingeniería del Software la base de conocimientos para afrontar una vida profesional condicionada por las necesidades de su entorno industrial y empresarial.

Arquitectura del currículo

En el diseño de este currículo intervienen una serie de elementos básicos, como se puede apreciar en la Figura 4.9.

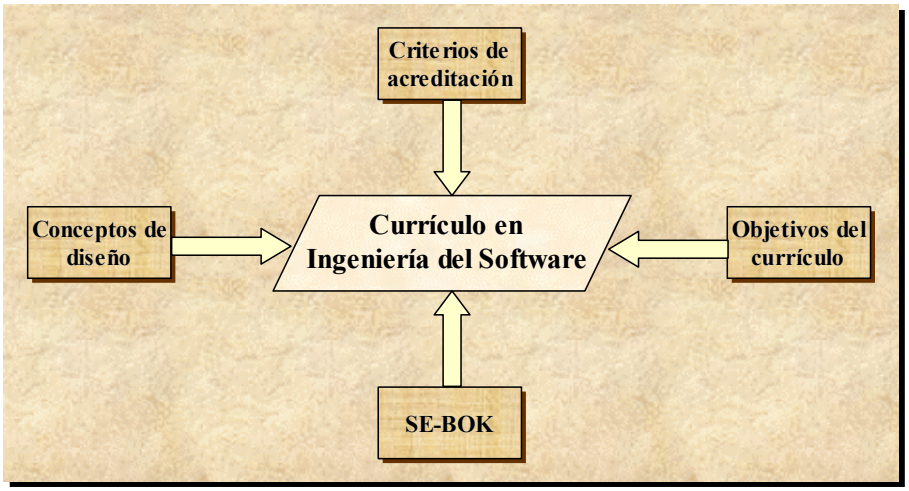

Figura 4.9. Arquitectura del currículo en Ingeniería del Software del WGSEET [Bagert et al., 1999]

La determinación de los objetivos del currículo se convierte en una actividad prioritaria en el diseño del mismo. Conlleva consideraciones sobre la misión de la institución que lo va a implantar y sobre la estrategia que se quiere seguir. En un nivel más fundamental, todos los afectados por la definición del currículo (profesores, alumnos, empresarios...) debieran estar representados en el momento de establecer estos objetivos.

El cuerpo de conocimientos ofrece la base de fundamentos para fijar el contenido del currículo, mientras que los criterios de acreditación introducen las bases para asegurar la calidad y la evaluación del mismo.

Por último, los conceptos de diseño construyen un marco filosófico para el desarrollo de currículos efectivos en Ingeniería del Software, así como para la definición del modelo de currículo del WGSEET.

Conceptos de diseño

La lista de conceptos que aparecen en esta sección constituye los principios básicos para el modelo de currículo en Ingeniería del Software propuesto por el WGSEET.

1. Soporta el desarrollo de varios programas de estudios, con la característica común de centrarse en la Ingeniería del Software (Ingeniería del Software, Ciencia de la Computación, Sistemas de Información...).
2. Introduce a los alumnos, desde el comienzo de sus estudios, en la naturaleza, ámbito e importancia de la Ingeniería del Software.
3. El modelo incorpora dos niveles de educación en Ingeniería del Software:

a. La denominada Ingeniería del Software a pequeña escala (*Software Engineering in the Small*)

   i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados de forma individual o en pequeños grupos.

   ii. El desarrollo del software con estas características se encuentra concentrado en los primeros cursos.

b. La denominada Ingeniería del Software a gran escala (*Software Engineering in the Large*)

   i. Aplica los principios de la Ingeniería del Software en el desarrollo de productos software realizados en equipo.

   ii. El desarrollo del software con estas características se encuentra localizado en cursos avanzados o en pequeños proyectos realizados para empresas.

4. El modelo establece un balance entre *producto* y *proceso*.

   a. Las actividades relacionadas con el producto incluyen métodos, técnicas y propiedades usados en la construcción de los elementos software asociados en un producto software (planes de desarrollo, planes de aseguramientos de la calidad, especificación de requisitos, especificación de diseño, código, documentación...).

   b. Las actividades relacionadas con el proceso incluyen los estándares, procedimientos, guías, métricas y técnicas de análisis y soporte que ofrecen el marco adecuado para llevar a cabo las actividades de construcción del producto de forma efectiva y eficiente.

5. El modelo ofrece una guía para el desarrollo de programas de graduación que pueden ser acreditados por organizaciones externas.

6. Se incluyen asuntos éticos, sociales y profesionales que estén directamente relacionados con la práctica profesional de la Ingeniería del Software.

7. El modelo enfatiza el concepto de trabajo en grupo.

8. Ofrece especialización en dominios de aplicación concretos (sistemas empotrados, bases de datos y Sistemas de Información, sistemas inteligentes, redes...).

9. Asegura la práctica de la Ingeniería del Software:
    a. Prácticas en laboratorios planificadas.
    b. Proyectos de Ingeniería del Software a realizar en laboratorio.
    c. Educación cooperativa y prácticas en empresas.

Contenidos del currículo

Aunque la organización y puesta en marcha de varios currículos centrados en la Ingeniería del Software puede diferir de unos a otros, todos ellos deben ofrecer asignaturas que incluyan conocimientos sobre Matemáticas, Fundamentos de Ciencia de la Computación, Ingeniería del Software, Educación General, Ciencias Naturales y Conocimientos de Dominios Específicos.

En la siguiente lista se describe lo que, como mínimo, el WGSEET considera que se debe incluir en cada área:

1. **Fundamentos de Ciencia de la Computación**
    a. Programación, estructuras de datos y algoritmos.
    b. Conceptos de lenguajes de programación.
    c. Organización de computadores.
    d. Sistemas software (gestión de procesos y recursos, computación concurrente y distribuida, redes).
2. **Matemáticas**
    a. Matemática discreta.
    b. Estadística y probabilidad.
3. **Ciencias naturales**
4. **Educación general**
    a. Comunicación (oral y escrita).
    b. Ciencias sociales y humanidades.
5. **Ingeniería del Software**
    a. Ingeniería de requisitos.
    b. Diseño del software.
    c. Calidad del software.
    d. Arquitecturas software.
    e. Construcción del software.

f. Evolución del software.

g. Métodos formales.

h. Interfaces hombre-máquina.

i. Organización, planificación de proyectos.

j. Proceso software.

k. Ética y profesionalidad en la Ingeniería del Software.

Módulos propios de la Ingeniería del Software

Los módulos que se presentan en la Tabla 4.17, pueden hacerse corresponder cada uno de ellos con una sola asignatura, o combinarse varios de ellos en una asignatura.

Módulo	Título	Descripción
IS1	*Introducción a la Ciencia de la Informática para ingenieros del software 1*	Introducción a la programación, normalmente en el paradigma procedimental. Las características básicas de la Ingeniería del Software se integran en este curso.
IS2	*Introducción a la Ciencia de la Informática para ingenieros del software 2*	Introducción a las estructuras de datos y al paradigma objetual. Se continúa con la introducción de conceptos de Ingeniería del Software.
IS3	*Introducción a la Ingeniería del Software*	Descripción general de la Ingeniería del Software como disciplina; introduce los principios fundamentales y las metodologías.
IS4	*Ética y profesionalismo*	Cubre material sobre aspectos históricos, sociales y económicos de la Ingeniería del Software. Incluye el estudio de las responsabilidades y riesgos profesionales, así como sobre la propiedad intelectual.
IS5	*Requisitos del software*	Introduce los conceptos básicos y principios de la ingeniería de requisitos: sus herramientas, técnicas y métodos para el modelado del software.
IS6	*Diseño del software*	Métodos y técnicas utilizados en la fase de diseño. Énfasis en el diseño orientado a objetos.
IS7	*Calidad del software*	Aseguramiento de la calidad y gestión de la configuración.
IS8	*Construcción y evolución del software*	Examina problemas, métodos y técnicas asociadas con la construcción del software, dado un diseño de alto nivel y teniendo en cuenta el mantenimiento del mismo.
IS9	*Proyecto*	Permite a los estudiantes poner en práctica los conocimientos adquiridos en el resto de módulos, al entrar a formar parte de un equipo de desarrollo que se encarga de la realización de un proyecto.

Tabla 4.17. Módulos relacionados con la Ingeniería del Software en el currículo WGSEET

Para una descripción más detallada de cada uno de estos módulos, incluyendo un índice de primer nivel de los contenidos de cada uno de ellos, se recomienda la consulta de [Bagert et al., 1999].

Influencias de otras propuestas curriculares

La propuesta curricular del WGSEET recibe influencias de otras propuestas anteriores, que también coinciden en el objetivo de establecer un currículo independiente para la disciplina de Ingeniería del Software. De todas ellas se van presentar las dos más relevantes.

Thomas B. Hilburn propone un modelo conceptual de currículo en Ingeniería del Software [Hilburn, 1997] que se basa en tres pilares fundamentales:

- ***Las personas:*** Un currículo debe soportar las actividades que formen al alumno para trabajar y comunicarse con sus semejantes de una forma efectiva. Las capacidades a considerar son: *educación general, capacidad de comunicación, trabajo en grupo* así como *ética y profesionalismo.*
- ***Proceso:*** Los estudiantes deben darse cuenta de la necesidad de utilizar un proceso definido para desarrollar productos software, distinguiendo el proceso para la creación de software de carácter individual, el proceso para trabajar en equipo y el proceso que involucra a los estándares seguidos por una organización.
- ***Tecnología:*** Los alumnos deben adquirir los conocimientos tecnológicos y científicos necesarios para el desarrollo de software de calidad, incluyendo conocimientos de *Matemáticas y Ciencia, Ciencia de la Computación* y *Desarrollo de Software*.

A. J. Cowling propone un modelo de currículo en Ingeniería del Software multi-dimensional [Cowling, 1998], que establece las siguientes dimensiones:

- **Los diferentes niveles de abstracciones que definen los componentes hardware y software**. Establece que el ámbito de la Ingeniería del Software está en la construcción de sistemas en los que su parte principal está constituida por componentes software. Permite distinguir a los ingenieros del software de aquéllos que trabajan en la parte de arquitecturas de computadores.
- **El balance de los contenidos en Informática con respecto a otras ramas de la Ciencia y la Ingeniería**. Permite distinguir entre la Ingeniería del Software y otras disciplinas.

- **El balance entre la teoría, el modelado y su aplicación práctica**. La base de esta tercera dimensión es la observación de que la Ingeniería del Software no es la mera construcción de sistemas software, sino que la construcción de éstos se hace acorde a un método de Ingeniería.
- **El balance entre la parte técnica y la parte no técnica**. Las tres primeras dimensiones se centran en la parte técnica, pero una característica importante de la Ingeniería es que consiste en algo más que un conjunto de aspectos técnicos. Se necesitan contemplar aspectos económicos, sociales y sicológicos.

4.5 La Ingeniería del Software en otros Planes de Estudios españoles

En [Molina, 2000] se realiza un pequeño estudio sobre los currículos de Ingeniería del Software en 11 universidades españolas. Se advierte que, aunque los contenidos globales de las titulaciones aplican las propuestas de los currículos internacionales (y por supuesto las directrices de los Planes de Estudio oficiales) la organización en asignaturas concretas es muy variada, probablemente debido a razones históricas y a una introducción progresiva de la disciplina en las titulaciones. Los temas más frecuentemente abordados son los referidos a aspectos generales como *Análisis y diseño del Software*, *Ciclo de vida del Software*, *Planificación*, *Pruebas y Validación*... Por otro lado, los temas menos corrientes son los relacionados con *Aspectos legales* (sólo 10%), *Especificación formal*, *Reingeniería* o *Componentes* (todos ellos con un 20%).

Siguiendo la misma estela, en este apartado se presenta un análisis actualizado y no exhaustivo sobre la inclusión de materias relacionadas con Ingeniería del Software en Facultades y Escuelas de Informática de varias Universidades españolas. El estudio se ha dividido en dos partes, correspondientes a las asignaturas de las titulaciones técnicas y superior. Aunque todavía existen Universidades que mantienen diplomaturas o licenciaturas con Planes a extinguir, sólo se han considerado titulaciones vigentes y temarios reales de las asignaturas.

En el caso de las Ingenierías Técnicas se han incluido en el estudio las asignaturas directamente relacionadas con la Ingeniería del Software. En la especialidad de Gestión (I.T.I.G.), esta materia es troncal, al contrario de la especialidad de Sistemas (I.T.I.S.). En este segundo caso se han recogido las asignaturas obligatorias u optativas correspondientes cuando ha sido posible.

Respecto a la titulación en Ingeniería en Informática (I.I.), además de las Universidades con mayor tradición en Informática, se han analizado específicamente situaciones similares a la que se da en la Universidad de Salamanca (una titulación superior sólo de segundo ciclo). En cualquier caso, además de las asignaturas troncales de la materia se han incluido aquellas optativas con relación directa con la Ingeniería del Software.

Las universidades estudiadas han sido las siguientes: Burgos, Carlos III de Madrid, Complutense de Madrid, Málaga, Oviedo, Politécnica de Cataluña, Politécnica de Madrid, Politécnica de Valencia, Sevilla y Valladolid. Se han tenido en cuenta las distintas situaciones en especial en la titulación superior (estudios de primer y segundo ciclos frente a estudios de sólo segundo ciclo).

Por último, hay que tener en cuenta que en la mayoría de las Universidades españolas, los Planes de Estudios y el contenido concreto de cada materia de las titulaciones de Informática están en constante actualización, por lo que es posible que se hayan producido modificaciones desde el momento en que se recogió esta información (enero y febrero de 2002, vía web, en general). La información recogida se resume en los siguientes apartados.

4.5.1 Titulaciones de primer ciclo

Universidad de Burgos

http://www.ubu.es/.

Asignatura: Análisis e Ingeniería del Software.

Titulación: ITIG.

Créditos: 12.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software. Ciclo de vida del proyecto. Paradigma estructurado: análisis y diseño. Paradigma objetual: análisis y diseño, introducción a los patrones de diseño. Gestión de proyectos.

Universidad Carlos III de Madrid

http://www.uc3m.es/.

Asignatura: Ingeniería del Software 1.

Titulación: ITIG.

Créditos: 7.

Carácter: Troncal.

Contenidos: Introducción la ingeniería del Software. Técnicas de análisis de aplicaciones de gestión. Técnicas de especificación del funcionamiento de la aplicación. Técnicas de diseño de aplicaciones de gestión.

Asignatura: Ingeniería del Software 2.

Titulación: ITIG.

Créditos: 7.

Carácter: Troncal.

Contenidos: Introducción. Procesos de gestión de proyectos software. Gestión de la configuración. Estimación de proyectos software. Organización de proyectos software. Planificación de proyectos software. El Plan de desarrollo software. Seguimiento y control de proyectos software. Gestión y Control de calidad.

Universidad Complutense de Madrid

http://www.ucm.es/.

Asignatura: Ingeniería del Software de Gestión 1. Ingeniería del Software 1.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal en ITIG, optativa en ITIS.

Contenidos: Conceptos básicos. Gestión, estimación y planificación de proyectos. Gestión de configuraciones. Ingeniería del Software orientada a objetos.

Asignatura: Ingeniería del Software de Gestión 2. Ingeniería del Software 2.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal en ITIG, optativa en ITIS.

Contenidos: El modelo de objetos. Clases y objetos. El Lenguaje Unificado de Modelado. Notación. Modelo de Proceso. El proceso de desarrollo de software unificado. Patrones de diseño orientados a objetos. Pruebas y métricas orientadas a objetos.

Universidad de Málaga

http://www.uma.es/.

Asignatura: Ingeniería del Software de Gestión.

Titulación: ITIG.

Créditos: 12.

Carácter: Troncal.

Contenidos: Introducción y estrategias para el desarrollo de sistemas de información. Metodologías de desarrollo. Gestión de proyectos. Métrica. Planificación de sistemas. Análisis, diseño e implementación de sistemas de información.

Asignatura: Ingeniería del Software.

Titulación: ITIS.

Créditos: 6.

Carácter: Optativa.

Contenidos: Introducción al Proceso Unificado. Desarrollo del Software. Captura de Requisitos. Análisis, diseño e implementación. Prueba. Métrica 3.

Universidad Politécnica de Cataluña

http://www.upc.es/.

Asignatura: Ingeniería del Software I. Especificación.

Créditos: 6.

Carácter: Obligatoria en ITIG y Optativa en ITIS.

Contenidos: Paradigmas en Ingeniería del Software. Requisitos y especificaciones. Especificación orientada a objetos: la notación UML. Especificación de sistemas de tiempo real en UML. Pruebas. Otros métodos de especificación.

Asignatura: Ingeniería del Software. Diseño I.

Créditos: 6.

Carácter: Obligatoria en ITIG y Optativa en ITIS.

Contenidos: Introducción al diseño. Diseño orientado a objetos. Pruebas del software.

Universidad Politécnica de Madrid

http://www.upm.es/.

Asignatura: Ingeniería del Software.

Titulación: ITIG/ITIS.

Créditos: 9.

Carácter: Troncal en ITIG y Obligatoria en ITIS.

Contenidos: Proceso y gestión del software. Análisis de requisitos del sistema y del software. Diseño e implantación del software. Prueba, mantenimiento y documentación del software. Automatización del software.

Asignatura: Metodologías de Desarrollo.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción. Análisis y determinación de requisitos. Análisis estructurado. Diseño de sistemas. Pruebas de software. Implantación del sistema. Interfaces de métrica v.3.

Universidad Politécnica de Valencia

http://www.upv.es/.

Asignatura: Ingeniería del Software.

Titulación: ITIG/ITIS.

Créditos: 6.

Carácter: Troncal ITIG, Optativa ITIS.

Contenidos: Introducción. Paradigmas de ciclo de vida. Planificación y gestión de proyectos. Desarrollo de software orientado a objetos. Técnicas de prueba de programas.

Asignatura: Laboratorio de Ingeniería del Software.

Titulación: ITIG.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Análisis de aplicaciones de gestión. Construcción de proyectos software. Uso de herramientas CASE.

Universidad de Sevilla

http://www.us.es/.

Asignatura: Ingeniería del Software de Gestión I.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería de Software. Análisis de requisitos. Metodología. Especificación estructurada de sistemas software. Especificación orientada a objetos de sistemas software.

Asignatura: Ingeniería del Software de Gestión II.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción. Diseño. Diseño estructurado. Diseño OO. Diseño de planes de pruebas. La visión del diseño en Métrica.

Asignatura: Ingeniería del Software de Gestión III.

Titulación: ITIG.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Fundamentos de programación, diseño de interfaz de usuario. Arquitecturas de sistemas. Acceso a datos. Programación OO. Mapeo de modelo de objetos a modelo de datos relacional. Bases de datos en Internet. Sistemas de soporte a la toma de decisiones.

Universidad de Valladolid

http://www.infor.uva.es/.

Asignatura: Ingeniería del Software I.

Titulación: ITIG.

Créditos: 7,5.

Carácter: Troncal.

Contenidos: Conceptos básicos. Desarrollo del software. Análisis de necesidades y estudio de viabilidad. Diseño. Diseño de la interfaz. Diseño del tratamiento. Las etapas del método.

Asignatura: Ingeniería del Software II.

Titulación: ITIG.

Créditos: 6.

Carácter: Troncal.

Contenidos: Métodos y técnicas de análisis y diseño orientado a objetos.

Asignatura: Ingeniería del Software I.

Titulación: ITIS.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Conceptos básicos. Principios de la Ingeniería del Software. Modelado de análisis. Modelado de la implementación.

Asignatura: Ingeniería del Software II.

Titulación: ITIS.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Conceptos básicos de la orientación a objetos. Especificaciones orientadas a objetos. Diseño de sistemas interactivos.

4.5.2 Titulaciones de Segundo ciclo

Universidad de Burgos

http://www.ubu.es/.

Asignatura: Planificación y Gestión de Proyectos.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Análisis y definición de requisitos. Gestión de configuraciones. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones.

Asignatura: Diseño y Mantenimiento del Software I.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Diseño, propiedades y mantenimiento de software. Análisis de aplicaciones.

Asignatura: Diseño y Mantenimiento del Software II.

Titulación: II (segundo ciclo).

Créditos: 6.

Carácter: Troncal.

Contenidos: Diseño, propiedades y mantenimiento de software. Análisis de aplicaciones.

Universidad Carlos III de Madrid

http://www.uc3m.es/.

Asignatura: Procesos Software I.

Titulación: II (segundo ciclo).

Créditos: 7.

Carácter: Troncal.

Contenidos: El proceso de desarrollo de software. La orientación a objetos. UML: el lenguaje unificado de modelado. Modelado estructural. Modelado dinámico. Modelado arquitectónico.

Asignatura: Procesos Software II.

Titulación: II (segundo ciclo).

Créditos: 5.

Carácter: Troncal.

Contenidos: La Ingeniería del negocio: Perspectivas. Patrones de Diseño. Metodologías Específicas para la Reutilización. Arquitecturas basadas en Componentes. Diseño de componentes. Modelos de Madurez en Reutilización.

Universidad Complutense de Madrid

http://www.ucm.es/.

Asignatura: Ingeniería del Software.

Titulación: II.

Créditos: 18.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería de Software. El proceso de desarrollo de software. Planificación y gestión de Proyectos. Análisis y Especificación de Requisitos. Diseño de Software. Ingeniería del software en sistemas distribuidos. Análisis de programas. La medida del Software. Mantenimiento de aplicaciones. Reingeniería de Software.

Universidad de Málaga

http://www.uma.es/.

Asignatura: Ingeniería del Software-Análisis.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Sistemas software complejos. Tecnologías de Objetos. El Lenguaje de Modelado Unificado. Proceso Unificado. Especificación de Requisitos. Análisis.

Asignatura: Ingeniería del Software-Diseño.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: El modelo de objetos. Patrones de diseño. Diseño orientado a objetos con UML. Lenguajes y aplicaciones.

Asignatura: Ingeniería del Software-Proyectos.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Gestión de Proyectos. Herramientas. Proyecto.

Universidad de Oviedo

http://www.uniovi.es/.

Asignatura: Ingeniería del Software I.

Titulación: II (segundo ciclo).

Créditos: 18.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software. Metodología estándar en la Administración: Métrica-2. Análisis de Sistemas según Métrica-2. Métodos de diseño. Diseño estructurado. Técnicas y estrategias de prueba del software. Métodos orientados a objetos. Introducción a la Planificación y la estimación. Documentación, Estándares y Herramientas CASE.

Asignatura: Ingeniería del Software II.

Titulación: II (segundo ciclo).

Créditos: 18.

Carácter: Optativa.

Contenidos: Métodos Estructurados para desarrollo de sistemas en Tiempo Real. Aseguramiento de la Calidad del Software. Funciones SQA. El Plan General de Calidad. Auditoría informática. Planificación, estimación y métricas. La Ingeniería del Software desde el punto de vista de la empresa.

Universidad Politécnica de Cataluña

http://www.upc.es/.

En el Plan de II, también aparecen las asignaturas reflejadas en el primer ciclo. Además aparece una segunda asignatura dedicada al diseño y, como optativa, una asignatura enfocada específicamente a los métodos formales.

Asignatura: Ingeniería del Software. Diseño II.

Titulación: II.

Créditos: 6.

Carácter: Obligatoria.

Contenidos: Arquitectura del software. Plataformas de sistemas distribuidos. CORBA. Diseño orientado a objetos de sistemas de información.

Asignatura: Métodos Formales en Ingeniería del Software.

Créditos: 4,5.

Carácter: Optativa.

Contenidos: Métodos formales orientados a modelos (Z, VDM). Métodos formales axiomáticos (Larch, OBJ), demostradores automáticos: el demostrador de Larch.

Universidad Politécnica de Madrid

http://www.upm.es/.

Asignatura: Ingeniería del Software I.

Créditos: 9.

Carácter: Troncal.

Contenidos: Introducción al proceso de IS. Factores Humanos. Gestión de Configuración. Aseguramiento de la calidad. Gestión de proyectos. Evaluación de Procesos. Adquisición del Software.

Asignatura: Ingeniería del Software II.

Créditos: 12.

Carácter: Troncal.

Contenidos: Ingeniería de Requisitos. Aproximación de Desarrollo Estructurado. Aproximación al Diseño Orientado a Objetos.

Asignatura: Profundización en Ingeniería del Software.

Créditos: 6.

Carácter: Optativa.

Contenidos: Seminarios sobre diversos temas como evaluación del proceso, métricas, ingeniería de usabilidad.

Asignatura: Aspectos profesionales de la Ingeniería del Software.

Créditos: 4,5.

Carácter: Libre elección.

Universidad Politécnica de Valencia

http://www.upv.es/.

A pesar de que el nombre que recibe, las asignaturas de Ingeniería de la Programación e Ingeniería de Requerimientos encajan con el perfil de Ingeniería del Software. También se ofrece una asignatura optativa sobre métodos formales.

Asignatura: Ingeniería de la Programación.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Análisis y diseño orientados a objetos. Garantía de calidad del software (se trata de mostrar una metodología de ciclo de vida completo, como por ejemplo OMT; el tema de calidad se orientará hacia métricas).

Asignatura: Laboratorio de Ingeniería de la Programación.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Nuevas tecnologías para el desarrollo de software. Programación por componentes. Programación cliente-servidor, etc. Prácticas en Borland Delphi (o similar).

Asignatura: Ingeniería de Requerimientos.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Estabilización de Requisitos. Validación de Requisitos. Modelos de tiempo real. Dinámica. Modelos del tiempo. Lógicas modales dinámicas. Herramientas y metodologías asociadas.

Asignatura: Métodos Formales de la Ingeniería del Software.

Titulación: II.

Créditos: 6.

Carácter: Optativa.

Contenidos: Necesidad del formalismo en Ingeniería del Software. Prototipado automático. Especificación lógica del Software. Especificación algebraica del software. Especificación orientada a objetos. Técnicas mixtas. Demostración formal de propiedades.

Universidad de Sevilla

http://www.us.es/.

Asignatura: Ingeniería del Software I.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la Ingeniería del Software y a la orientación a objetos. Elicitación de requisitos de sistemas software. Modelos orientados a objetos de un sistema software. Especificación orientada a objetos de sistemas software. Metodología.

Asignatura: Ingeniería del Software II.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Conceptos básicos y fundamentales. Separación de conceptos, diseño modular, abstracción y diseño genérico. Diseño Arquitectónico basado en Patrones. Tecnologías de Diseño.

Asignatura: Ingeniería del Software III.

Titulación: II.

Créditos: 6.

Carácter: Troncal.

Contenidos: Introducción a la gestión y dirección de Proyectos. Métricas del software. Métodos tradicionales de planificación y estimación. Gestión de la calidad del software. Gestión y control del riesgo. Herramientas de ayuda a la gestión.

Asignatura: Métodos Formales en la Ingeniería del Software.

Titulación: II.

Créditos: 6.

Carácter: Optativa.

Contenidos: Lenguajes de especificación. Especificación basada en UML y OCL. Una metodología de especificación basada en métodos formales. Especificación de tipos abstractos de datos, componentes, sistemas. Técnicas de diseño basadas en métodos formales, patrones de diseño. Diseño de los aspectos de concurrencia. Diseño de los aspectos de persistencia. Diseño de los aspectos específicos de las aplicaciones distribuidas.

Universidad de Valladolid

http://www.infor.uva.es/.

Asignatura: Ingeniería del Software I.

Titulación: II (segundo ciclo).

Créditos: 9.

Carácter: Troncal.

Contenidos: Modelos de ciclo de vida. Ingeniería de requisitos. UML. Métodos formales.

Asignatura: Ingeniería del Software II.

Titulación: II (segundo ciclo).

Créditos: 9.

Carácter: Troncal.

Contenidos: Diseño de software. Arquitectura de software. Métricas. Gestión de proyectos.

4.6 Resultados del análisis. Conclusiones

Desde el punto de vista de las recomendaciones internacionales, realizadas en países con diferentes estructuras universitarias y profesionales, la aplicación directa de las mismas no es realizable sin tener en cuenta la realidad universitaria y el entorno social y profesional de nuestro país. No obstante, el reconocimiento internacional de las mismas así como la autoridad científica y profesional de sus autores las convierten en valiosas guías a considerar a la hora de elaborar una propuesta docente específica.

Todas ellas coinciden en la importancia de perseguir tanto la obtención de conocimientos como de capacidades y habilidades por parte del alumno.

El *Computing Curricula 2001* [ACM/IEEE-CS, 2001], aunque no se especializa en Ingeniería del Software, aumenta considerablemente los contenidos sobre la materia respecto a la propuesta de 1991. Sin entrar en demasiadas profundidades, proporciona una guía sobre los contenidos básicos de la disciplina en el contexto de las titulaciones generalistas de Informática.

Por otro lado, las propuestas del SEI (e incluso los currículos propuestos para titulaciones de Sistemas de Información, en el caso de ITIG) aportan matices que pueden ayudar a definir una visión global de la Ingeniería del Software. De estas propuestas, el informe del SEI [Ford, 1991a], aunque un poco antiguo y dirigido básicamente a una titulación de postgrado, es bastante completo y detallado, incluyendo numerosas referencias bibliográficas e incluso material audiovisual de soporte.

Por último, la publicación del SWEBOK [Abran et al., 2001c] representa poner a nuestra disposición un cuerpo organizado de conocimientos sobre la Ingeniería del Software que puede jugar un importante papel a la hora de localizar referencias sobre temas concretos.

Analizando los contenidos de las distintas propuestas, se pueden agrupar, a grandes rasgos, en dos grandes bloques que se corresponden con la doble visión producto/proceso de la Ingeniería del Software. El primer bloque se puede organizar siguiendo las fases del ciclo de

vida del software. Siguiendo el esquema del SWEBOK y del *Computing Curricula 2001* (entre paréntesis, la clave correspondiente a las unidades del *Computing Curricula*):

- Requisitos del software (SE5).
- Diseño del software (SE1).
- Construcción del software (SE2, además de PF y PL).
- Prueba del software (SE6).
- Mantenimiento y evolución del software (parcialmente SE7).

El segundo bloque está constituido por las materias relacionadas con el Proceso Software y su gestión, en general:

- Gestión de la configuración del software (SE8).
- Gestión de la Ingeniería del software (SE8).
- Proceso de Ingeniería del software (SE4).
- Calidad del software (SE8).

Se puede dejar fuera, puesto que es transversal, la unidad de conocimiento Herramientas y métodos de la Ingeniería del Software (equivalente aproximadamente a la unidad SE3 del *Computing Curricula 2001*).

En cuanto a la importancia relativa de cada bloque, en la asignación de horas del *Computing Curricula 2001* al primer bloque se le dedican 23 de las 31 horas propuestas como fundamentales (*core*), o lo que es lo mismo, el 74%, en tanto que para el Proceso y las herramientas se deja el 26% restante. Evidentemente se trata de un enfoque dirigido a alumnos de cursos de graduación y la mayoría del tiempo se dedica al estudio de conceptos y técnicas fundamentales. En los programas de postgrado, del tipo del SEI, los porcentajes varían considerablemente. En otro orden de cosas, en el *Computing Curricula* la presencia de métodos formales se limita a aspectos básicos en la unidad SE5 (Requisitos software y especificaciones), pero la unidad opcional SE10 (Métodos formales) está dedicada expresamente a ellos.

Por último, no hay que olvidar un componente importantísimo de los sistemas modernos, la interfaz persona-ordenador. Tanto el SWEBOK como el *Computing Curricula* la consideran un área independiente de la Ingeniería del Software. Sin embargo, al no estar incluida en las directrices de los Planes de Estudio españoles, sus contenidos básicos se deben incluir en las asignaturas de Ingeniería del Software, con independencia de la necesidad de asignaturas optativas sobre el tema.

Sobre la presencia de la Ingeniería del Software en las Universidades españolas, se observa que en muchas de ellas se aumenta la troncalidad recomendada y se añaden asignaturas obligatorias u optativas sobre la disciplina, apareciendo también en la Ingeniería Técnica de Informática de Sistemas, donde no es troncal. En relación con los contenidos propuestos se observa cómo se intenta conjugar la adaptación de los currículos académicos a la realidad social, con el interés en mantener el rigor y el carácter de "persistencia" de los conceptos relacionados con esta disciplina. Éste es un objetivo compatible con la adquisición de conocimientos y destrezas concretas mediante la realización de prácticas y desarrollo de proyectos, contribuyendo así a alcanzar una formación integral del estudiante.

En el primer ciclo, se aprecia una fuerte presencia de lo que se puede llamar Ingeniería del Software convencional, básicamente, métodos estructurados y orientados a objetos, con una explosión reciente de temas dedicados al Lenguaje Unificado de Modelado (UML). La gestión de los proyectos software ocupa el segundo lugar entre los temas abordados con más frecuencia. A la hora de repartir la disciplina en dos o tres asignaturas cuatrimestrales, las diferencias entre los temarios estudiados se limitan, en lo fundamental, a pocas variantes: división basada en el ciclo de vida, división entre métodos estructurados y orientados a objeto, división entre temas de gestión y metodológicos.

En el segundo ciclo la situación es mucho más compleja. El punto de partida común es la presencia de al menos 18 créditos troncales, que en algunos casos se aumenta y en otros se complementa con asignaturas optativas. Se pueden detectar dos situaciones diferentes, según el tipo de organización de los estudios. La mayoría de las titulaciones superiores son "completas", de modo que el alumno llega al segundo ciclo sin haber estudiado previamente la disciplina. El segundo caso corresponde a algunas titulaciones de sólo segundo ciclo, enfocadas a alumnos con titulación previa en alguna de las Ingenierías Técnicas y por tanto con conocimientos previos sobre Ingeniería del Software. Sin embargo, a pesar de la diferencia, en el segundo ciclo predomina la formación desde cero en Ingeniería del Software, aunque con mayor profundidad. Sin embargo, y como se expone en [Canos et al., 1998], en el diseño de los currículos sería conveniente evitar repeticiones en el contenido en los dos ciclos así como distinguir entre la orientación del futuro ingeniero en Informática (más cercano a tareas de Planificación, Gestión de proyectos, Análisis y Especificación de Requisitos) y los ingenieros técnicos (más orientados al trabajo en las fases clásicas del desarrollo propiamente dicho).

En general se detecta mayor presencia de temas avanzados: Patrones de diseño, Desarrollo basado en componentes, Reutilización, Reingeniería de Software, Sistemas distribuidos, Aseguramiento de la calidad... Otros temas, como Prototipado, Especificaciones formales,

Demostración de propiedades, aparecen en la mayoría de los casos en asignaturas optativas o no aparecen en absoluto.

Llama la atención la coincidencia de casi todas las propuestas internacionales en cuanto a la consideración de los aspectos éticos, sociales y profesionales, y la poca importancia que a estos temas se le da en las Universidades españolas; esta situación está cambiando tímidamente, mediante la introducción de asignaturas optativas sobre dichos aspectos.

A modo de resumen, se puede concluir que un programa completo de formación en Ingeniería del Software ha de tener en cuenta los siguientes puntos:

- La formación específica en Ingeniería del Software ha de estar relacionada con la formación en algoritmos, programación, arquitectura de ordenadores, bases de datos, sistemas operativos o redes de ordenadores dentro de un marco general de formación en Informática.
- El programa ha de contemplar todos los aspectos del desarrollo, cubriendo todos los aspectos relacionados con el ciclo de vida del software, incluyendo el funcionamiento en un entorno real. Por lo dicho en el punto anterior, algunas fases de desarrollo (construcción y prueba del software o diseño de bases de datos) se tratan en otras disciplinas y hay que tener ese hecho en cuenta a la hora de diseñar y coordinar los temarios.
- En el programa de Ingeniería del Software han de incluirse además las áreas de gestión de procesos software, selección y utilización de componentes y herramientas software, garantía de calidad, interacción persona-ordenador, documentación y estándares. Sería deseable incluir aspectos complementarios como la formación en los aspectos sociales, éticos, legales, económicos, trabajo en grupo y técnicas de comunicación oral y escrita.
- El desarrollo de este programa necesita la existencia de recursos de laboratorio y herramientas que permitan la realización de prácticas y proyectos. Es esencial el acceso por parte de los alumnos y los profesores a herramientas que soporten el ciclo de vida del software.
- Es necesaria la existencia de material bibliográfico sobre todo en lo que se refiere a fuentes primarias (revistas) dada la continua y rápida evolución de los principios, métodos y técnicas de la Ingeniería del Software, con el fin de mantener una actualización permanente.

- Por último, hay que tener en cuenta que la formación en Ingeniería del Software ha de integrar requisitos técnicos con requisitos generales de educación para preparar a los estudiantes para una carrera profesional en este campo. Esta formación ha de permitirles adaptarse a los desarrollos tecnológicos de forma efectiva, rápida y con el menor coste posible a través de seminarios de adaptación impartidos desde el entorno académico (cursos de postgraduado) o desde el entorno industrial (seminarios de formación continuada).

El próximo capítulo se dedica a presentar la propuesta concreta que, siguiendo estos objetivos, el candidato se propone llevar adelante en el marco de las titulaciones en Informática de la Universidad de Salamanca.

4.7 Perfil profesional del ingeniero en informática

El contexto profesional en el que se van a mover los futuros titulados en Informática es un aspecto a tener en cuenta cuando se elabora un Proyecto Docente, ya que el objetivo de la educación universitaria es formar profesionales capaces de hacer frente a las demandas de la sociedad. Es esencial, por lo tanto, orientar los contenidos de las asignaturas para dar respuesta a las exigencias que la sociedad tiene en cada momento.

Peter J. Denning y R. Dunham [Denning y Dunham, 2001] defienden que los profesionales de las tecnologías de la información son los que ocupan en primer lugar en la tercera oleada profesional. Alvin Toffler [Toffler y Toffler, 1990] divide la historia de la civilización en tres oleadas: la era agrícola, la era industrial y la era de la información. En esta era de la información van a tomar una importancia capital habilidades como coordinación, relación con los clientes, gestión de compromisos, trabajo en equipo, aprendizaje continuo o capacidad de empresarial [Denning y Dunham, 2001].

Actualmente, es difícil precisar qué se entiende por un ingeniero en Informática, incluso muchos profesionales de la informática todavía no tienen muy claro en que consiste su profesión [Earles, 2000; Denning, 2001]. Esto puede ser debido a la falta de madurez de la misma unido al hecho de que esta disciplina ha ido acumulando a lo largo de los años tanto conocimiento científico que la obliga a cambiar constantemente. Existen períodos de tiempo en los que el cambio social y tecnológico es particularmente rápido, por lo que es más necesario entonces analizar y discutir la naturaleza y la evolución de la profesión. El momento actual puede ser uno de esos períodos.

Algunos pioneros en la Informática y organizaciones profesionales han intentado definir esta disciplina [Abrahams, 1987; Hartmanis, 1994; Denning, 1998; Bagert, 1999; Shaw, 2000]. La mayoría de esos intentos concluyen que la Ciencia de la Computación no es ni una Ciencia ni una Ingeniería en el sentido tradicional. Esta Ciencia define una relación nueva entre la teoría y la práctica. En Informática, a diferencia de otras disciplinas, no se puede separar la teoría y la Ingeniería porque sería desastroso [Denning, 1998]. Los recientes avances en algoritmos genéticos y redes neuronales son sólo unos pocos ejemplos de lo que podría producirse cuando se acopla la teoría con la práctica. Esto pone de manifiesto que los aspectos de Ciencia y de Ingeniería, en computación, están mucho más cercanos que en otras disciplinas.

Actualmente, la confusión sobre esta Ciencia, deriva en gran medida del nombre de la disciplina. La palabra Ciencia se refiere a conjunto de conocimientos. Puede usarse colectivamente para hacer referencia al campo de la Ingeniería como Ciencias de Ingeniería, pero el título de ingeniero en Informática no se debe asociar automáticamente con la construcción de software. Los problemas en computación demandan la invención de nuevas metodologías para la exploración de procesos intelectuales y de información. Algunas de esas metodologías se usan para desarrollar hardware y software. A los profesionales de la Informática les concierne el análisis y diseño de hardware y software para realizar nuevas funciones, o para realizar funciones antiguas de una nueva forma [Simons et al., 1991].

Hay una fuerte demanda en la sociedad de profesionales de la Informática que puedan desarrollar sistemas basados en metodologías con una base teórica sólida y probadas científicamente [West, 1997]. Esta necesidad es más evidente en el campo de la Ingeniería del Software debido a que está menos madura que la Ingeniería del Hardware, es más compleja, y produce productos estratégicos.

En la producción de sistemas software son importantes los aspectos técnicos, pero también lo son la disciplina, la capacidad de trabajo en grupo y el conocimiento del dominio de aplicación en el que se va a trabajar. Sin embargo, aún contando con una formación correcta y continuada, no es fácil hacer las cosas correctas porque suele faltar un soporte de la parte de gestión. A diferencia de otros campos profesionales, en Informática no se puede esperar que los gestores y los clientes tengan conocimientos de los métodos de desarrollo de software. Se necesitan profesionales que conozcan sus capacidades y puedan demandar los recursos y el soporte que necesitan para llevar a cabo un trabajo de calidad [Humphrey, 2000].

Los programas en Informática se pueden ver desde una nueva perspectiva, a través de la que se reconoce la importancia del enfoque de sistemas como sistemas de hardware y software. Se puede, entonces, apreciar tres procesos: teoría, abstracción y diseño [Tucker et al., 1991a].

En los últimos años, los mejores centros educativos ofrecen programas en Informática fuertemente acoplados con la Ingeniería. El problema es que esos programas no tienen el mismo nombre. En EEUU se conocen como "*Computer Science*", "*Computer Engineering*" y "*Computer Science and Engineering*", mientras que en Europa se denominan "*Informatics*". La Ingeniería del Software ha evolucionado pasando de ser una actividad de la "Ingeniería de Computadores" a ser una disciplina propia.

Actualmente existen dos tendencias relativas a la profesión informática, una de ellas defiende una única profesión en la que coexistan varias especialidades, mientras que otra tendencia propone la separación en dos profesiones Ingeniería de Computadores e Ingeniería del Software.

Los defensores de la primera corriente sugieren que el establecimiento de la profesión requiere la coordinación de esfuerzos para abordar los siguientes aspectos (en el marco educativo y laboral de EEUU) [El-Kadi, 1999]:

- *Redefinición*. La Informática es una nueva disciplina de Ingeniería. No se debe perder más tiempo intentando conocer su naturaleza, sino saber qué es y qué papel juega en la sociedad.
- *Educación*. Desde esta nueva perspectiva, IEEE-CS y ACM deben diseñar un currículo para preparar a los ingenieros en informática. El *Computing Curricula 91* [Tucker et al., 1991a] es un excelente punto de partida. Puede actualizarse con algunas reformas para centrar los objetivos educativos en una disciplina unificada con algunas subespecialidades. En el diseño de los nuevos currículos se debe prestar especial atención en introducción de nuevas áreas para incluir nuevos campos [Lewis, 1994; Simons et al., 1991]: diseño VSLI, procesamiento paralelo, diseño de redes de ordenadores, sistemas distribuidos, diseño de algoritmos, tecnología de orientación a objetos, inteligencia artificial, sistemas expertos y robótica. La cobertura con detalle de todos esos campos permitirá realizar especializaciones. (Este punto se ha hecho realidad con el *Computing Curricula 2001* [ACM/IEEE-CS, 2001]).
- *Titulación*. Es necesaria una cooperación entre IEEE-CS, ACM, CSAB y ABET en el diseño de exámenes para la obtención de la titulación [ACM/IEEE-CS, 1998; Mead y Turner, 1998; CSAB, 2000; ABET, 2000].

Las actividades que puede desarrollar un Ingeniero en Informática se pueden encuadrar en alguna de las áreas siguientes, que no son mutuamente excluyentes [Ng, 1999]:

- Desarrollo de software.
- Desarrollo de hardware.
- Integración de sistemas.
- Infraestructura de redes.
- Sistemas de control.
- Data warehousing.
- ERP.
- Negocio electrónico y computación Internet/Intranet.

Los defensores de la profesión única argumentan que la Ingeniería del Software no ha madurado lo suficiente para establecerse como una profesión separada, sin embargo sociedades tan prestigiosas como IEEE-CS y ACM promueven la profesión de Ingeniería del Software.

El SEI (*Software Engineering Institute*) ha desarrollado un modelo para caracterizar la madurez de una profesión en función de ocho componentes [Ford y Gibbs, 1996]:

- Educación profesional inicial.
- Acreditación.
- Desarrollo de habilidades.
- Certificación.
- Titulación.
- Desarrollo profesional.
- Código de ética.
- Sociedad profesional.

G. Ford y N. Gibbs estudiaron varias profesiones que pueden considerarse maduras como Medicina o Derecho y comprobaron que el camino de desarrollo profesional en esos campos es muy similar. La Figura 4.10 muestra esas características.

Los aspirantes a profesionales primeramente reciben una **educación**, generalmente universitaria, que precede al trabajo profesional. La calidad de un programa de una titulación universitaria se asegura por la acreditación. Esta acreditación permite a los profesionales graduados en esos programas iniciar su vida profesional con el conocimiento que necesitan para desarrollarla de una manera efectiva.

Para llegar a ser un profesional se deben adquirir habilidades en la aplicación de los conocimientos adquiridos. La **certificación y/o titulación** asegura la capacidad individual para entrar en la práctica profesional. La certificación es un proceso voluntario que ayuda a determinar quien está cualificado para participar en una profesión. La titulación es similar salvo en que es obligatoria y la administra una autoridad gubernamental.

El **desarrollo profesional** se refiere al mantenimiento y mejora de los conocimientos y habilidades adquiridos durante el período educativo después de iniciada la práctica profesional. Estos requisitos de desarrollo son especialmente fuertes en profesiones cuyo cuerpo de conocimiento técnico cambia rápidamente. Un aspecto del desarrollo profesional es el aprendizaje de estándares de práctica apropiados.

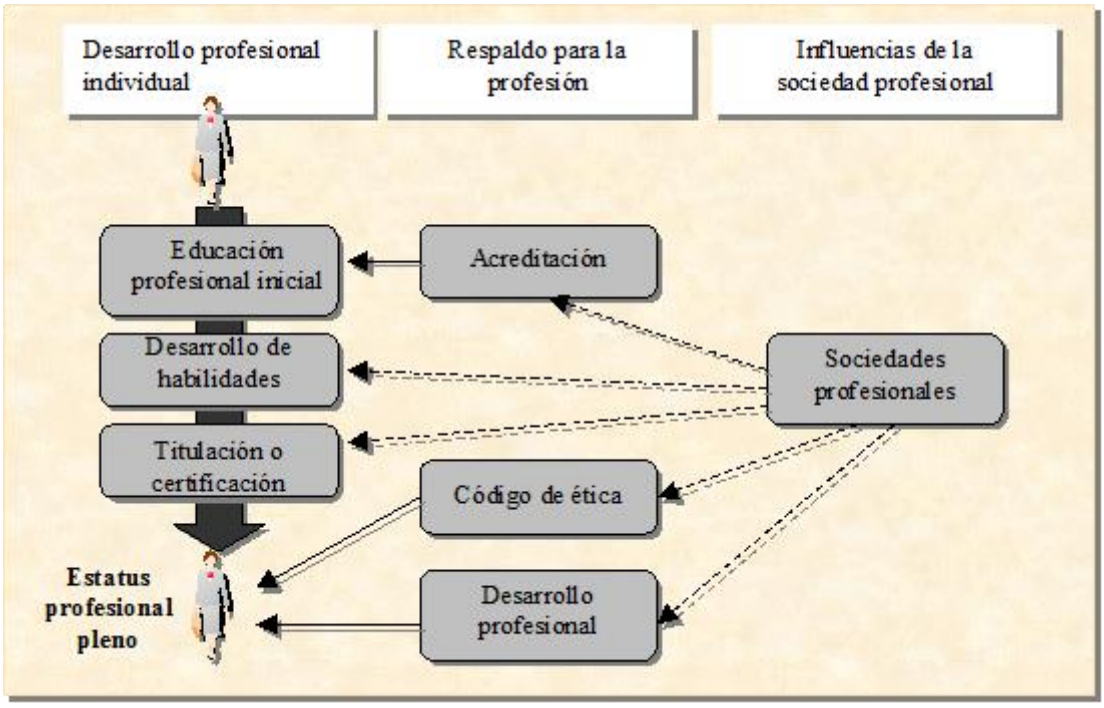

Figura 4.10. Etapas del desarrollo profesional en profesiones bien establecidas

En su comienzo las **sociedades profesionales** tenían como único objetivo promover el intercambio de conocimiento. Las funciones actuales incluyen desarrollo de estándares, definición de criterios de certificación, definición de códigos de ética y acciones disciplinarias para la violación de dichos códigos. Las sociedades IEEE-CS y ACM han trabajado activamente en la definición de la profesión de Ingeniería del Software. Dicho trabajo ha sido instrumentado, bajo su amparo, por el SWECC (*Software Engineering Coordinating Committe*).

La definición de un **código ético** se realiza para dirigir el comportamiento responsable de los profesionales. Una sociedad profesional ayuda a asegurar que el resto de los componentes interactúan de forma apropiada.

Componente	*Ad hoc*	Específico	Maduro
Educación profesional inicial	Preparación común de los titulados para entrar en la profesión.	Existe una formación reconocida de la educación inicial en Ingeniería del Software pero no existe un currículo estándar.	Existe un modelo de currículo nacionalmente aceptado; el modelo se examina y se revisa regularmente.
Acreditación de la Educación	Acreditación basada en criterios de ingeniería o de ciencia de la computación.	Acreditación basada en criterios de Ingeniería del Software. Fusión de ABET y CSAB.	Las directrices se examinan y se revisan regularmente.
Desarrollo de habilidades	Trabajo en proyectos en los centros educativos, programas de cooperación, programas de entrenamiento en compañías.	Surgen las directrices sobre la práctica necesaria para la profesión de ingeniero de software.	Los mecanismos de adquisición de habilidades están situados y su uso está generalizado.
Certificación	Certificación ICCP, ASQC; certificación comercial relacionada con paquetes de software y tecnologías.	Certificación como ingeniero de software. Estándares reconocidos nacionalmente.	Certificación en áreas especializadas de Ingeniería del Software. Estándares de certificación especializada reconocidos nacionalmente.
Titulación	Titulación oficial como ingeniero profesional.	Exámenes dirigidos específicamente a habilidades en Ingeniería del Software.	Titulación basada en exámenes apropiados; colaboración NSPE y NCEE.
Desarrollo profesional	Desarrollo seguido de forma individual según sus necesidades.	Directrices para el desarrollo profesional.	Educación reconocida nacionalmente y directrices de entrenamiento y currículo.
Código ético	Códigos de ACM, IEEE, ASQC, ICCP.	Código ético específico par Ingenieros de software.	Código ampliamente respetado y adoptado; mecanismos disciplinarios de incumplimiento.
Sociedad profesional	ACM, IEEE-CS, otras.	Sociedad que representa explícitamente la Ingeniería del Software.	Sociedad con un nivel apropiado de productos y servicios para los ingenieros de Software.

Tabla 4.18. Evolución de los componentes de la profesión de Ingeniería del Software

La caracterización de la madurez en el ámbito de componentes de una profesión se realiza en función de cuatro etapas evolutivas:

- *No existencia*. El componente no existe de ninguna forma.

- *Ad Hoc*. Existe algo relacionado con el componente, pero no se identifica con la profesión.
- *Específico*. El componente existe y se identifica claramente con la profesión.
- *Maduro*. El componente ha existido durante muchos años, se ha llevado a cabo de manera apropiada dentro de la profesión y está sujeto a mejoras continuas.

Se considera que una profesión está madura cuando sus componentes han alcanzado el nivel de madurez.

Los autores del documento que recoge este modelo de madurez, reconocen que la Ingeniería del Software es una profesión relativamente inmadura, sin embargo sugieren que el modelo propuesto puede ayudar a la maduración de la profesión. La Tabla 4.18 refleja la visión de dichos autores sobre la evolución de la profesión de Ingeniería del Software.

En el *Computing Curricula 2001* [ACM/IEEE-CS, 2001] se indican las características, capacidades y habilidades que se les suponen a los titulados en Informática. Brevemente, se enumeran a continuación:

- **Características generales:**
  - Perspectiva en el ámbito del sistema.
  - Apreciación de la interrelación entre teoría y práctica.
  - Familiaridad con los temas comunes (tales como abstracción, complejidad y evolución).
  - Experiencia significativa en proyectos.
  - Adaptabilidad.
- **Capacidades y habilidades:**
  - *Cognitivas:*
    - Conocimiento y comprensión.
    - Modelado.
    - Obtención y especificación de requisitos.
    - Métodos y herramientas.
    - Responsabilidad profesional.
  - *Prácticas:*
    - Diseño e implementación.
    - Evaluación.

- Gestión de la información.
- Interacción Persona-Ordenador.
- Manejo de los riesgos.
- Herramientas.
- Operación con hardware y software.

- *Adicionales:*
  - Comunicación.
  - Trabajo en equipo.
  - Capacidad de comprender y explicar cuantitativamente la dimensión de un problema.
  - Autogestión.
  - Desarrollo profesional.

Como se ha comentado anteriormente, sociedades como IEEE-CS y ACM han desarrollado actividades encaminadas a hacer evolucionar la profesión de Ingeniería del Software, tales como el establecimiento de requisitos de certificación para los desarrolladores de software y acreditación para programas universitarios [ACM/IEEE-CS, 1998]. La acreditación es un tema espinoso, con opiniones encontradas. Así, como argumenta Parnas, las certificaciones dan confianza, pero no garantizan resultados seguros [Parnas, 2001].

Para completar este trabajo *IEEE-CS/ACM Join Task Force* han establecido recientemente otra clave de éxito para esta profesión: el código de ética de la Ingeniería del Software (*Software Engineering Code of Ethics and Professional Practice*) [ACM/IEEE-CS, 1999c]. La versión 5.2 ha sido adoptada por ambas sociedades después de un intenso proceso de revisión [Gotterbarn et al., 1999a]. En el Cuadro 4.1 se ofrece un breve resumen del mismo.

En España existe una Proposición de Ley para la creación de los Colegios Oficiales de Doctores e Ingenieros en Informática. Tales Colegios Profesionales deberán contemplar códigos de conducta. Gran parte de las tareas de los ingenieros en informática están relacionadas con el software, por lo que el código de la ACM/IEEE-CS que puede ser de gran utilidad para orientar la profesión en nuestro país [Dolado, 1999].

Como se ha comentado en apartados anteriores de este capítulo, las sociedades ACM e IEEE-CS están trabajando también de manera conjunta en un proyecto para desarrollar la guía SWEBOK (*Software Engineering Body of Knowledge*) [Abran et al., 2001c]. La articulación de

dicho cuerpo de conocimiento es un paso esencial par el desarrollo de una profesión debido a que representa un amplio consenso en los contenidos de la disciplina.

4.8 Referencias

[ABET, 2000] Accreditation Board for Engineering and Technology. "*Accreditation Policy and Procedure Manual*". Baltimore, MD: ABET, Inc., http://www.abet.org/images/policies.pdf. November 2000.

[Abi-Raad, 2000] Abi-Raad, M. "*Systems analysis with attitude!*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITICSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 57-60. ACM Press, 2000.

[Abrahams, 1987] Abrahams, P. "*President's letter*". Communications of the ACM, 30(6):472-47, June 1987.

[Abran et al., 1999] Abran, Alain (Co-Excutive Editor), Moore, James W. (Co-Excutive Editor), Bourque, Pierre (Editor), Dupuis, Robert (Editor) and Tripp, Leonard L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.5. ACM/IEEE-CS, October 1999.

[Abran et al., 2000] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Project Champion). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.7. ACM/IEEE-CS, April 2000. Available on line at http://www.swebok.org [Última vez visitado 31-12-2001].

[Abran et al., 2001a] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair, Professional Practics Committee). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Version 0.9. ACM/IEEE-CS, February 2001. Available on line at http://www.swebok.org [Última vez visitado 31-12-2001].

[Abran et al., 2001b] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair, Professional Practics Committee). "*Guide to the Software Engineering Body of Knowledge. A Stone Man Version*". Trial Version 0.95. IEEE-CS, May 2001. Available on line at http://www.swebok.org [Última vez visitado 28-12-2001].

[Abran et al., 2001c] Abran, A. (Co-Excutive Editor), Moore, J. W. (Co-Excutive Editor), Bourque, P. (Editor), Dupuis, R. (Editor), Tripp, L. L. (Chair). "*Guide to the Software Engineering Body of Knowledge SWEBOK*". IEEE-CS Press, 2001.

[ACM, 1965] ACM Curriculum Committee on Computer Science. "*An Undergraduate Program in Computer Science – Preliminary Recommendations*". Communications of the ACM, 8(9):543-552. September 1965.

[ACM, 1968] ACM Curriculum Committee on Computer Science. "*Curriculum '68: Recommendations for Academic Programs in Computer Science*". Communications of the ACM, 11(3):151-197. March 1968.

[ACM, 1981] ACM Committee on Computer Curricula of ACM Education Board. "*ACM Recommended Curricula for Computer Science and Information Processing Programs in Colleges and Universities*". ACM, New York, 1981.

[ACM/IEEE-CS, 1998] ACM/IEEE-CS. "*Accreditation Criteria for Software Engineering*". http://www.acm.org/serving/se/Accred.htm. [Última vez visitado 31/12/2001]. September 1998.

[ACM/IEEE-CS, 1999a] ACM/IEEE-CS. "*Software Engineering Education Project*". http://www.acm.org/serving/se/SWEE.htm. [Última vez visitado 31/12/2001]. 1999.

[ACM/IEEE-CS, 1999b] ACM/IEEE-CS. "*1999 Plan for the Software Engineering Education Project (SWEEP)*". Draft 0.5. http://www.acm.org/serving/se/sweep.htm. [Última vez visitado 31/12/2001]. April 1999.

[ACM/IEEE-CS, 1999c] ACM/IEEE-CS. "*Software Engineering Code of Ethics and Professional Practice*". Version 5.2. http://www.acm.org/serving/se/code.htm. [Última vez visitado 31/12/2001]. 1999.

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

[Ada95-Web] "*Ada 95 Reference Manual: Language and Standard Library*". http://lglwww.epfl.ch/Ada/LRM/9X/rm9x/rm9x-toc.html [Última vez visitado 8/1/2000].

[Adams, 1996] Adams, J. C. "*Object-Centered Design. A Five-Phase Introduction to Object-Oriented Programming in CS1-2*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 78-82. ACM. 1996.

[AECC, 1986] Asociación Española para la Calidad. "*Glosario de Términos de Calidad e Ingeniería del Software*". AECC, 1986.

[Aiken, 1980] Aiken, R. M. "*Computer Science Education in the 1980's*". IEEE Computer, 13(1):41-46. January 1980.

[Alhir, 1998] Alhir, S. S. "*The Object-Oriented Paradigm*". http://home.earthlink.net/~salhir/theobjectorientedparadigm.html. [Última vez visitado 23/12/1999]. October 1998.

[Ardis y Ford, 1989] Ardis, M., Ford, G. "*1989 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-89-TR-21 (ESD-TR-89-29), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). June 1989.

[Ashenhurst, 1972] Ashenhurst, R. L. (Editor). "*Curriculum Recommendations for Graduate Professional Programs in Information Systems*". ACM, 1972.

[Ashworth y Goodland, 1990] Ashworth, C., Goodland, M. "*SSADM: A Practical Approach*". McGraw-Hill, 1990.

[Atkinson et al., 1989] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S. "*The Object-Oriented Database System Manifesto*". In Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto (Japan). 1989. Also in Deductive and Object-Oriented Databases, Elsevere Science Publishers, Amsterdam, Netherlands, 1990.

[Atkinson y Buneman, 1987] Atkinson, M., Buneman, P. "*Types and Persistence in Database Programming Languages*". ACM Computing Surveys, 19(2). 1987.

[Austing et al., 1979] Austing, R., Barnes, B., Bonnette, D., Engel, G., Stokes, G. "*Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*". Communications of the ACM, 22(3):147-166. March 1979.

[Baber, 1998] Baber, R. L. "*Software Engineering Education: Issues and Alternatives*". Annals of Software Engineering, 6:39-59. 1998.

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

[Bagert et al., 1999] Bagert, D. J., Hilburn, T. B., Hislop, G., Lutz, M., McCracken, M., Mengel, S. "*Guidelines for Software Engineering Education. Version 1.0*". Working Group on Software Engineering Education and Training (WGSEET). August 1999.

[Bailin, 1989] Bailin, S. C. "*An Object-Oriented Requirements Specification Method*". Communications of the ACM, 32(5):608-623. May 1989.

[Barr y Feigenbaum, 1981] Barr, A., Feigenbaum, E. "*The Handbook of Artificial Intelligence*". Vol. 1. William Kaufmann, 1981.

[Basili, 1991] Basili, V. R. "*The Future Engineering of Software: A Management Perspective*". IEEE Computer, 24(9):90-96. September 1991.

[Bauer, 1972] Bauer, F. L. "*Software Engineering*". Information Processing 71. Amsterdam: North Holland, 1972.

[BCS, 1989] The British Computer Society and The Institution of Electrical Engineering. "*A Report on Undergraduate Curricula for Software Engineering Curricula*". June 1989.

[BCS, 1995] The British HCI Group. "*Whither HCI education in the UK?*". Interfaces, Nº28, Spring, 1995.

[Beizer, 2000] Beizer, B. "*Software Is Different*". Annals of Software Engineering, 10:293-310. 2000.

[Bellin, 1999] Bellin, D. "*Pedagogical Pattern #4. Brainstorming Pattern*". Version 2.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp4.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Bergin, 1998a] Bergin, J. "*Pedagogical Patterns*". http://csis.pace.edu/~bergin/PedPat1.2.html. [Última vez visitado, 3/8/1999]. October 1998.

[Bergin, 1998b] Bergin, J. "*Six Pedagogical Patterns*". http://csis.pace.edu/~bergin/fivepedpat.html. [Última vez visitado, 3/8/1999]. October 1998.

[Bergin, 1998c] Bergin, J. "*Pedagogical Pattern #32. Spiral Pattern*". Versión 1.2. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp32.htm. [Última vez visitado, 20/8/1999]. October 1998.

[Bertino y Martino, 1993] Bertino, E., Martino, L. "*Object-Oriented Database Systems. Concepts and Architectures*". Addison-Wesley, 1993.

[Berztiss, 1987] Berztiss, A. "*A Mathematically Focused Curriculum for Computer Science*". Communications of the ACM, 5(30):356-365. May 1987.

[Bézivin et al., 1992] Bézivin, J., Roux, O., Royer, J.-C. "*Teaching Object-Oriented Programming or Using the Object Model to Teach Software Engineering*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 269-275. ACM. 1992.

[Birtwistle et al., 1973] Birtwistle, G. M., Dahl, O.-J., Myhrhaug, B., Nygaard, K. "*Simula Begin*". Studentliteratur, 1973.

[Blaha y Premerlani, 1998] Blaha, M., Premerlani, W. "*Object-Oriented Modeling and Design for Database Applications*". Prentice Hall, 1998.

[Blair et al., 1991] Blair, G., Gallagher, J., Hutchinson, D., Shepard, D. "*Object-Oriented Languages, Systems and Applications*". Halsted Press, 1991.

[Blum, 1992] Blum, B. I. "*Software Engineering, A Holistic View*", Oxford University Press, New York, 1992.

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

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

[Boehm, 1976] Boehm, B. W. "*Software Engineering*". IEEE Transactions on Computers. C-25(12):1226-1241. December 1976.

[Boehm, 1981] Boehm, B. W. "*Software Engineering Economics*". Prentice Hall, 1981.

[Booch, 1991] Booch, G. "*Object Oriented Design with Applications*". The Benjamin/Cummings Publishing Company, 1991.

[Booch, 1994] Booch, G. "*Object Oriented Analysis and Design with Applications*". 2nd Edition. The Benjamin/Cummings Publishing Company, 1994.

[Booch et al., 1996a] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.9 Addendum. Rational Software Corporation, June 1996.

[Booch et al., 1996b] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 0.91 Addendum UML update. Rational Software Corporation, September 1996.

[Booch et al., 1997a] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0. Rational Software Corporation, 13 January 1997.

[Booch et al., 1997b] Booch, G., Jacobson, I., Rumbaugh, J. "*The Unified Modeling Language for Object-Oriented Development*". Documentation set, version 1.0.1. Rational Software Corporation, 19 March 1997.

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. "*The Unified Modeling Language User Guide*". Object Technology Series. Addison-Wesley, 1999.

[Booch y Rumbaugh, 1995] Booch, G., Rumbaugh, J. "*Unified Method for Object-Oriented Development*". Documentation set, version 0.8. Rational Software Corporation, 1995.

[Bourque et al., 1998] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L., Shyne, K., Pflug, B., Maya, M., Tremblay, G. "*Guide to the Software Engineering Body of Knowledge. A Straw Man Version*". ACM/IEEE-CS, September 1998.

[Bourque et al., 1999a] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L. "*The Guide to the Software Engineering Body of Knowledge*". IEEE Software, 16(6):35-44. November-December 1999.

[Bourque et al., 1999b] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., Tripp, L., Frailey, D. "*Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge*". Technical Report. January 1999.

[Bowyer, 1996] Bowyer, K. W. "*Ethics and Computing: Living Responsibly in a Computerized World*". IEEE Computer Society Press, 1996.

[Brandt, 1997] Brandt, D. S. "*Constructivism: Teaching for Understanding of the Internet*". Communications of the ACM, 40(10):112-117. October 1997.

[Bryant, 2000] Bryant, A. "*Metaphor, Myth and Mimicry: The Bases of Software Engineering*". Annals of Software Engineering, 10:273-292. 2000.

[Budd, 1991] Budd, T. "*An Introduction to Object-Oriented Programming*". Addison-Wesley, 1991.

[Buxton y Randell, 1970] Buxton, J. N., Randell, B. (Editors). "*Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 October, 1969*". Brussels: Scientific Affairs Division, NATO. April 1970.

[Buxton et al., 1976] Buxton, J. M., Naur, P., Randell, B. (Editors) "*Software Engineering Concepts and Techniques*". Proceedings of 1968 NATO Conference on Software Engineering, Van Nostrand Reinhold, 1976.

[Cameron, 1989] Cameron, J. "*JSP & JSD: The Jackson Approach to Software Development*". 2nd edition. IEEE Computer Society Press, 1989.

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

[Canos et al., 1998] Canós, J. H., Penadés, M. C., Pelechano, V., Sánchez, J., Ramos, I., González, A. "*La Ingeniería del Software en los Planes de Estudio: ¿Dos Perspectivas de una Disciplina o Más de lo Mismo?*". IV Jornades sobre L'Ensenyament Universitari de la Informática (JENUI'98). Páginas 105-112, San Julià de Loira, Andorra, 1998.

[Castellanos et al., 1991] Castellanos, M. G., Saltor, F., García-Solaco, M. "*The Development of Semantic Concepts in BLOOM Model Using an Object Metamodel*". Technical Report LSI-91-22. Dept. de Llenguatges i Sistemes Informátics. Universidad Politécnica de Cataluña, 1991.

[Coleman et al., 1994] Coleman, D., Arnold, P., Bodoff, S., Dolin, C., Hayes, F., Jeremaes, P. "*Object-Oriented Development: The Fusion Method*". Prentice-Hall, 1994.

[Cook y Daniels, 1994] Cook, S., Daniels, J. "*Designing Object Systems: Object Oriented Modelling with Syntropy*". Prentice-Hall, 1994.

[Cook et al., 1997] Cook, S., Selic, B., Gangopadhyay, D., Gheorge, S., Gullekson, G., Hogg, J., McGee, J., Meier, M., Mitra, S., Saaltink, M., Warmer, J., Wills, A. "*OMG OA&D RFP OMG OA&D RFP Response*". Document Version 1.0. IBM Corporation and ObjecTime Limited. 10 January 1997.

[COSINE, 1967] COSINE Committee. "*Computer Science in Electrical Engineering*". Washington, DC: Commission on Engineering Education. September 1967.

[Couger, 1973] Couger, J. (Editor). "*Curriculum Recommendations for Undergraduate Programs in Information Systems*". Communications of the ACM, 16(12): 727-749. December 1973.

[Cowling, 1998] Cowling, A. J. "*A Multi-Dimensional Model of the Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education and Training – CSEE&T'98. (February 22-25, 1998, Atlanta, GA, USA). Pages 44-55. IEEE Computer Society. 1998.

[CSAB, 2000] Computing Sciences Accreditation Board. "*Criteria for Accrediting Programs in Computer Science in the United States*". Version 1,0, http://www.csab.org/criteria2k_v10.html. January 2000.

[Champeaux et al., 1993] Champeaux, D., Lea, D., Faure, P. "*Object-Oriented System Development*". Addison Wesley, 1993.

[Chang et al., 1999] Chang, C. K., Engel, G., King, W., Roberts, E., Shackelford, R., Sloan, R. H., Srimani, P. K. "*Curricula 2001: Bringing the Future to the Classroom*". Computer, 32(9):85-88. September 1999.

[Chen, 1976] Chen, P. "*The Entity-Relationship Model: Toward a Unified View of Data*". ACM Transactions on Database Systems, 1(1):9-36. March 1976.

[Dahl et al., 1970] Dahl, O.-J., Myrhaug, B., Nygaard, K. "*(Simula 67) Common Base Language*". Norsk Regnesentral (Norwegian Computing Center), Publication N. S-22, Oslo. October 1970.

[Dahl y Hoare, 1972] Dahl, O.-J., Hoare, C. A. R. "*Hierarchical Program Structures*". In Dahl, Dijkstra, Hoare, *Structured Programming*, Academic Press, pages 175-220. 1972.

[Davis, 1982] Davis, A. M. "*Rapid Prototyping Using Executable Requirements Specifications*". ACM Software Engineering Notes, 7(5):39-44, 1982.

[Davis, 1992] Davis, A. M. "*Operational Prototyping: A new Development Approach*". IEEE Software, 9(5):70-78, 1992.

[Davis, 1993] Davis, A. M. "*Software Requirements. Objects, Functions and States*". Prentice-Hall International, 1993.

[Davis et al., 1997] Davis, G. B., Gorgone, J. T., Couger, J. D., Feinstein, D. L., Longenecker, Jr. H. E. (Editors). "*IS'97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*". ACM, AIS y AITP, 1997.

[Decker y Hirshfield, 1994] Decker, R., Hirshfield, S. "*The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS1*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education (SIGCSE'94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 51-55. ACM. 1994.

[DeMarco, 1979] DeMarco, T. "*Structured Analysis and System Specification*". Prentice-Hall, 1979.

[Denning, 1992] Denning, P. J. "*Educating a New Engineer*". Communications of the ACM, 35(12):82-97. December 1992.

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

[Denning, 2001] Denning, P. J. "*Who Are We*". Communications of the ACM, 44(2):15-19. February 2001.

[Denning et al., 1988] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J., Young, P. R. "*Report of the ACM Task Force on the Core of Computer Science*". ACM Press, 1988.

[Denning et al., 1989] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A. B., Turner, A. J., Young, P. R. "*Computing as a Discipline*". Communications of the ACM, 32(1):9-23. January 1989.

[Denning y Dunham, 2001] Denning, P. J., Dunham, R. "*The Core of the Third-Wave Professional*". Communications of the ACM, 44(11):21-25. November 2001.

[Devlin, 2001] Devlin, K. "*The Real Reason Why Software Engineers Need Math*". Communications of the ACM, 44(10):21-22, October 2001.

[Dijkstra, 1968a] Dijkstra, E. "*Go to Statement Considered Harmful*". Communications of the ACM, 11(3):147-148. March 1968.

[Dijkstra, 1968b] Dijkstra, E. "*The Structure of 'THE' Multiprogramming System*". Communications of the ACM, 11(5):341-346. May 1968.

[Diller, 1990] Diller, A. "*Z: An Introduction to Formal Methods*". John Wiley & Sons, 1990.

[Dodani, 1999] Dodani, M. "*OO Learning AntiPatterns: Rewriting Data and Functional Thinkers into Object Technology Developers*". Journal of Object-Oriented Programming (JOOP), 11(8):59-63. January 1999.

[Dolado, 1999] Dolado, J. "*El Código de Ética y Práctica Profesional de la Ingeniería del Software de la ACM/IEEE Computer Society*". Novática. Nº 140. Julio-Agosto, 1999.

[Dorling, 1993] Dorling, A. "*Software Process Improvement and Capability Determination*". Software Quality Journal, 12(4): 209-224. December 1993.

[Dowell y Long, 1989] Dowell, J., Long, J. "*Towards a Conception of an Engineering Discipline of HCI*". Ergonomics, 32:1513-1535. 1989.

[DPMA, 1981] Data Processing Management Association. "*DPMA Model Curriculum, 1981*". Published by DPMA, Chicago, 1981.

[DRAE, 1995] Real Academia Española. "*Diccionario de Real Academia*". Vigésimo primera edición. Espasa-Calpe. Edición electrónica, versión 21.1.0. 1995.

[Duncan, 1996] Duncan, W. R. "*A Guide to the Project Management Body of Knowledge*". PMI Standards Committee. Project Management Institute, Four Campus Boulevart, Newtown Square, PA 19073-3299, USA. 1996.

[Dupuis et al., 1999] Dupuis, R., Bourque, P., Abran, A., Moore, J. W., Tripp, L. L. "*Guide to the Software Engineering Body of Knowledge*". In 1999 Frontiers in Education Conference – FIE'99. (San Juan, Porto Rico, November 10-13, 1999). http://www.lrgl.uqam.ca/publications/pdf/467.pdf. [Última vez visitado, 31-12-2001]. 1999.

[Dupuis et al., 2000] Dupuis, R., Bourque, P., Tait, J., McGarry, J., DeWeese, P., Moore, J. W. "*Guide to the Software Engineering Body of Knowledge. Overview and Applications*". In STC 2000, The Twelfth Annual Software Technology Conference, Salt Lake City, Utah 30 April - 4 May 2000. http://www.lrgl.uqam.ca/publications/pdf/514.pdf. [Última vez visitado, 31-12-2001]. 2000.

[D'Souza, 1996] D'Souza, D. F. "*Objects: Education vs. Training*". ICON Computing Inc. http://www.iconcomp.com/papers/education-vs-training/EducationvsTraining.frm.html. [Última vez visitado, 24/5/1999]. 1996.

[D'Souza y Wills, 1999] D'Souza, D. F., Wills, A. C. "*Objects, Components, and Frameworks with UML. The Catalysis Approach*". Object Technology Series. Addison-Wesley, 1999.

[EAB, 1983] Education Activities Board. "*The 1983 Model Program in Computer Science and Engineering*". Technical Report 932. IEEE Computer Society. December 1983.

[Earles, 2000] Earles, J. "*Software Engineering – Myth or Reality*". http://www.cbd-hq.com. 2000.

[EC, 1977] Education Committee of the IEEE Computer Society. "*A Curriculum in Computer Science and Engineering*". Publication EHO119-8, Computer Society of the IEEE. January 1977.

[Ehrig y Mahr, 1985] Ehrig, H., Mahr, B. "*Fundamentals of Algebraic Specification 1*". Springer-Verlag, EATCS Nº6, 1985.

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

[Fairley, 1985] Fairley, R. "*Software Engineering Concepts*". McGraw-Hill, 1985.

[Faulkner y Culwin, 2000] Faulkner, X., Culwin, F. "*Enter the Usability Engineer: Integrating HCI and Software Engineering*". In Proceedings of 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education – ITICSE'00. (July 11 - 13, 2000, Helsinki Finland). Pages 61-64. ACM Press, 2000.

[Firesmith et al., 1998] Firesmith, D., Henderson-Sellers, B., Graham, I. "*OPEN Modeling Language (OML) Reference Manual*". Cambridge University Press, 1998.

[Ford, 1990] Ford, G. "*1990 SEI Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-90-TR-3 (ESD-TR-90-204), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). March 1990.

[Ford, 1991a] Ford, G. "*1991 SEI Report on Graduate Software Engineering Education*". Technical Report CMU/SEI-91-TR-2 (ESD-TR-91-2), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April 1991.

[Ford, 1991b] Ford, G. "*The SEI Undergraduate Curriculum in Software Engineering*". In Proceedings of the Twenty-Second SIGCSE Technical Symposium on Computer Science Education – SIGCSE'91. (March 7-8, 1991, San Antonio, Texas, USA). Pages 375-385. ACM. 1991.

[Ford, 1994] Ford, G. "*A Progress Report on Undergraduate Software Engineering Education*". Technical Report CMU/SEI-94-TR-11 (ESC-TR-94-011), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). May 1994.

[Ford y Gibbs, 1989] Ford, G., Gibbs, N. E. "*A Master of Software Engineering Curriculum*". IEEE Computer, 22(9):59-71, September 1989.

[Ford y Gibbs, 1996] Ford, G., Gibbs, N., E. "*Mature Profession of Software Engineering*". Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). January 1996.

[Frakes et al., 1991] Frakes, W. B., Fox, C., Nejmeh, B. A. "*Software Engineering in the UNIX/C Environment*". Prentice Hall, 1991.

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

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

[GAFC-USAL, 1999] Facultad de Ciencias. "*Guía Académica de la Facultad de Ciencias 1999-2000*". Ediciones Universidad de Salamanca, 1999.

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

[Gane y Sarson, 1977] Gane, C., Sarson, T. "*Structured Systems Analysis and Design*". Improved Systems Technologies, Inc., 1977.

[Gane y Sarson, 1979] Gane, C., Sarson, T. "*Structured Systems Analysis: Tools and Techniques*". Prentice-Hall, 1979.

[García, 2000] García, F. J. "*Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*". Tesis Doctoral. Facultad de Ciencias, Universidad de Salamanca. Enero, 2000.

[García et al., 2000] García Peñalvo, F. J., Moreno García, Mª N., García-Bermejo Giner, J. R. y Luis Reboredo, A. de. "*Unidad Docente de Ingeniería del Software y Orientación a Objetos. Plan de Calidad Versión 1.1*". Ingeniería Técnica en Informática de Sistemas. Universidad de Salamanca. Bienio 1999-2001. Marzo, 2000.

[García y Pardo, 1998] García Peñalvo, F. J., Pardo Aguilar, C. "*UML 1.1. Un Lenguaje de Modelado Estándar para los Métodos de ADOO*". Revista Profesional para Programadores (RPP), Editorial América-Ibérica, V(1):57-61. Enero, 1998.

[Garlan et al., 1997] Garlan, D., Gluch, D. P., Tomayko, J. E. "*Agents of Change: Educating Software Engineering Leaders*". IEEE Computer, 30(11):59-65 November 1997.

[Gibbs, 1989] Gibbs, N. E. "*The SEI Education Program: The Challenge of Teaching Future Software Engineers*". Communications of the ACM, 32(5):594-605. May 1989.

[Gibbs y Tucker, 1986] Gibbs, N. E., Tucker, A. B. "*Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 29(3):202-210. March 1986.

[Ginige y Murugesan, 2001] Ginige, A., Murugesan, S. "*Web Engineering-An Introduction*". IEEE Multimedia, 8(1):14-18. January-March 2001.

[Goguen et al., 1992] Goguen, J. A., Winkler, T., Meseguer, J., Futatsugui, K., Jouannaud, J. P. "*Introducing OBJ*". SRI-CSL Report. Draft of January 1992.

[Goguen y Meseguer, 1988] Goguen, J. A., Meseguer, J. "*Order-Sorted Álgebra I*". Technical Report, SRI International, Stanford University, 1988.

[Goldberg, 1985] Goldberg, A. "*Smalltalk-80: The Interactive Programming Environment*". Addison-Wesley, 1985.

[Goldberg, 1986] Goldberg, R. "*Software Engineering: An Emerging Discipline*". IBM Systems Journal. 25(3/4), 1986.

[Goldberg y Robson, 1983] Goldberg, A., Robson, D. "*Smalltalk-80: The Language and its Implementation*". Addison-Wesley, 1983.

[Gomaa, 1983] Gomaa, H. "*The Impact of Rapid Prototyping on Specifying User Requirements*". ACM Software Engineering Notes, 8(2):17-28, 1983.

[Gómez et al., 1998] Gómez, A., Juristo, N., Montes, C. y Pazos, J. "*Ingeniería del Conocimiento*". Madrid. Ceura, 1998.

[Gorgone et al., 1999] Gorgone, J. T., Gray, P., Feinstein, D. L., Kasper, G. M., Luftman, J. N., Stohr, E. A., Valacich, J., Wigand, R. T. "*MSIS 2000 Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems*". ACM and AIS. November 1999.

[Gotterbarn, 1999] Gotterbarn, D. "*How the New Software Engineering Code of Ethics Affects You*". IEEE Software, 16(6):58-64. November/December 1999.

[Gotterbarn et al., 1997a] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics*". Communications of the ACM, 40(11):110-118. November 1997.

[Gotterbarn et al., 1997b] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics, Version 3.0*". IEEE Computer, 30(11):88-92. November 1997.

[Gotterbarn et al., 1999a] Gotterbarn, D., Miller, K., Rogerson, S. "*Computer Society and ACM Approve Software Engineering Code of Ethics*". IEEE Computer, 32(10):84-88. October 1999.

[Gotterbarn et al., 1999b] Gotterbarn, D., Miller, K., Rogerson, S. "*Software Engineering Code of Ethics Is Approved*". Communications of the ACM, 42(10):102-107. October 1999.

[Graham, 1994] Graham, I. "*Object-Oriented Methods*". 2nd edition. Addison-Wesley, 1994.

[Graham et al., 1997] Graham, I., Henderson-Sellers, B., Younessi, H. "*The Open Process Specification*". Addison Wesley (Open Series), 1997.

[Guttag, 1980] Guttag, J. "*Abstract Data Types and the Development of Data Structures*". In *Programming Language Design*. Computer Society Press, 1980.

[Hadjerrouit, 1999] Hadjerrouit, S. "*A Constructivist Approach to Object-Oriented Design and Programming*". In Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education (ITiCSE'99). (June 27-July 1, 1999, Cracow, Poland). Pages 171-174. ACM. 1999.

[Hartmanis, 1994] Hartmanis, J. "*On Computational Complexity and the Nature of Computer Science*". Communications of the ACM, 37(10):198-202. October 1994.

[Hatley y Pirbhai, 1987] Hatley D. J., Pirbhai, A. "*Strategies for Real-Time System Specification*". Dorset House, 1987.

[Hefley et al., 1992] Hefley, B. (editor), Hewett, T. T. (chair), Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W. "*Curricula for Human-Computer Interaction*". ACM SIGCHI. http://sigchi.org/cdg/. [Última vez visitado, 4/1/2002]. ACM Press, 1992.

[Henderson-Sellers y Edwards, 1994a] Henderson-Sellers, B., Edwards, J. M. "*BOOKTWO of Object-Oriented Knowledge: The Working Object*". Prentice-Hall, 1994.

[Henderson-Sellers y Edwards, 1994b] Henderson-Sellers, B., Edwards, J. M. "*MOSES: A Second Generation Object-Oriented Methodology*". Object Magazine, pp. 68-73. June 1994.

[Henderson-Sellers et al., 1998] Henderson-Sellers, B., Simons, A., Younessi, H. "*The Open Toolbox of Techniques*". Open Series. Addison Wesley, 1998.

[Hilburn, 1997] Hilburn, T. B. "*Software Engineering Education: A Modest Proposal*". IEEE Software, 14(6):44-48. November/December 1997.

[Hilburn et al., 1998] Hilburn, T. B., Bagert, D. J., Mengel, S., Oexmann, D. "*Software Engineering Across Computing Curricula*". In Proceedings of the 6th Annual Conference on the Teaching of Computing/3rd Annual Conference on Integrating Technology into Computer Science Education on Changing the Delivery of Computer Science Education, ITiCSE '98. (Aug. 17-21, 1998, Dublin City Univ., Ireland). Pages 117-121. ACM. 1998.

[Hilburn et al., 1999] Hilburn, T. B., Hirmanpour, I., Khajenoori, S., Turner, R., Qasem, A. "*A Software Engineering Body of Knowledge Version 1.0*". Technical Report CMU/SEI-99-TR-004 (ESC-TR-99-004), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). April 1999.

[Hirmanpour et al., 1995] Hirmanpour, I. Hilburn, T. B., Kornecki, A. "*A Domain Centered Curriculum. An Alternative Approach to Computing Education*". In Proceedings of the 26th SISCSE Technical Symposium on Computer Science Education, SIGCSE'95. (March 2-4, 1995, Nashville, TN, USA). ACM. 1995.

[Hoare, 1975] Hoare, C. A. R. "*Software Engineering*". Computer Bulletin. Pages 6-7. December 1975.

[Hoare, 1985] Hoare, C. A. R. "*Communicating Sequential Processes*". Prentice-Hall, 1985.

[Holland et al., 1997] Holland, S., Griffiths, R., Woodman, M. "*Avoiding Object Misconceptions*". In Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE'97). (Feb. 27-Mar. 1, 1997, San Jose, CA – USA). Pages 131-134. 1997.

[Hopcroft, 1987] Hopcroft, J. E. "*Computer Science: The Emergence of a Discipline*". Communications of the ACM, 30(3):198-202. March 1987.

[Horan, 1995] Horan, P. "*Software Engineering - A Field Guide*". Deakin University. http://www.cm.deakin.edu.au/~peter/SEweb/field_gu.html**.** December 1995.

[Humphrey, 1989] Humphrey, W. S. "*Managing the Software Process*". Addison-Wesley, 1989.

[Humphrey, 1993] Humphrey, W. S. "*Software Engineering*" in Ralston, A. and Reilly, E.D. (eds.), *Encyclopedia of Computer Science,* Van Nostrand Reinhold, p. 1218,1993.

[Humphrey, 2000] Humphrey, W. S. "*Software – A Performing Science?*". Annals of Software Engineering, 10:261-271. 2000.

[IEEE, 1983] IEEE. "*Standard Glossary of Software Engineering Terminology*". ANSI/IEEE Std. 729-1983. IEEE, 1983.

[IEEE, 1999] IEEE. "*IEEE Software Engineering Standards Collection 1999 Edition. Volume 1: Customer and Terminology Standards*". IEEE Computer Society Press, 1999.

[IEEE-CS, 2001a] IEEE-CS. "*IEEE Multimedia. Special Issue on Web Engineering Part I*". Vol.8, N1, January-March 2001.

[IEEE-CS, 2001b] IEEE-CS. "IEEE Multimedia. Special Issue on Web Engineering Part II". Vol.8, N2, April-June 2001.

[ISO/IEC, 1995] ISO/IEC. "*Information Technology – Software Life Cycle Processes*". Technical ISO/IEC 12207:1995(E), 1995.

[Jacobson et al., 1993] Jacobson, I., Christerson, M., Jonsson, P., Övergaad, G. "*Object Oriented Software Engineering: A Use Case Driven Approach*". Addison-Wesley, 1992. Revised 4th printing, 1993.

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J. "*The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.

[Jackson, 1975] Jackson, M. A. "*Principles of Program Design*". Academic Press, 1975.

[Jackson, 1983] Jackson, M. A. "*System Development*". Prentice-Hall, 1983.

[Jackson et al., 1997] Jackson, U., Manaris, B., McCauley, R. "*Strategies for Effective Integration of Software Engineering Concepts and Techniques into the Undergraduate Computer Science Curriculum.*". In Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education, SIGCSE'97. (Feb. 27-Mar. 1, 1997, San José, CA). Pages 360-364. ACM. 1997.

[Jalics y Golden, 1995] Jalics, P., Golden, D. "*A Profile of Undergraduate Computer Science Curricula*". Computer Science Education, 6(2):179-192. November 1995.

[Jalloul, 1999] Jalloul, G. "*Pedagogical Pattern #48. Academic to Industrial Project Link (LINK) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp48.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Jones, 1987] Jones, A. K. "*The Object Model: A Conceptual Tool for Structuring Software*". In Gerald E. Peterson, editor, *TUTORIAL: Object-Oriented Computing*, Volume 2: Implementations. IEEE Computer Society Press, 1987.

[Jones, 1991] Jones, C. B. "*Systematic Software Construction Using VDM*". Prentice-Hall, 1991.

[Kirby et al., 1994] Kirby, M., Life, A., Istance, H., Hole L., Crombie, A. "*HCI Curricula in the UK: What Is Being Taught and What Should Be Taught*". BCS HCI Curriculum WG Interim Report. London: BCS. 1994.

[Kirby et al., 1995] Kirby, M., Life, A., Istance, H., Hole L., Crombie, A. "*HCI Curricula: What Is Being Taught on Computing Courses in the UK*". In Proceedings of Interact '95, Lillehammer, Norway. 1995.

[Knight et al., 1994] Knight, J. C., Prey, J. C., Wulf, W. A. "*Undergraduate Computer Science Education: A New Curriculum Philosophy & Overview*". In Proceedings of the Twenty-fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 155-159. ACM. 1994.

[Knoke y Bagert, 1998] Knoke, P., Bagert, D. "*Graduate Software Engineering Program Survey Results & Evaluation*". Forum for Advancing Software Engineering Education (FASE), 8(9), September 1998. (http://www.cs.ttu.edu/fase/v8n09.txt).

[Kobryn, 1999] Kobryn, C. "*UML 2001: A Standardization Odyssey*". Communications of the ACM, 42(10):29-37. October 1999.

[Kobryn, 2002] Kobryn, C. "*Will UML 2.0 Be Agile or Awkward?*". Communications of the ACM, 45(1):107-110. January 2002.

[Koffman et al., 1984] Koffman, E., Miller, P., Wardle, C. "*Recommended Curriculum for CS1: 1984*". Communications of the ACM, 27(10):998-1001. October 1984.

[Koffman et al., 1985] Koffman, E., Miller, P., Wardle, C. "*Recommended Curriculum for CS2: 1984*". Communications of the ACM, 28(8):815-818. August 1985.

[Konrad et al., 1995] Konrad, M. D., Paulk, Mark C., Graydon, A. W. "*An Overview of SPICE's Model for Process Management*". In Proceedings of the Fifth International Conference on Software Quality, Austin, TX, 23-26 October 1995.

[Lalonde y Pugh, 1990] Lalonde, W. R., Pugh, J. R. "*Inside Smalltalk*". Vol. 1. Prentice-Hall, 1990.

[Lalonde y Pugh, 1991] Lalonde, W. R., Pugh, J. R. "*Inside Smalltalk*". Vol. 2. Prentice-Hall, 1991.

[Layman, 1994] Layman, B. "*ISO-9000 Standards and Existing Quality Models: How They Relate*". American Programmer Review, pp. 9-15. February 1994.

[Le Moigne, 1973] Le Moigne, J. L. "*Les Systemes D'Information dan les Organizations*". Presses Universitaires de France, 1973.

[Lebsanft y Synspace, 1994] Lebsanft, E., Synspace, A. G. "*BOOTSTRAP: Experiences with Europe's Software Process Assesment & Improvement Method*". In Proceedings of the 1st World Congress for Software Quality, 1994.

[Letelier et al., 1998] Letelier, P., Sánchez, P., Pastor, O., Ramos, I. "*OASIS Versión 3: Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain). SPUPV-98.4011, 1998.

[Levine et al., 1991] Levine, L., Pesante, L. H., Dunkle, S. B. "*Technical Writing for Software Engineers*". Curriculum Module, SEI-CM-23, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). November 1991.

[Levy, 1984] Levy, H. "*Capability-Based Computer Systems*". Digital Press, 1984.

[Lewerentz y Rust, 2000] Lewerentz, C., Rust, H. "*Are Software Engineers True Engineers?*". Annals of Software Engineering, 10:311-328. 2000.

[Lewis, 1994] Lewis, T.G. "*Where Is Computing Headed?*". IEEE Computer, 27(8):59-63. August 1994.

[Lilly, 1996] Lilly, S. "*Patterns for Pedagogy*". Object Magazine, 5(8):93-96. January 1996.

[Liskov, 1993] Liskov, B. "*A History of CLU*". History of Programming Languages - The Second ACM SIGPLAN Conference on History of Programming Languages. (April 20 - 23, 1993, Cambridge United States). Pages 133-147. ACM Press, 1993.

[Liskov y Zilles, 1977] Liskov, B., Zilles, S. "*An Introduction to Formal Specifications of Data Abstractions*". Current Trends in Programming Methodology: Software Specification and Design. Vol. 1. Prentice-Hall, 1977.

[Longenecker y Feinstein, 1991] Longenecker, Jr. H. E., Feinstein, D L. (Editors). "*IS'90 The DPMA Model Curriculum for a Four Year Undergraduate Degree for the 1990s*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1991.

[Longenecker et al., 1994] Longenecker, Jr. H. E., Fournier, R., Reaugh, W. R., Feinstein D. L. (Editors). "*IS'94 The DPMA Two Year Model Curriculum for IS Professionals*". DPMA Data Processing Management Association Model Curricula for the 1990s. 505 Busee Highway, Park Ridge, IL. 60068. 1994.

[Magel et al., 1981] Magel, K. I., Austing, R. H., Berztiss, A., Engel, G. L., Hamblen, J. W., Hoffmann, A. A. J., Mathis, R. "*Recommendations for Master's Level Programs in Computer Science*". Communications of the ACM, 24(3):115-123. March 1981.

[Manns, 1999] Manns, M. L. "*Pedagogical Pattern #8. Lab-Discussion-Lecture-Lab (LDLL) Pattern*". Version 1.0. In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp8.htm. [Última vez visitado, 20/8/1999]. July 1999.

[MAP, 1995] Ministerio de las Administraciones Públicas. "*Metodología Métrica 2.1*". Volúmenes 1-3. Editorial Tecnos, 1995.

[MAP, 2001a] Ministerio de Administraciones Públicas. "*MÉTRICA 3.0*", Volúmenes 1-3. Ministerio de Administraciones Públicas, 2001.

[MAP, 2001b] Ministerio de Administraciones Públicas. "*MÉTRICA. Versión 3. Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información*". http://www.map.es/csi/metrica3/. [Última vez visitado, 20-10-2001]. 2001.

[Marqués, 1995] Marqués Corral, J. M. "*Jerarquías de Herencia en el Diseño de Software Orientado al Objeto*". Tesis Doctoral. Facultad de Ciencias, Universidad de Valladolid, 1995.

[McDermid, 1991] McDermid, J. (Editor). "*The Software Engineer's Reference Book*". Butterworth Heinemann, 1991.

[Mead y Turner, 1998] Mead, N. R., Turner, A. J. "*Current Accreditation, Certification, and Licensure Activities Related to Software Engineering*". Annals of Software Engineering, 6:167-180. 1998.

[Meyer, 1990] Meyer, B. "*Introduction to the Theory of Programming Languages*". Prentice Hall, 1990.

[Meyer, 1996] Meyer, B. "*Teaching Object Technology*". IEEE Computer, 29(12):117. December 1996.

[Meyer, 1997] Meyer, B. "*Object Oriented Software Construction*". 2nd Edition. Prentice Hall, 1997.

[Meyer, 2001] Meyer, B. "*Software Engineering in the Academy*". IEEE Computer, 34(5):28-35, May 2001.

[Michael, 2000] Michael, M. "*Fostering and Assessing Communication Skills in the Computer Science Context*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 119-123. ACM Press. 2000.

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

[Mulder y Dalphin, 1984] Mulder, M. C., Dalphin, J. "*Computer Science Program Requirements and Accreditation – An Interim Report of the ACM/IEEE Computer Society Joint Task Force*". Communications of the ACM, 27(4):330-335, April 1984.

[Mynatt, 1990] Mynatt, B. "*Software Engineering with Students Project Guidance*". Prentice Hall International, 1990.

[Naur y Randell, 1969] Naur, P., Randell, B. (Editors). "*Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 October 1968*". Brussels: Scientific Affairs Division, NATO. January 1969.

[Ng, 1999] Ng, C. "*Experience Qualification Guidelines for Registration through the Computer and Software Engineering Disciplines*". Draft. Technical Report. Association of Professional Engineers y Geoscientists, British Columbia, March 1999.

[Norman, 1988] Norman, D. A. "*The Psychology of Everyday Things*". New York: Basic Books, Inc. 1988.

[Northrop, 1992] Northrop, L. M. "*Finding an Educational Perspective for Object-Oriented Development*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 245-249. ACM. 1992.

[Nunamaker, 1981] Nunamaker, J. F. (Editor). "*Educational Programs in Information Systems. A Report of the ACM Curriculum Committee on Information Systems*". Communications of the ACM, 24(3):124-133. March 1981.

[Nunamaker et al., 1982] Nunamaker, J. F., Couger, J. D., Davis, G. B. "*Information System Curriculum Recommendations for the 80s: Undergraduate and Graduate Programs*". Communications of the ACM 25(11):781-805. November 1982.

[OMG, 1998] OMG. "*OMG Unified Modeling Language Specification. Version 1.2*". Object Management Group Inc. ftp://ftp.omg.org/pub/docs/ad/98-12-02-pdf. July 1998.

[OMG, 1999] OMG. "*OMG Unified Modeling Language Specification. Version 1.3*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. [Última vez visitado, 31/12/2001]. June 1999.

[OMG, 2000a] OMG. "*OMG UML 2.0 Infraestructure RFP*". Document ad/00-09-01. September 2000.

[OMG, 2000b] OMG. "*OMG UML 2.0 Superstructure RFP*". Document ad/00-09-02. September 2000.

[OMG, 2000c] OMG. "*OMG UML 2.0 OCL RFP*". Document ad/00-09-03. September 2000.

[OMG, 2001a] OMG. "*OMG UML 2.0 Diagram Interchange RFP*". Document ad/01-02-09. February 2001.

[OMG, 2001b] OMG. "*OMG Unified Modeling Language Specification. Version 1.4*". Object Management Group Inc. http://www.celigent.com/omg/umlrtf/artifacts.htm. [Última vez visitado, 01/09/2001]. September 2001.

[Orr, 1977] Orr, K. T. "*Structured System Development*". Yourdon Press, 1977.

[Orr, 1981] Orr, K. T. "*Structured Requirements Definition*". Ken Orr & Associates, 1981.

[Osborne, 1992] Osborne, M. "*The Role of Object-Oriented Technology in the Undergraduate Computer Science Curriculum*". In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA'92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 303-308. ACM. 1992.

[Pancake, 1995] Pancake, C. M. "*The Promise and the Cost of Object Technology: A Five Years Forecast*". Communications of the ACM 38(10):32-49. October 1995.

[Parnas, 1972] Parnas, D. L. "*On the Criteria To Be Used in Decomposing Systems into Modules*". Communications of the ACM, 15(12):1053-1058. December 1972.

[Parnas, 1990] Parnas, D. L. "*Education for Computing Professionals*". IEEE Computer, 23(1):17-22, January 1990.

[Parnas, 1998] Parnas, D. L. "*Software Engineering Programmes Are Not Computer Science Programmes*". Annals of Software Engineering, 6(1):19-37, January 1998.

[Parnas, 2001] Parnas, D. L. "*Why Software Developers Should Be Licensed*". Engineering Dimensions, 36-39. May-June 2001.

[Parnas y Weiss, 1987] Parnas, D. L., Weiss, D. M. "*Active Design Reviews: Principles and Practices*". Journal of Systems and Software, 7(4): 259-265, December 1987.

[Parrish et al., 1998] Parrish, A., Borie, R., Cordes, D., Dixon, B., Hale, D., Hale, J., Jackson, J., Sharpe, S. "*Computer Engineering, Computer Science and Management Information Systems: Partners in a Unified Software Engineering Curriculum*". In Proceedings of the 11th Conference on Software Engineering Education & Training – CSEET'98. (February 22-25, 1998. Atlanta, GA – USA). Pages 67-75. IEEE Computer Society. 1998.

[Pastor et al., 1997] Pastor, O., Insfrán, E., Pelechano, V., Romero, J., Merseguer, J. "*OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods*". In Advanced Information Systems Engineering (9th International Conference, CAiSE97 – Barcelona, Catalonia, Spain, June 1997). A. Olivé, J. A. Pastor (editors). Pages 145-158. Lecture Notes in Computer Science – LNCS 1250. Springer-Verlag. 1997.

[Pastor y Ramos, 1995] Pastor, O., Ramos. I. "Oasis 2.1.1: *A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach*". Servicio de Publicaciones UPV, Universidad Politécnica de Valencia. Valencia (Spain) SPUPV-95.788, 1995.

[Paulk et al., 1993a] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model for Software, Version 1.1*" Technical Report CMU/SEI-93-TR-24, Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA), February 1993.

[Paulk et al., 1993b] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V. "*Capability Maturity Model, Version 1.1*". IEEE Software, 10(4):18-27. July 1993.

[Pham, 1997] Pham, B. "*The Changing Curriculum of Computing and Information Technology in Australia*". In Proceedings of the Second Australasian Conference on Computer Science Education, ACSE '97. (July 2-4, 1997, The Univ. of Melbourne, Australia). ACM. 1997.

[Piattini, 1994] Piattini Velthuis, M. G. "*Definición de una Metodología para el Desarrollo de Bases de Datos Orientadas al Objeto Fundamentadas en Extensiones del Modelo Relacional*". Tesis Doctoral. Facultad de Informática, Universidad Politécnica de Madrid. 1994.

[Piattini et al., 1996] Piattini, M. G., Calvo-Manzano, J. A., Cervera, J., Fernández, L. "*Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*". Ra-ma, 1996.

[Pollock, 2001] Pollock, L. "*Integrating an Intensive Experience with Communications Skills Development into a Computer Science Course*". In Proceedings of the Thirty Second SIGCSE Technical Symposium on Computer Science Education, SIGCSE'2001. (Charlotte, NC – USA, 2001). Pages 287-291. ACM, 2001.

[Pressman, 1987] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 2nd edition. McGraw Hill, 1987.

[Pressman, 1992] Pressman, R. S. "*Software Engineering. A Practitioner's Approach*". 3rd Edition. McGraw Hill, 1992.

[Pressman, 1997] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 4th Edition. McGraw Hill, 1997.

[Pressman, 2000] Pressman, R. S. "*Software Engineering: A Practitioner's Approach – European Adaptation*". 5th Edition. McGraw-Hill, 2000.

[Prieto y Victory, 1999] Prieto, M., Victory, P. "*Pedagogical Pattern #20. Identity Pattern*". Version 1.0. http://www-lifia.info.unlp.edu.ar/ppp/pp20.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Proto-Patterns, 1999] "*The Pedagogical Patterns Project. Successes in Teaching Object-Technology (PROTO-PATTERNS)*". http://www-lifia.info.unlp.edu.ar/ppp/index.html. [Última vez visitado, 20/8/1999]. July 1999.

[Ralston, 1984] Ralston, A. "*The First Course in Computer Science Needs a Mathematics Corequisite*". Communications of the ACM, 10(27):1002-1005. October 1984.

[Ralston y Shaw, 1980] Ralston, A., Shaw, M. "*Curriculum '78 – Is Computer Science Really that Unmathematical?*". Communications of the ACM, 23(2):67-70. February 1980.

[Ramamoorthy y Sheu, 1988] Ramamoorthy, C., Sheu, P. "*Object-Oriented Systems*". IEEE Expert, 3(3). Fall 1988.

[Rand, 1979] Rand, A. "*Introduction to Objectivist Epistemology*". New American Library, 1979.

[Randell, 1998] Randell, B. "*Memories of the NATO Software Engineering Conference*". In *Anecdotes Column*, James E. Tomayko (Editor). IEEE Annals of the History of Computing, 20(1):51-54. January-March 1998.

[Rational et al., 1997] Rational Software Corporation, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime. Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam. "*UML Proposal to the Object Management. In Response to the OA&D Task Force's RFP-1*". UML 1.1 Referece Set 1.1. 1 September 1997.

[Rayside y Campbell, 2000] Rayside, D., Campbell, G. T. "*Aristotle and Object-Oriented Programming: Why Modern Students Need Traditional Logic*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 237-244. ACM Press. 2000.

[Redwine, 1985] Redwine, S. T. "*Software Technology Maduration*". In Proceedings of the 1st International Conference on Software Engineering, Washington D. C. (USA). IEEE, 1985.

[Reenskaug et al., 1996] Reenskaug, T., Wold, P., Lehne, O. A. "*Working with Objects. The OOram Software Engineering Method*". Manning Publications Co./Prentice Hall, 1996.

[Reynolds y Fox, 1996] Reynolds, C., Fox, C. "*Requirements for a Computer Science Curriculum Emphasizing Information Technology Subject Area: Curriculum Issues*". In Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE'96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 247-251. ACM. 1996.

[Rivas et al., 1997] Rivas, E., DeSilva, D., McDaniel, T., Atkinson, C. "*Object Analysis and Design Facility*". Response to OMG/OA&D RFP-1. Version 1.0. Platinum Technology, Inc. January 1997.

[Roberts et al., 1999] Roberts, E., Shackelford, R., LeBlanc, R., Denning, P. J. "*Curriculum 2001: Interim Report from the ACM/IEEE-CS Task Force*". In Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, SIGCSE'99. (March 24-28, 1999, New Orleans, LA - USA). Pages 343-344. ACM. 1999.

[Robillard y Robillard, 1998] Robillard, P. N., Robillard, M. "*Improving Academic Software Engineering Projects: A Comparative Study of Academic and Industry Projects*". Annals of Software Engineering, 6:343-363. 1998.

[Rosson y Carroll, 1996] Rosson, M. B., Carroll, J. M. "*Scaffolded Examples for Learning Object-Oriented Design*". Communications of the ACM, 39(4):46-47. April 1996.

[Rumbaugh, 1987] Rumbaugh, J. "*Relations as Semantic Constructs in an Object-Oriented Language*". In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications – OOPSLA'87. (October 4 - 8, 1987, Orlando, FL USA). Pages 466-481. ACM Press, 1987.

[Rumbaugh, 1988] Rumbaugh, J. "*Controlling Propagation of Operations Using Attributes on Relations*". In Proceedings of Conference on Object Oriented Programming Systems Languages and Applications – OOPSLA'88. (September 25 - 30, 1988, San Diego, CA USA). Pages 285-296. ACM Press, 1988.

[Rumbaugh, 1996] Rumbaugh, J. "*OMT Insights. Perspectives on Modeling from the Journal of Object-Oriented Programming*". SIGS Books Publications, 1996.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. "*Object-Oriented Modeling and Design*". Prentice-Hall, 1991.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., Booch, G. "*The Unified Modeling Language Reference Manual*". Object Technology Series. Addison-Wesley, 1999.

[SAC, 1967] President's Science Advisory Commission. "*Computers in Higher Education*". Washington DC: The White House. February 1967.

[Sánchez et al., 1998] Sánchez, P., Letelier, P., Ramos, I., Pastor, O. "*OASIS 3.0: Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*". Actas de las III Jornadas de Trabajo MENHIR, Moros, B. y Sáez, J. (editores) (13 y 14 de noviembre de 1998, Murcia - Spain). Páginas 63-74. 1998.

[Schmauch, 1994] Schmauch, C. "*ISO9000 for Software Developers*". IEEE Computer Society Press, 1994.

[Scragg et al., 1994] Scragg, G., Baldwin, D., Koomen, H. "*Computer Science Needs and Insight-Based Curriculum*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 150-154. ACM. 1994.

[Sernadas et al., 1989] Sernadas, A., Fiadeiro, J., Sernadas, C., Eric, H.-D. "*The Basic Building Blocks of Information Systems*". In *Information Systems Concepts*. North Holland, Namur, 1989.

[Shackelford y LeBlanc, 1994] Shackelford, R. L., LeBlanc, R. J. "*Integrating 'Depth First' and 'Breadth First' Models of Computing Curricula*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE'94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 6-10. ACM. 1994.

[Sharp et al., 2000] Sharp, H., Robinson, H., Woodman, M. "*Software Engineering: Community and Culture*". IEEE Software, 17(1):40-47. January/February 2000.

[Shaw, 1984] Shaw, M. "*Abstraction Techniques in Modern Programming Languages*". IEEE Software, 1(4). October 1984.

[Shaw, 1985] Shaw, M. (Editor). "*The Carnegie-Mellon Curriculum for Undergraduate Computer Science*". Springer-Verlag, 1985.

[Shaw, 1990] Shaw, M. "*Prospects for an Engineering Discipline of Software*". IEEE Software, 7(6):15-24. November 1990.

[Shaw, 1992] Shaw, M. "*We Can Teach Software Better*". Computing Research News, 4(4):2-12, September 1992.

[Shaw, 1998] Shaw, M. "*A Profession of Software Engineering: Is There a Need? YES; Are We Ready? NO*". In Proceedings of the ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering, SIGSOFT'98. (Nov. 3-5, 1998, Lake Buena Vista, FL). Pages 207-208, ACM Press, 1998.

[Shaw, 2000] Shaw, M. "*Software Engineering Education: A Roadmap*". In Proceedings of the International Conference on Software Engineering - The Future of Software Engineering. (June 4 - 11, 2000, Limerick Ireland). Pages 371-380. ACM Press, 2000.

[Shaw y Garlan, 1996] Shaw, M., Garlan, D. "*Software Architecture: Perspectives on a Emerging Discipline*". Prentice-Hall, 1996.

[Shaw y Tomayko, 1991] Shaw, M., Tomayko, J. E. "*Models for Undergraduate Project Courses in Software Engineering*". Technical Report CMU/SEI-91-TR-10 (ESD-91-TR-10), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August 1991.

[Shlaer y Mellor, 1992] Shlaer, S., Mellor, S. "*Object Lifecycles: Modeling the World in States*". Yourdon Press, 1992.

[Simons et al., 1991] Simons, B., Frailey, D. J., Turner, A. J., Zweben, S. H., Denning, P. J. "*An ACM Response: The Scope and Directions of Computer Science*". Communications of the ACM, 34(10):121-131. October 1991

[SIS, 1987] SIS. "*Data Processing - Programming Languages — SIMULA*". Standardiseringskommissionen i Sverige (Swedish Standards Institute), Svensk Standard SS 63 61 14, 20 May 1987.

[Smith y Tockey, 1988] Smith, M., Tockey, S. "*An Integrated Approach to Software Requirements Definition Using Objects*". Seattle, WA: Boeing Commercial Airplane Support Division, 1988.

[Sommerville, 1985] Sommerville, I. "*Software Engineering*". 2nd edition. Addison-Wesley, 1985.

[Sommerville, 1989] Sommerville, I. "*Software Engineering*". 3rd edition. Addison-Wesley, 1989.

[Sommerville, 1996] Sommerville, I. "*Software Engineering*". 5th edition. Addison-Wesley, 1996.

[Sommerville, 2001] Sommerville, I. "*Software Engineering*". 6th edition. Addison-Wesley, 2001.

[Spivey, 1989] Spivey, J. M. "*The Z Notation. A Reference Manual*". International Series in Computer Science. Prentice-Hall International, 1989.

[Stevens et al., 2000] Stevens, K. T., Henry, J., Lawhead, P. B., Lewis, J., Bland, C., Peters, M. J. "*Using Large Projects in a Computer Science Curriculum*". In Proceedings of the Thirty-First SIGCSE Technical symposium on Computer Science Education – SIGCSE'00. (Austin, TX, USA – March 2000). Pages. 399-400. ACM Press. 2000.

[Stillings et al., 1987] Stillings, N., Feinstein, M., Garfield, J., Rissland, E., Rosenbaum, D., Weisler, S., Baker-Ward, L. "*Cognitive Science: An Introduction*". The MIT Press, 1987.

[SUN, 2002] SUN Microsystems. "*The Java Tutorial. A Practical Guide for Programmers*". http://java.sun.com/docs/books/tutorial/index.html. [Última vez visitado, 12/04/2002]. March 2002.

[Sutcliffe, 1995] Sutcliffe, A. "*Human Computer Interaction*". Macmillan, 1995.

[Sutcliffe y Maiden, 1998] Sutcliffe, A., Maiden, N. "*The Domain Theory for Requirements Engineering*". IEEE Transactions on Software Engineering, 24(3): 174-196. March 1998.

[Toffler y Toffler, 1990] Toffler, A., Toffler, H. "*The Third Wave*". Bantam Books, 1990.

[Tardieu et al., 1986] Tardieu, H., Rochfeld, A., Coletti, R. "*La Méthode Merise. Tomo 1. Principes et Outils*", Editions d'Organisation, 1986.

[Tewari y Friedman, 1992] Tewari, R., Friedman, F. "*The Impact of Object-Oriented Software Engineering in the Introductory Computer Science Curriculum*". In addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) - OOPSLA '92. (Oct. 18-22, 1992, Vancouver, British Columbia, Canada). Pages 289-292. ACM. 1992.

[Tobin y Tippins, 1993] Tobin, K., Tippins, D. "*Constructivism as a Referent for Teaching and Learning*". In Tobin, K. (ed.). *The Practice of Constructivism in Science Education*. Pages 3-21. AAAS Press, Washington D. C. 1993.

[Tomayko, 1987] Tomayko, J. E. "*Teaching a Project-Intensive Introduction to Software Engineering*". Technical Report CMU/SEI-87-TR-20 (ESD-TR-87-171), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, PA 15213 (USA). August 1987.

[Tomayko, 1998] Tomayko, J. E. "*Forging a Discipline: An Outline History of Software Engineering Education*". Annals of Software Engineering, 6:3-18, 1998.

[Tomayko, 2000] Tomayko, J. E. "*A Historian's View of Software Engineering*". In Proceedings of the Thirteenth Conference on Software Engineering and Training. (6-8 March, 2000. Austin, Texas (USA)). Pages 101-108. IEEE Press, 2000.

[Tucker et al., 1991a] Tucker, A. B., Barnes, B. H., Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*Computing Curricula 1991*". ACM Press. February 1991.

[Tucker et al., 1991b] Tucker, A. B., Barnes, B. H., Aiken, R. M., Barker, K., Bruce, K. B., Cain, J. T., Conry, S. E., Engel, G. L., Epstein, R. G., Lidtke, D. K., Mulder, M. C., Rogers, J. B., Spafford, E. H., Turner, A. J. "*A Summary of the ACM/IEEE-CS Joint*

Curriculum Task Force Report. Computing Curricula 1991". Communications of the ACM, 34(6):68-84. June 1991.

[Tucker et al., 1996] Tucker, A. B., Astrachan, O., Bruce, K., Cupper, R., Denning, P., Drysdale, S., Horton, T., Kelemen, C., McGeoch, C., Patt, Y., Proulx, V., Rada, R., Rasala, R., Roberts, E., Rudich, S., Stein, L., Van Loan, C. "*Strategic Directions in Computer Science Education*". ACM Computing Surveys, 28(4):836-845. December 1996.

[Tucker y Barnes, 1991] Tucker, A. B., Barnes, B. H. "*Flexible Design: A Summary of Computing Curricula 1991*". IEEE Computer, 24(11):56-66, November 1991.

[Tucker y Wegner, 1994] Tucker, A. B., Wegner, P. "*New Directions in the Introductory Computer Science Curriculum*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, SIGCSE '94. (March 10-11, 1994, Phoenix, AZ - USA). Pages 11-15. ACM. 1994.

[Tymann et al., 1994] Tymann, P. T., Lea, D., Raj, R. K. "*Developing an Undergraduate Software Engineering Program in a Liberal Arts College*". In Proceedings of the Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education (SIGCSE '94). (March 10-11, 1994, Phoenix, AZ – USA). Pages 276-280. ACM. 1994.

[Vaitkevitchius, 1999] Vaitkevitchius, R. "*Pedagogical Pattern #25. BASE-and-Supplementary-Languages in Lectures (BSLL)*". In [Proto-Patterns, 1999]. http://www-lifia.info.unlp.edu.ar/ppp/pp25.htm. [Última vez visitado, 20/8/1999]. July 1999.

[Vaughn, 2000] Vaughn, R. B. Jr. "*Software Engineering Degree Programs*". Crosstalk, The Journal of Defense Software Engineering, 13(3):7-9. March 2000.

[Walker y Schneider, 1996] Walker, H. M., Schneider, G. M. "*A Revised Model Curriculum for a Liberal Arts Degree in Computer Science*". Communications of the ACM, 39(12):85-95. December 1996.

[Ward, 1989] Ward, P. T. "*How to Integrate Object Orientation with Structured Analysis and Design*". IEEE Software, 6(2):74-82. March/April 1989.

[Ward y Mellor, 1985] Ward, P., Mellor, S. "*Structured Development for Real-Time Systems*". Vols. 1-3. Yourdon Press, 1985.

[Warnier, 1974] Warnier, J. "*Logical Construction of Programs*". Van Nostrand Reinhold, 1974.

[Wasserman, 1996] Wasserman, A. "*Toward a Discipline of Software Engineering*". IEEE Software, 13(6):23-31. November/December 1996.

[WebE, 2001] "*Actas del I Taller sobre Ingeniería del Software Orientada al Web (Web Engineering*". Almagro, Ciudad Real, 22 de noviembre de 2001. http://www.dlsi.ua.es/webe01. [Última vez visitado, 7/1/2002]. 2001.

[Wegner, 1990] Wegner, P. "*Concepts and Paradigms of Object-Oriented Programming*". OOPS Messenger, 1(1). August 1990.

[Weinberg, 1971] Weinberg, G. F. "*The Psychology of Computer Programming*". Van Nostrand Reinhold, 1971.

[West, 1997] West, D. "*Hermeneutic Computer Science*". Communications of the ACM, 40(4):115-116. April 1997.

[Wielinga et al. 1991] Wielinga, B. J., Schreiber, A. T., Breuker, J. A. "*KADS: A Modeling Approach to Knowledge Engineering*". Technical Report ESPRIT Project P5248 KAD-II, 1991.

[Wirfs-Brock et al., 1990] Wirfs-Brock, R., Wilkerson, B., Wiener, L. "*Designing Object-Oriented Software*". Prentice-Hall, 1990.

[Wirfs-Brock y Johnson, 1990] Wirfs-Brock, R., Johnson, R. E. "*Surveying Current Research in Object-Oriented Design*". Communications of the ACM, 33(9):104-124. September 1990.

[Woodman et al., 1996] Woodman, M., Davies, G., Holland, S. "*The Joy of Software – Starting with Objects*". In Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education - SIGCSE '96. (Feb. 15-18, 1996, Philadelphia, PA, USA). Pages 88-92. ACM. 1996.

[Yonezawa y Tokoro, 1987] Yonezawa, A., Tokoro, M. "*Object-Oriented Concurrent Programming: An Introduction*". In *Object-Oriented Concurrent Programming*. Cambridge, MA: The MIT Press, 1987.

[Yourdon, 1989] Yourdon, E. "*Modern Structured Analysis*". Prentice Hall, 1989.

[Yourdon Inc., 1993] Yourdon Inc. "*Yourdon™ Systems Method. Model-Driven Systems Development*". Prentice Hall International Editions, 1993.

[Yourdon y Constantine, 1975] Yordon, E., Constantine, L. "*Structured Design*". 1st edition. Prentice Hall, 1975.

[Yourdon y Constantine, 1979] Yordon, E., Constantine, L. "*Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*". Yourdon Press, 1979.

[Yourdon y Constantine, 1989] Yordon, E., Constantine, L. "*Structured Design*". 2nd edition. Yourdon Press, 1989.

[Zelkovitz et al., 1979] Zelkovitz, M. V., Shaw, A. C., Gannon, J. D. "*Principles of Software Engineering and Design*". Prentice-Hall, 1979.