La Programación en los planes de estudio

Jesús García Molina
Facultad de Informática, Universidad de Murcia
jmolina@um.es

Resumen

Este documento analiza la organización de los contenidos de programación en los planes de estudio. Primero se analiza cómo se organizan los contenidos de programación en el *Computing Currícula 2001* de ACM/IEEE, como principal recomendación curricular internacional. Luego se analizan los planes de estudio de varias universidades prestigiosas, siete españolas y cinco extranjeras. A partir de estos dos análisis se establece una definición del área como paso previo a discutir un conjunto de cuestiones que es necesario abordar para el diseño de la parte de un plan de estudios relacionada con la programación. Por último, se esboza cuál podría ser la estructura de un documento destinado a elaborar una recomendación curricular completa del Grado de Ingeniero en Informática para las universidades españolas.

1. Introducción

El Libro Blanco del Título de Grado en Ingeniería en Informática [11] propone unos contenidos formativos comunes (CFC) para el Grado de Informática. Estos CFC no constituyen una verdadera recomendación curricular para el diseño de planes de estudio, puesto que se limitan a una distribución de contenidos obligatorios en varios bloques a los que se asocia una lista de descriptores de materias.

Aunque existen recomendaciones curriculares internacionales de referencia, como el *Computing Curricula 2001* de ACM/IEEE (CC2001) [1], sin duda la más conocida, y los objetivos formativos de un estudiante de informática son similares en cualquier lugar del mundo, una recomendación centrada en el contexto español podría ser de gran utilidad para nuestras universidades. En muchos países, el Estado no regula los contenidos obligatorios de las titulaciones, sino que cada universidad tiene total libertad para diseñar sus planes de estudio que están sujetos al control de una agencia de acreditación. Probablemente, el nuevo marco legislativo español que regulará la adaptación al Espacio Europeo de Educación Superior no establecerá directrices propias para el Grado de Ingeniería en Informática. En ese contexto, una recomendación curricular tiene un gran valor como marco de referencia.

Por ello, la iniciativa de la AENUI, junto a otras asociaciones, en esta edición del SINDI puede resultar valiosa para arrancar un proceso que culmine con una recomendación curricular para el grado de Informática en las universidades españolas. Se trata de un proyecto ambicioso pero factible que debe desarrollarse al margen de los intereses particulares de las áreas de conocimiento, y siguiendo un esquema de trabajo similar al que se sigue en la elaboración de las recomendaciones de ACM e IEEE. Tras el SINDI, se debería preparar una propuesta para cada área temática por parte de un grupo reducido de expertos, y luego otro grupo de expertos se debería encargar de integrar todas las propuestas redactando el documento definitivo.

Con esta idea en mente, el objetivo de este documento es ofrecer un marco de trabajo que propicie y oriente una discusión durante la sesión del SINDI asignada a tal efecto y que permita establecer cómo llegar a definir una propuesta curricular para el área de Programación. No se trata de ningún borrador inicial sino sólo una guía para el debate, tras el cual se debería elegir el grupo de expertos que elaborase la propuesta definitiva.

El presente documento se ha organizado de la siguiente forma. En el siguiente apartado se describe cómo se organiza el área de Programación en el CC2001. Luego se presenta un análisis de cómo se ha incorporado esta área en los planes de estudio de varias universidades (una de EE.UU., cuatro europeas y siete españolas). A continuación, se define el área de

Programación: objetivos formativos y contenidos obligatorios, y en el siguiente apartado se discuten un conjunto de cuestiones que se plantean cuando se diseña la parte relacionada con la programación de un plan de estudios. Por último, se esboza cuál podría ser la estructura del documento definitivo que plasmase la recomendación curricular.

2. El área "Programación" en CC2001

Los CFC del Libro Blanco de Informática se organizan en cuatro categorías: "Fundamentos Científicos", "Contenidos Específicos de la Ingeniería Informática", "Contenidos Generales de la Ingeniería" y "Proyecto Fin de Carrera" [11]. "**Programación**" aparece como una de las cuatro subcategorías existentes dentro de la segunda categoría y sus descriptores son: *Fundamentos y Metodología de la Programación, Algoritmia, Computabilidad, Lenguajes de Programación, Paradigmas de Programación, y Estructuras de Datos*.

Estos descriptores reflejan que los contenidos de "Programación" corresponden a las áreas "Fundamentos de Programación", "Algoritmos y Complejidad" y "Lenguajes de Programación" del CC2001 de ACM/IEEE. Esta correspondencia parece natural y es conforme a los contenidos y objetivos formativos que comúnmente se asigna al conjunto de asignaturas relacionadas con la programación. Aunque no hay que olvidar que algunos aspectos básicos de la programación son incluidos en el área de "Ingeniería del Software" en el CC2001, como son el diseño de programas, los entornos de programación, las técnicas formales o las pruebas.

Conviene señalar que el Libro Blanco plantea un grado para un título generalista de informática, en la línea de la disciplina "Computer Science" de las recomendaciones curriculares de ACM e IEEE [2]. Por lo que en este documento no se tendrá en cuenta que la organización del área de Programación sería distinta si se considerasen otras disciplinas como "*Software Engineering*", "*Computer Engineering*", "*Information Systems*" o "*Information Technology*" consideradas también en esas recomendaciones (en el caso de "*Software Engineering*" prácticamente no habría diferencias).

En vez de comenzar definiendo el área de Programación (contenidos y objetivos formativos), analizaremos antes el CC2001 y veremos cómo se ajustan a esta recomendación los planes de estudio de varias universidades españolas y extranjeras.

En el apéndice A de CC2001 se describen las catorce áreas de conocimiento en las que se organiza el cuerpo de conocimiento de la informática. Cada área se divide en varias unidades temáticas que son consideradas obligatorias (*core*) u optativas. A cada unidad se le asocia unos descriptores de contenidos y unos objetivos de aprendizaje. A cada área (y unidad *core*) se asigna un número de horas mínimo para cubrir la enseñanza de los contenidos obligatorios en un formato de clases magistral tradicional. La Tabla 1 muestra las catorce áreas y entre paréntesis las horas asignadas a cada una de ellas.

Matemática Discreta (43)	Arquitectura y Organización (36)
Fundamentos de Programación (38)	Sistemas Operativos (18)
Algoritmos y Complejidad (31)	Computación centrada en la red (15)
Lenguajes de Programación (21)	Sistemas Inteligentes (10)
Ingeniería del Software (31)	Interacción Persona-Ordenador (8)
Gestión de la información (10)	Cuestiones sociales y profesionales (16)
Computación gráfica y visual (3)	Ciencia Computacional (0)

Tabla 1. Areas de conocimiento en CC2001

Esta asignación de horas permite realizar una estimación del peso de los contenidos relacionados con la programación en relación con las otras áreas. Por ejemplo, se observa que:

- Los contenidos relacionados con el área de "Programación", tal y como se entiende en el Libro Blanco, suponen un 32% de la carga del plan de estudios. Sin duda, una carga bastante elevada.
- El peso de "Fundamentos de Programación" es igual al de "Arquitectura y Organización de Computadores".

- El peso de los contenidos de algoritmia y complejidad es el mismo que el de la ingeniería del software, tres veces el de los sistemas inteligentes y las bases de datos y el doble que el de redes de ordenadores y la Web.
- La programación orientada a objetos (sin contar la enseñanza de los patrones de diseño y el análisis y diseño orientado a objetos) tiene el mismo peso que los sistemas inteligentes y la gestión de la información (bases de datos)
- Por cada crédito de arquitectura y organización de computadores deberían haber 2.5 de programación.
- Por cada crédito de sistemas inteligentes o gestión de información deberían haber nueve de programación.

Las áreas "Fundamentos de Programación" (PF), "Lenguajes de Programación" (PL) y "Algoritmos y Complejidad" (AL) del CC2001 ilustran muy bien, en su conjunto, los conocimientos y destrezas sobre programación que debe aprender un alumno y que constituyen la base para estudiar las asignaturas de ingeniería del software.

La Tabla 2 muestra las unidades obligatorias de cada una de las tres áreas. Entre paréntesis, en vez de las horas mínimas, aparece el porcentaje de carga lectiva con respecto a la carga total de 280 horas que se supone en el CC2001.

PF. Fundamentos de Programación (13,5)
PF1. Construcciones básicas en programación (3,2)
PF2. Algoritmos y resolución de problemas (2,1)
PF3. Estructuras de datos básicas (5)
PF4. Recursión (1,8)
PF5. Programación dirigida por eventos (1,4)
AL. Algoritmos y Complejidad (11)
AL1. Análisis algorítmico básico (1,4)
AL2. Estrategias algorítmicas (2,1)
AL3. Algoritmos básicos (4,3)
AL4. Algoritmos distribuidos (1,1)
AL5. Computabilidad básica (2,1)
PL. Lenguajes de Programación (7,5)
PL1. Introducción a los lenguajes de programación (0,7)
PL2. Máquinas virtuales (0,3)
PL3. Introducción a la traducción (0,7)
PL4. Declaraciones y tipos (1,1)
PL5. Mecanismos de abstracción (1,1)
PL6. Programación orientada a objetos (3,6)

Tabla 2. Áreas de Programación en CC2001

"Fundamentos de Programación" incluye los contenidos que enseñan los conocimientos y destrezas precisos para realizar el diseño, implementación y depuración de un programa organizado mediante los mecanismos de composición secuencial, condicional, iterativa y recursiva, y utilizando las estructuras de datos básicas. Contempla tanto la formación propia de un primer año de programación y una formación más avanzada en técnicas recursivas y estructuras de datos.

"Algoritmos y Complejidad" incluye los contenidos que "permiten seleccionar y aplicar los algoritmos apropiados a un determinado problema, considerando la posibilidad de que no exista tal algoritmo", como son el análisis de algoritmos (eficiencia), los esquemas algorítmicos (divide y vencerás, vuelta atrás, etc.), algoritmos básicos (ordenación, búsqueda binaria, árboles binarios de búsqueda, etc.), algoritmos distribuidos y una introducción a la computabilidad.

“Lenguajes de Programación” tiene como objetivo que el estudiante conozca la existencia de varios paradigmas de programación, sus mecanismos y las ventajas e inconvenientes de cada uno de ellos, prestando especial atención al paradigma orientado a objetos. También incluye contenidos relacionados con los compiladores e intérpretes.

Dentro de estas tres áreas hay algunas unidades optativas que muchos planes de estudio las consideran obligatorias como son “AL6. *Complejidad Algorítmica*”, “PL7. *Programación funcional*“ y “PL9. *Procesadores de Lenguaje*”. En el caso español, la última unidad es obligatoria según las directrices propias de la titulación Ingeniería en Informática que establecen las materias troncales.

Por otra parte, el área de *Sistemas Operativos* incluye como obligatoria la unidad “OS3. Concurrencia” que también puede dar lugar a un curso de programación concurrente, separado de una asignatura de sistemas operativos, junto con contenidos de algoritmos distribuidos.

2.1. Programación e Ingeniería del Software

La enseñanza de la programación no se reduce al conjunto de asignaturas que comúnmente se entiende que conforman el área de Programación, sino que la mayor parte de las asignaturas del plan de estudios están relacionadas, en mayor o menor medida, con la programación, ya que su aprendizaje requiere y potencia los conocimientos y habilidades adquiridos por los alumnos para la construcción de programas. Pero sin duda, es el área de Ingeniería del Software la que tiene una estrecha relación con el área de Programación, de hecho el objetivo de las asignaturas de ingeniería del software es extender la formación en programación que han recibido los alumnos en los primeros cursos para que conozcan los métodos, técnicas y herramientas que permiten el *desarrollo industrial de software*, esto es, el “*desarrollo de sistemas que pueden ser de gran tamaño, destinados a ser utilizados en un entorno de producción, durante un periodo de tiempo que puede extenderse a lo largo del tiempo, y que son manejados por un número de usuarios que puede ser llegar a ser muy grande, y que pueden estar sujetos a un gran número de cambios*” [12].

No existe una nítida línea de separación entre programación e ingeniería del software y hay contenidos que normalmente son considerados parte del área de Ingeniería del Software pero que también podrían serlo del área de Programación, como es el caso del diseño de software, los patrones de diseño, las arquitecturas software o los componentes software. Así, en el área de *Ingeniería del Software* del CC2001 se incluyen las siguientes unidades *core*: “SE1. *Diseño Modular*” (patrones de diseño, componentes, arquitectura software), “SE2. *Utilización de APIs*”, “SE3. Entornos y herramientas software”, “SE6. *Validación del Software*” y “SE7. *Evolución del Software*” y las unidades optativas “SE9. *Desarrollo basado en Componentes*” y “SE10. *Métodos Formales*”. Todas estas unidades podrían ser consideradas parte del área de Programación.

En el Libro Blanco dentro de la subcategoría de “Ingeniería del Software, Sistemas de Información y Sistemas Inteligentes”[1] se incluyen los descriptores “*Desarrollo de software: Procesos, Requisitos, Especificación y Diseño. Gestión de Proyectos, Calidad del Software,*” como los propios de la ingeniería del software. Por tanto, según esta categorización, los contenidos obligatorios del área de Ingeniería del Software corresponderían a las unidades “SE1. Diseño de Software”, “SE4. Procesos software”, “SE5. Requisitos y Especificación de software” y “SE8. Gestión de Proyectos Software” del área de Ingeniería del Software del CC2001.

No cabe duda de que el diseño sobre cómo se organiza el área de Programación en un plan de estudios debe estar totalmente coordinado con el diseño del área de Ingeniería del Software.

[1] Resulta curioso que Ingeniería del Software e Inteligencia Artificial se hayan reunido en la misma subcategoría. Inteligencia Artificial, según el CC2001, es un área pequeña por su reducida carga obligatoria, por lo que no se consideró oportuno separarla en una subcategoría independiente y se integró en el área “más cercana”.

A continuación, analizaremos planes de estudio de algunas universidades. Para cada universidad identificaremos las asignaturas obligatorias del área de *Programación* que incorpora su plan de estudios y mostraremos cómo se organizan mediante una tabla en la que se indica el nombre de la asignatura, el semestre al que pertenece, los créditos (ECTS para las de fuera y LRU para las españolas) y las unidades temáticas de CC2001 más relevantes. Esta tabla es acompañada con comentarios que pretenden destacar los aspectos más significativos.

Se pretende ofrecer una idea general de la situación en cada universidad con el fin de comparar contenidos y objetivos formativos. La descripción del programa docente de una asignatura no puede sustituirse por la asociación de una lista de unidades de CC2001, aunque sí sirve para dar una idea de sus objetivos generales.

3. La programación en algunas universidades extranjeras

Hemos elegido cinco prestigiosas universidades, una americana y cuatro europeas. En las cuatro europeas el grado es de tres años (6 semestres, 180 créditos ECTS) y en la de Carnegie Mellon es de cuatro años (8 semestres). La muestra es muy pequeña y sólo se ha pretendido tener algunas referencias para contrastar con la situación nacional.

School of Computer Science de la Carnegie Mellon University (CMU) (http://www.csd.cs.cmu.edu/education/bscs/#courses)

En la secuencia de cursos que se recomienda a los alumnos encontramos las asignaturas que muestra la Tabla 3.

Asignatura	Semestre	Créditos (ECTS)	Unidades
Programación	1	7.5	PF3, PF4, PF5, AL3, PL6
Algoritmos y Estructuras de datos	2	9	PF3, PF4, AL1, AL3, SE10
Grandes ideas teóricas en informática	2	9	PF2, PF3, AL5, Matemáticas
Programación en C y Unix	3	7	PF1, PF3, PF5, SE3
Principios de Programación	3	9	PL7, SE10
Análisis y diseño de algoritmos	5	9	AL1, AL2, AL3, AL5 (AL9 o AL10 o AL11)

Tabla 3. Asignaturas de Programación en CMU

- "Programación", (que supone una formación previa en PF1 y PF2), es un curso de introducción a la programación orientada a objetos en Java, en el que también se estudian estructuras de datos (listas, pilas, colas, árboles y grafos) y una introducción al análisis de algoritmos que se aplica sobre los algoritmos definidos para las estructuras de datos estudiadas.
- Existe una asignatura específica para enseñar el lenguaje C y Unix como entorno de programación.
- "Principios de Programación" es un curso de programación funcional en el que se enseñan técnicas avanzadas de programación como definición de estructuras de datos infinitas, metaprogramación, construcciones de control de alto nivel, y también métodos formales para especificación y verificación de programas. En la actualidad se utiliza ML como lenguaje de programación.
- Además del curso típico de estructuras de datos y algoritmos, hay un curso avanzado en el que se estudian algoritmos para una variedad de problemas (algoritmos de ordenación y búsqueda, estructuras de datos eficientes, algoritmos para grafos y algoritmos avanzados que en profesor elige entre algoritmos paralelos, algoritmos geométricos, algoritmos aleatorios y criptografía), y completidud NP.

- Llama mucho la atención por su originalidad el curso “Grandes ideas teóricas en informática” en el que se enseña cómo usar ideas teóricas para formular y resolver problemas en informática. Para ello, integra contenidos de matemáticas con técnicas de resolución de problemas y aplicaciones de la informática. Los ejemplos de aplicación son extraídos de las áreas de Algoritmos, Complejidad, Teoría de Juegos, Teoría de la Probabilidad, Teoría de Grafos, Teoría de autómatas, Algebra, Criptografía y Combinatoria. El trabajo del alumno implica tanto prueba matemática como programación.
- En la secuencia recomendada, las únicas asignaturas obligatorias de informática son las de programación y una introducción a los computadores, y en total suponen sólo el 25% de la carga. Materias como sistemas operativos, redes, desarrollo de aplicaciones web, bases de datos y compiladores son consideradas optativas.
- El número total de créditos de asignaturas de programación es 50.5 ECTS.

Facultad de Ciencias e Tecnología, Universidade Nova de Lisboa (UNL)
(http://www.di.fct.unl.pt/ensino/licenciatura/plano_estudos.php)

Asignatura	Semestre	Créditos (ECTS)	Unidades
Introducción a la Programación	1	8	PF1, PF2, PF3, PF4, AL3
Programación Orientada a Objetos	2	6	PL6, PF3, AL3, PF5, SE2
Algoritmos y Estructuras de Datos	3	6	PF3, PF4, AL1, AL2, AL3
Teoría de la Computación	3	6	AL5, AL7, SE10
Lenguajes de Programación	4	9	PL

Tabla 4. Asignaturas de Programación en UNL

- El curso de introducción a la programación del primer semestre está basado en el enfoque imperativo, aunque se utiliza C++ o Java, y es seguido de un curso de programación orientada a objetos con C++ en el segundo semestre.
- No se estudian con profundidad algoritmia y estructuras de datos: la asignatura del segundo semestre es un curso sobre C++ y STL, y la asignatura del tercer semestre sólo tiene 6 créditos.
- “Teoría de la Computación” es una asignatura de informática teórica que incluye computabilidad, teoría de autómatas y verificación formal.
- “Lenguajes de Programación” es un curso típico sobre los mecanismos de los lenguajes en el paradigma imperativo, funcional y orientado a objetos.
- El número total de créditos es 37 ECTS.

Universita degli Studi de Milán (UNIMI)
(http://www.ccdi.unimi.it/it/corsiDiStudio/F49/index.html)

Asignatura	Semestre	Créditos (ECTS)	Unidades
Programación	1	18	PF1, PF2, PF5, PL6
Algoritmos y Estructuras de Datos	3	18	PF3, PF4, AL1, AL2, AL3

Tabla 5. Asignaturas de Programación en UNIMI

- Sólo incluye dos asignaturas semestrales, cada una de las cuales tiene una duración propia de una asignatura anual (18 créditos). La asignatura de programación del primer semestre es un curso de programación orientada a objetos en Java aunque primero se presenta la programación estructurada. La del segundo semestre es un curso de algoritmos y estructuras de datos. Las dos asignaturas se organizan en dos partes: teoría y laboratorio.

- El número total de créditos es 36 ECTS.

Institut für Informatik, Teschniche Universität de Munich (TUM)
(http://studienberatung.in.tum.de/bachelor/informatik/Studienplan_Bachelor_neu.htm)

Asignatura	Semestre	Créditos (ECTS)	Unidades
Introducción a la Informática 1	1	6	PF1, PF2, PF3, PF4, PF5, AL1, AL2, AL3, PL6, PL7, OS3, SE10
Introducción a la Informática 2	2	4	
Laboratorio de Programación	1	6	
Algoritmos y Estructuras de Datos	2	5	PF3, AL1, AL2, AL3
Introducción a la Informática Teórica	4	8	AL5, AL6, AL7, AL8

Tabla 6. Asignaturas de Programación en TUM

- Las dos asignaturas de introducción a la informática, junto con "Laboratorio de Programación", no solo cubren los contenidos habituales de un primer año de programación sino que introducen la programación orientada a objetos, la programación funcional, y la programación concurrente. Sin embargo, el número de créditos parece reducido para enseñar todos esos contenidos.
- Incluye un curso de algoritmia y estructuras de datos pero el número de créditos es muy reducido.
- En "Introducción a la Informática Teórica" se estudia teoría de la complejidad y computabilidad.
- De todos los planes analizados es el que dedica menor número de créditos a la programación, 29 ECTS.

Swiss Federal Institute Technologie de Zurich (ETH)
(http://www.inf.ethz.ch/education/programs/bachelor/)

Asignatura	Semestre	Créditos (ECTS)	Unidades
Introducción a la Programación	1	8	PF1, PF2, PF3, PF4, PF5, PL6, AL1, AL3
Algoritmos y Estructuras de datos	2	7	PF3, AL1, AL2, AL3
Programación de sistemas	3	6	
Informática Teórica	3	7	AL5, AL6, AL7, AL8
Métodos formales y Programación Funcional	4	6	PL7
Arquitectura del software	4	4	SE1, IS6

Tabla 7. Asignaturas de Programación en ETH

- El curso de introducción a la programación está basado en el enfoque orientado a objetos, en el que se utiliza el lenguaje Eiffel. El texto "*Touch of Class*" escrito por el profesor responsable del curso y creador de Eiffel, el Dr. Bertrand Meyer, refleja cómo se organiza la asignatura. Un borrador del texto está disponible en la página del curso (http://se.inf.ethz.ch/teaching/ws2004/0001/english_index.html).
- "Métodos formales y Programación Funcional" es un curso de programación funcional en Haskell y modelado y diseño de sistemas de discretos.
- En "Arquitectura del Software" se enseñan patrones de diseño, agentes y diseño por contrato.
- Incluye un curso de informática teórica en el que se estudia computabilidad y complejidad.
- "Programación de Sistemas" es un curso de programación a bajo nivel que enseña técnicas de optimización y otras para mejorar la robustez y portabilidad.
- El número total de créditos es 38 ECTS.

4. La programación en las universidades españolas

Se han analizado planes de estudio de Ingeniería en Informática de siete universidades españolas. Se ha elegido a cinco facultades de reconocido prestigio, con experiencia docente y grupos de investigación consolidados en la ciencia y tecnología del software (UCM, UMA, UPC, UPV y US), junto con la universidad anfitriona del CEDI (UNIZAR) y la universidad del ponente (UMU). No se han analizado planes de carreras técnicas pero es sabido que se han diseñado como si fuesen primeros ciclos de la titulación superior. Tampoco se han tenido en cuenta las materias optativas, sólo las obligatorias.

El análisis permite adquirir una idea general sobre cómo se organiza el área de *Programación* en la universidad española. Los planes consultados se han diseñado a partir de las directrices propias de la titulación, directrices que eran muy abiertas y que en el área de programación permitían, prácticamente, cualquier tipo de diseño, salvo algunas restricciones como la obligatoriedad de una asignatura sobre "Procesadores de Lenguaje" de 9 créditos.

En la mayoría de universidades se estudian patrones de diseño orientados a objetos dentro de alguna de las asignaturas de ingeniería del software pero no se han incluido estos cursos ya que es difícil calcular la carga crediticia que corresponde al estudio de los patrones y además están incluidos en el área de ingeniería de software en CC2001 y en el Libro Blanco. En alguna de las universidades se estudian componentes software y objetos distribuidos pero tampoco se han considerado esas asignaturas por el mismo motivo.

Universidad Politécnica de Cataluña (UPC) (http://www.fib.upc.edu/fib/infoAca/estudis/EI.html)

Todas las asignaturas son semestrales (en todo el texto nos referiremos a las asignaturas que se imparten en un cuatrimestre como semestrales en vez de cuatrimestrales ya que es el término habitual en el resto de países). Un total de 54 créditos (unos 43 ECTS) para el área de Programación que son distribuidos como muestra la siguiente tabla. Hemos supuesto la distribución por semestres a partir de los prerrequisitos establecidos en el plan de estudios.

Asignatura	Semestre	Créditos	Unidades
Programación I	1	9	PF1, PF2, PF3, PF4, AL1, AL3
Practicas de Programación	2	7.5	SE1, PF4, PF5
Programación y Estructuras de Datos	3	7.5	PL6, PF3
Análisis y Diseño de Algoritmos	3	6	PF3, PF4, AL1, AL2,
Proyecto de Programación	4	6	PL6
Teoría de la Computación	6	9	AL5, AL6
Compiladores	7	9	PL3, PL8

Tabla 8. Asignaturas de Programación en UPC

- En "Programación I" se estudia el diseño iterativo y recursivo, algoritmos de recorrido y búsqueda en secuencias y tablas, y nociones de eficiencia a partir de los algoritmos de ordenación. La asignatura de "Prácticas de Programación" aborda el diseño modular y la profundización en el estudio de la recursividad. Se utiliza el lenguaje Java.
- En "Programación y Estructuras de datos" se introduce el concepto de tipo abstracto de datos y se especifican e implementan algunas estructuras de datos básicas, se estudia la representación dinámica con punteros, y algunos aspectos de un lenguaje de programación orientado a objetos.
- En "Análisis y diseño de algoritmos" se estudia análisis de algoritmos, algunos esquemas algorítmicos y se amplía el estudio de las estructuras de datos.
- La programación orientada a objetos se estudia en la asignatura de "Proyecto de Programación" que permite al alumno integrar sus conocimientos sobre programación adquiridos en los dos primeros años de carrera en el desarrollo de una aplicación de tamaño mediano. Los alumnos forman grupos de tres y son tutelados.
- "Teoría de la Computación" es un curso de computabilidad y complejidad.

Universidad Politécnica de Valencia (UPV)
(http://www.fiv.upv.es/default_c.htm)

Coexisten asignaturas anuales con semestrales. Un total de 49.5 créditos (unos 40 ECTS) para el área de programación que se distribuyen de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Programación	1 y 2	12	PF1, PF2, PF3, PF5, AL1, AL3
Algoritmos y Estructuras de Datos	1 y 2	12	PF3, PF4, AL1, AL2, AL3
Algorítmica	6	4.5	AL5, AL6, AL8
Lenguajes y Paradigmas de Programación	6	6	PL10
Ingeniería de la programación	7	6	PL6 (aparte modelado OO)
Procesadores de Lenguajes	7	9	PL3, PL8

Tabla 9. Asignaturas de Programación en UPV

- Las asignaturas de introducción a la programación y de algoritmos y estructuras de datos son anuales. El lenguaje utilizado en la primera es C y en la segunda C++.
- El plan de estudios incluye una asignatura denominada "Metodología y Tecnología de la Programación" que no se ha considerado del área de Programación ya que su objetivo principal es la enseñanza del análisis y diseño estructurado.
- En "Algorítmica" se profundiza en aspectos sobre algoritmos: programación dinámica, ramificación y poda, algoritmos probabilísticos, complejidad y computabilidad.
- Incluye un curso en el que se comparan paradigmas de programación y se estudia programación lógica, funcional y concurrente.
- De la asignatura anual "Ingeniería de Programación" sólo se ha considerado la parte de programación orientada a objetos y desarrollo con Java, pero no la de análisis y diseño orientado a objetos, por lo que se le ha asignado una carga de 6 de los 12 créditos que tiene asignados.

Universidad de Málaga (UMA)
(**http://www.informatica.uma.es/2_8.html**)

Todas las asignaturas son semestrales. Un total de 66 créditos (unos 53 ECTS) en el área de programación que se distribuyen de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Elementos de Programación	1	7.5	PF1, PF2, PF3
Metodología de la Programación	2	6	PF3, PF4, PF5, AL3
Laboratorio de Programación	2	4.5	
Tipos abstractos de datos	3	6	SE10
Análisis y diseño de algoritmos	4	6	PF4, AL1, AL2, AL3
Laboratorio de Tecnología de Objetos	4	6	PL6, SE2, SE3
Lenguajes de Programación	5	6	PL10
Programación Concurrente	5	6	OS3
Programación Declarativa	5	9	PL7
Procesadores de Lenguajes	7	9	PL3, PL8

Tabla 10. Asignaturas de Programación en UMA

- Primer año de programación con una organización típica en dos cursos semestrales. En el primero se utiliza un pseudolenguaje durante el diseño de los algoritmos y luego se implementan en C++ a partir de la mitad del curso. Hay clases de problemas con grupos reducidos para resolver problemas algorítmicos. Se enseña el diseño descendente. En el segundo semestre se estudia diseño modular, ficheros, recursividad y estructuras de datos enlazadas. Ambos cursos se complementan con "Laboratorio de Programación", curso totalmente práctico destinado a que el alumno desarrolle programas.

- En “Análisis y Diseño de Algoritmos” se estudia la complejidad algorítmica, se presentan esquemas algorítmicos, algoritmos de ordenación y búsqueda y se profundiza en el estudio de la recursividad con transformaciones de algoritmos recursivos a iterativos.
- En “Laboratorio de Tecnología de Objetos” se enseñan los conceptos de la orientación a objetos de forma independiente del lenguaje y se utiliza Java para la parte práctica.
- Se incluye un curso de programación concurrente y distribuida cuyo objetivo es enseñar los mecanismos de comunicación y sincronización tradicionales para que el alumno sea capaz de construir programas concurrentes para sistemas de memoria compartida y distribuida.
- “Lenguajes de Programación” es un curso sobre semántica de los lenguajes de programación.
- Se incluye un curso de programación declarativa que introduce tanto la programación lógica y funcional, a través de Prolog y Haskell, respectivamente.

Universidad de Sevilla (US)
(http://www.eii.us.es/ingenieria.php)

Todas las asignaturas son semestrales. Un total de 60 créditos (unos 48 ECTS) en el área de programación, que se distribuyen de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Introducción a la Programación I	1	7,5	PF1, PF2, PF3
Introducción a la Programación II	2	7,5	PL6, SE1, PF5
Análisis y diseño de algoritmos	3	7,5	PF4, AL2
Teoría de la Computabilidad	3	4.5	AL5, AL6
Estructuras de Datos y Algoritmos	4	7,5	PF3, PF4, AL1, AL3
Programación Declarativa	5	4.5	
Técnicas de Programación de Bajo Nivel	6	6	AR3, AR5, AR6
Procesadores de Lenguajes I	7	4.5	PL3, PL8
Procesadores de Lenguajes II	8	4.5	
Ingeniería del Software II	8	6	SE1, SE3, SE6

Tabla 11. Asignaturas de Programación en US

- Se utiliza C en el curso de introducción a la programación del primer semestre. La asignatura del segundo semestre es un curso de programación orientada a objetos en Java.
- “Teoría de la Computabilidad” es un curso sobre complejidad algorítmica y computabilidad
- Se incluye una asignatura sobre programación declarativa en la que se enseña programación lógica en Prolog y una introducción a la programación funcional.
- Una de las pocas universidades españolas en las que se imparte como obligatorio un curso de técnicas de programación de bajo nivel, similar al del ETH, cuyo objetivo es que el alumno conozca cómo afectan las características del hardware al software para su aplicación en áreas tales como desarrollo de sistemas operativos, control de procesos o sistemas de tiempo real.

Universidad Complutense (UCM)
(http://www.fdi.ucm.es/Guia_Docente/Prog_asignatura.asp?titu=3&fdicurso=2006-2007)

Coexisten asignaturas anuales con semestrales. Un total de 78 créditos (unos 62 ECTS) del área de programación que se distribuyen de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Introducción a la Programación	1, 2	9	PF1, PF2, PF3, PF5. AL3, SE3
Laboratorio de Programación 1	2	4.5	
Estructuras de Datos y de la Información	2, 3	15	PF3, PF4, AL1, AL3, SE10
Programación Orientada a Objetos	3	4.5	PL6, PF5, SE1

Laboratorio de Programación 2	1, 2	9	PL6, SE1, SE2, SE3
Metodología y Tecnología de la Programación	5, 6	12	PF4, AL2, AL5, AL6, SE10
Programación Funcional	5	4.5	PL7
Programación Lógica	5	4.5	
Laboratorio de Programación 3	6	6	PL6, SE1, SE2, SE3,
Procesadores de Lenguajes	7	9	PL3, PL8

Tabla 12. Asignaturas de Programación en UCM

- Existen asignaturas que no incluyen parte práctica sino que está se imparte en una asignatura de "laboratorio".
- "Introducción a la Programación" es un curso de Pascal con sólo clases de teoría y ejercicios. En "Laboratorio de Programación 1" se realizan las prácticas de programación en Pascal.
- En "Estructuras de Datos y de la Información" se enseña especificación de algoritmos iterativos y recursivos, derivación de algoritmos iterativos y recursivos, análisis de la complejidad de algoritmos, especificación formal de tipos abstractos de datos e implementación de estructuras de datos. Es un curso de enseñanza presencial teórica de 5 horas semanales basado en el enfoque que se plasma en el libro [15]
- "Programación Orientada a Objetos" es un curso teórico de orientación a objetos independiente del lenguaje.
- "Laboratorio de Programación 2" es un curso práctico de programación orientada a objetos en C++ a nivel medio.
- En "Metodología y Tecnología de la Programación" se enseña análisis avanzado de la complejidad de algoritmos, transformación de algoritmos recursivos a iterativos; esquemas algorítmicos, y complejidad de problemas. Es un curso teórico con clases de ejercicios.
- "Laboratorio de Programación 3" es un curso de desarrollo de aplicaciones en Java de nivel avanzado y en el que se implementan los esquemas algorítmicos estudiados en la anterior asignatura.
- Incluye un curso teórico de programación funcional en Haskell y otro de programación lógica en Prolog. En ambos cursos la realización del trabajo práctico es opcional

Universidad de Zaragoza (UNIZAR)

(http://ebro3.unizar.es:8080/acad/FMPro?-DB=w_titulaciones.fp5&-lay=cgi&-format=titulacion.htm&-error=error2.htm&id_titulacion=3&-Max=25&-SortField=Orden&-Find)

Todas las asignaturas son semestrales. Un total de 49.5 créditos (unos 40 ECTS) en el área de programación distribuidos de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Introducción a la Programación	1	7,5	PF1, PF2, PF3, AL3
Metodología de Programación	2	7,5	PF4, AL1, SE10
Estructuras de Datos y Algoritmos	3	7,5	PF3, PF4, PF5, AL2, AL3, SE10
Laboratorio de Programación	5	6	
Modelos Abstractos de Cálculo	5	4.5	AL5, AL6
Lenguajes de Programación	6	7.5	PL
Compiladores I	7	4.5	PL3, PL8
Compiladores II	8	4.5	

Tabla 13. Asignaturas de Programación en UNIZAR

- En el segundo cuatrimestre del primer año se imparte un curso sobre verificación formal de programas, lo que la diferencia de la mayoría de universidades españolas.
- En "Introducción a la Programación" se estudian los tipos básicos y estructurados, diseño descendente, ficheros secuenciales, algoritmos de recorrido y búsqueda en secuencias, y algoritmos de ordenación y mezcla de tablas. En "Metodología de la Programación" se estudia verificación formal de algoritmos, diseño recursivo y complejidad algorítmica. El lenguaje de programación utilizado en ambas asignaturas es Ada.

- En “Estructuras de Datos y Algoritmos” se estudia especificación formal con tipos abstractos de datos.
- Existe una asignatura en la cual los alumnos forman grupos y desarrollan un proyecto de programación desde la especificación hasta la documentación. Cada grupo mantiene una reunión semanal con el profesor de una hora de duración. No hay clases teóricas.
- En “Lenguajes de Programación“ se estudian los conceptos y mecanismos de los principales paradigmas de programación: imperativo, orientación a objetos, lógico, funcional y concurrente.
- En “Modelos Abstractos de Cálculo” se estudia complejidad algorítmica y computabilidad

Universidad de Murcia
(http://www.um.es/informatica/estudios/Curso0708/estudios/programas0708.php)

Coexisten asignaturas anuales con semestrales. Un total de 60 créditos (unos 48 ECTS) en el área de Programación que se distribuyen de la siguiente forma.

Asignatura	Semestre	Créditos	Unidades
Metodología y Tecnología de la Programación	1, 2	15	PF1, PF2, PF3, PF4, AL1, AL3
Algoritmos y Estructuras de Datos	3, 4	12	PF3, PF4, AL1, AL2, AL3
Computabilidad	3	4.5	AL5, AL6
Programación concurrente	3	6	AL4, OS3
Traductores	4	7.5	PL3, PL8
Programación orientada a objetos	5	6	PL6, PF5, SE2, SE3
Procesadores de Lenguajes	7	9	PL8, PL11

Tabla 14. Asignaturas de Programación en UMU

- Las asignaturas de introducción a la programación y de algoritmos y estructuras de datos son anuales. En el primer año se utiliza el lenguaje Java y en el segundo el lenguaje C. Cabe destacar que “Metodología y Tecnología de Programación” no sigue el mismo enfoque en las tres titulaciones ya que está asignada a departamentos diferentes; en Ingeniería Técnica de Informática de Gestión se sigue el enfoque descrito en el libro de texto [9] y desde el curso 2006/2007 se utiliza Ada como lenguaje de programación.
- Se incluyen asignaturas de programación orientada a objetos, programación concurrente y de informática teórica (computabilidad y complejidad algorítmica).
- Los procesadores de lenguaje se estudian en dos asignaturas: en “Traductores” se estudia el análisis léxico, sintáctico y semántico y en “Procesadores de Lenguaje” se estudia generación de código, optimización y diseño de lenguajes. Es la única universidad en la que se amplía el contenido de procesadores de lenguaje.

5. El área de conocimiento “Programación”

A partir del análisis realizado sobre cómo se organizan los contenidos del área de Programación en varias universidades, vamos a establecer una lista con las unidades temáticas comunes y luego haremos otra lista con aquellas unidades no comunes. Además, identificaremos varias asignaturas prototípicas en las que se suele organizar los contenidos del área. También presentaremos unas conclusiones del estudio comparativo y acabaremos estableciendo una definición sobre el objetivo general de las asignaturas del área de Programación.

Primero vamos a mostrar dos tablas que muestran el número de créditos ECTS que suman las asignaturas de programación que se imparten en cada una de las universidades consideradas.

Universidad	CMU	ETH	UNL	UNIMI	TUM
Créditos	50.5	38	37	36	29

Universidad	UCM	UMA	US	UMU	UPC	UPV	UNIZAR
Créditos	62	53	48	48	43	40	40

Tabla 15. Créditos en asignaturas de programación

Se puede observar como las universidades españolas dedican más créditos que las europeas a las asignaturas de programación, siendo la UCM la que más dedica, el equivalente a un curso completo. No obstante, hay que tener en cuenta que comparamos títulos de cinco años frente a grados de tres años, aunque normalmente los cursos de programación se incluyen en los primeros tres años, y que todos los planes de estudio españoles incluyen como obligatoria una asignatura sobre procesadores de lenguajes de 9 créditos que no es materia obligatoria en la mayoría de universidades europeas, como es el caso de las consultadas.

Si tenemos en cuenta el CC2001 y suponemos que en un grado de cuatro años (240 ECTS), unos 60 créditos se destinan a optativas, proyecto fin de carrera y prácticas en empresa, entonces de los 180 créditos restantes, unos 60 deberían ser de programación. Sólo la CMU (el número de créditos obligatorios son únicamente 60) y la UCM mantienen ese porcentaje.

La siguiente tabla muestra la lista de las unidades temáticas de programación comunes a todas las universidades. Todas ellas son unidades *core* del CC2001.

1. Construcciones básicas en programación (PF1)
2. Algoritmos y resolución de problemas (PF2)
3. Estructuras de datos básicas (PF3)
4. Recursión (PF4)
5. Programación dirigida por eventos (PF5)
6. Eficiencia de algoritmos (AL1)
7. Estrategias algorítmicas (AL2)
8. Algoritmos básicos (AL3)
9. Programación Orientada a Objetos (PL6)
10. Diseño de Software (SE1)
11. Manejo de APIs (SE2)
12. Entornos y herramientas de programación (SE3)

Tabla 16. Unidades del área de Programación comunes

La forma en la que habitualmente se organizan estos contenidos es la siguiente. En el primer año siempre existe un curso de **introducción a la programación** (8-18 ECTS) organizado normalmente en dos asignaturas, una en cada semestre (aunque en la UPV y en la UMU se organiza en una asignatura anual, y en la UCM una anual junto a una semestral). En España existe cierta homogeneidad en la organización de este primer año. Denominaremos IP1 a la asignatura del primer semestre e IP2 a la del segundo, IP cuando hay una única asignatura.

IP1 (6-9 ECTS)
Objetivo general: Que el alumno aprenda a escribir programas para resolver problemas elementales y pequeños, dentro del marco de la programación imperativa (aunque en algunas universidades, como en UNL, UNIMI, UPC se utiliza un lenguaje orientado a objetos como Java o C++ en vez de un imperativo)
Contenidos básicos (PF1, PF2, pequeña parte de PF3 y AL3) Construcciones básicas de los lenguajes de programación (composición secuencial, condicional e iterativa, procedimientos y funciones, tipos de datos básicos, tipos estructurados registro, tabla y string), diseño iterativo, diseño descendente y algunos algoritmos básicos como los de recorrido y búsqueda en tablas.
Variaciones: - Se enmarca dentro del paradigma orientado a objetos. (ETH) - Se incluyen también contenidos propios de IP2 (ETH, UNL)

- Se incluye diseño recursivo y análisis de algoritmos (UPC)

IP2 (6-9 ECTS)
Objetivo general: Que el alumno sea capaz de enfrentarse a problemas más complejos que los de IP1 conociendo el diseño modular, la recursión y las estructuras de datos básicas. El alumno debe desarrollar un pequeño proyecto de programación.
Contenidos básicos (PF3, PF4): Diseño Modular. Recursión. Estructuras de datos estáticas y dinámicas (pilas, colas, listas, árboles binarios)
Variaciones: - Introducir la programación orientada a objetos en esta asignatura (UNIMI, UPC, US). - Estudio del análisis de algoritmos (CMU) o introducir el problema de la eficiencia (ETH, UV, UMU) - Estudio de la verificación formal y complejidad algorítmica (UNIZAR)

En todas las universidades, en el segundo año se profundiza en el conocimiento de los **algoritmos y las estructuras de datos** a través del estudio de análisis de algoritmos, esquemas algorítmicos y estructuras de datos más avanzadas. Al igual que en el primer año, estos contenidos se organizan en dos asignaturas semestrales o en una anual. Denominaremos a estas asignaturas ADA (Análisis y Diseño de Algoritmos) y EDA (Estructuras de Datos y Algoritmos).

ADA (6-9 ECTS)
Objetivo general: Que los alumnos amplíen sus conocimientos en análisis y diseño de algoritmos para ser capaces de crear algoritmos eficientes y correctos para problemas no triviales. Se estudia análisis de algoritmos (eficiencia) y esquemas algorítmicos. Como aplicación de estos esquemas se diseñan algoritmos básicos.
Contenidos básicos (PF4, AL1, AL2, AL3): Análisis de Algoritmos. Esquemas Algorítmicos.
Variaciones: - Incluir transformación de algoritmos recursivos a iterativos y complejidad de problemas (UCM)

EDA (6-9 ECTS)
Objetivo general: Ampliar los conocimientos sobre estructuras de datos. Se estudia representación y algoritmos sobre estructuras de datos.
Contenidos básicos (PF3, PF4, AL3): TAD. Conjuntos y diccionarios. Árboles. Grafos.
Variaciones: - Incluir especificación formal de TAD (UCM, UMA, UNIZAR, UPC)

En cuanto a la enseñanza de la **programación orientada a objetos** encontramos varios enfoques según el momento elegido para introducirla: en el curso de introducción a la programación (ETH, UNIMI), en la asignatura de programación del segundo semestre del

primer año (CMU, TUM, US), en el curso de estructuras de datos (UPC), como una asignatura separada con una carga de 6 créditos (UNL, UMA, UMU, UPC, UPV). Sólo en UNIZAR no existe una asignatura obligatoria. El manejo de un lenguaje de programación orientado a objetos implica el estudio de una librería de clases (**manejo de APIs**) como STL en el caso de C++ o la librería de Java. Una organización típica de esta asignatura, que denominaremos POO, es la siguiente.

POO (4-6 ECTS)
Objetivo general: Enseñar el paradigma orientado a objetos de una forma independiente del lenguaje. El alumno realiza prácticas en un lenguaje orientado a objetos como Java o C++. Un curso de estas características se imparte en UCM, UMA, y UMU.
Contenidos básicos (PF6, SE2, SE3): Clases y objetos. Herencia. Polimorfismo y ligadura dinámica. Lenguajes orientados a objetos. Librería de clases: Colecciones, entrada, salida, componentes visuales.
Variaciones: - El curso de introducción a la programación del primer año está basado en el paradigma OO (ETH, UNIMI) - Curso de programación en un lenguaje orientado a objetos concreto (UNL) - Se enseña en un curso de estructuras de datos (UPC) - Se enseña en IP2 (US, UNIMI, TUM) - En este curso se suele enseñar manejo de excepciones y de eventos (PF5)

Dentro de los tres primeros años, el alumno suele enfrentarse al desarrollo de un proyecto de programación que le permite integrar los conocimientos adquiridos. Este proyecto puede organizarse dentro de los créditos prácticos de una asignatura de programación o en una asignatura independiente (sólo en UPC y UNIZAR). El desarrollo de este proyecto, exige al alumno el dominio de **entornos y herramientas de programación**.

Aunque el **diseño de software** es abordado en asignaturas del área de Ingeniería del Software, en las asignaturas de programación también se enseñan métodos como el diseño descendente o el diseño modular.

A continuación, mostramos una tabla que lista las unidades temáticas que no son comunes a los planes revisados, aunque cabe llamar la atención de que las cinco primeras unidades son *core* según el CC2001. En la primera columna aparece el nombre y entre paréntesis las unidades CC2001 que le corresponden y la etiqueta *opt* si es opcional según CC2001. En la segunda columna de la tabla se indica qué universidades han considerado la unidad como materia obligatoria.

Unidad Temática	Universidades
Computabilidad y Complejidad (AL5 y AL6)	Todas menos UNIMI y UMA
Programación Concurrente y Distribuida (OS3, AL4)	UMA, UMU
Lenguajes de Programación (PL1, PL4 y PL5)	UNL, UPV, UNIZAR
Validación de programas (SE6)	
Evolución de los programas (SE7)	
Programación Funcional (PL7, opt)	CMU, ETH, UMA,UPV,US, UCM
Programación Lógica (opt)	UMA, UPV, US, UCM
Procesadores de lenguaje (PL3, PL8 opt)	Todas en España
Técnicas de Programación Avanzadas(opt)	CMU
Métodos formales (SE10, opt)	UPC, UMA, UNIZAR, UCM

Tabla 17. Unidades del área de Programación no comunes

En todas las universidades consultadas, excepto CMU, UNIMI y UMA, encontramos una asignatura obligatoria de informática teórica en la que se estudia **computabilidad y complejidad algorítmica** y que corresponde a las unidades AL5 y AL6 del CC2001.

La mitad de las universidades consideradas tienen una asignatura de **programación funcional**. En los planes de UPV, UMA y US se trata de un curso de programación declarativa que también incluye programación lógica, y en la UCM hay dos cursos separados.

De acuerdo con las directrices de la titulación de Ingeniería en Informática, en todas las universidades españolas existe una asignatura para el estudio de los **procesadores de lenguajes**, en todos los casos de 9 créditos excepto en la UMU que se dedican 13.5 créditos en dos asignaturas.

La unidad "Lenguajes de Programación" se refiere a los contenidos habituales de un curso que enseña los mecanismos de los **lenguajes de programación** (tipos, abstracción, control, manejo de memoria, etcétera) en varios paradigmas. Una asignatura con estos contenidos sólo la hemos encontrado en UNL, UPV y UNIZAR.

La **validación** y **evolución del software** no tiene una presencia importante en los planes considerados, como asignatura aislada o como parte importante de una asignatura. Las pruebas se estudian en asignaturas de programación e ingeniería de software pero sin mucha profundidad, y parece que el estudio de la evolución del software no se aborda.

En cuanto a los **métodos formales**, hemos encontrado que en el curso de introducción a la programación se sigue un enfoque informal, excepto en la UPC y UMU en las que se sigue un enfoque semiformal que comentaremos en el siguiente apartado. La derivación formal de programas y la verificación formal se enseñan en UCM y UNIZAR. La especificación algebraica de TAD se estudia en UPC, UMA, UCM y UNIZAR.

No hemos encontrado en ninguna otra de las universidades un curso como "*Principios de Programación*" de CMU en el que se enseñan técnicas avanzadas de programación como metaprogramación, que en dicha asignatura se estudian en el contexto de un lenguaje funcional.

A pesar de tener aspectos comunes, los planes analizados han permitido comprobar también un alto grado de variabilidad en muchos aspectos. Las conclusiones más importantes, desde el punto de vista de los contenidos obligatorios, del análisis realizado de doce planes de estudio serían:

1. En todas las universidades, además del curso de **introducción a la programación** (anual o de dos semestres), se imparte un curso de **algoritmia** y **estructuras de datos**, con los contenidos comentados anteriormente y se enseña **programación orientada a objetos** (excepto en UNIZAR).
2. En todas las universidades españolas se imparte una asignatura sobre **procesadores de lenguaje** ya que es una material troncal en las directrices de la titulación superior, pero no se contempla como obligatorio en el resto de Europa.
3. El curso de **computabilidad y complejidad algorítmica** es incorporado en nueve de las doce universidades.
4. La mitad de las universidades incorporan un curso de **programación declarativa**, normalmente funcional.
5. **Programación concurrente y distribuida**, una materia obligatoria según el CC2001, es una materia optativa en todas las universidades consultadas excepto en UMA y UMU.
6. El curso sobre **lenguajes de programación** sólo se incorpora como obligatorio en tres universidades.
7. No hay asignaturas en las que la **validación** y **evolución del software** ocupe un papel destacado.
8. Se estudian **métodos formales** en cuatro de las doce universidades.
9. El número de créditos es inferior al establecido en CC2001 excepto en la UCM y CMU.

Tras este análisis de las asignaturas de programación en varios planes de estudio y antes de reflexionar sobre algunas cuestiones que deben ser consideradas cuando se diseña un plan de estudios, presentamos una definición del objetivo global de las asignaturas del área de Programación.

Objetivo del Área de Programación

Dotar a los alumnos de la formación precisa para desarrollar programas para problemas medianos, en el marco de expresividad de diferentes paradigmas de programación, y ser capaces de llegar de un modo riguroso desde la especificación de un problema a un programa que sea correcto, eficiente y legible, sabiendo valorar si la solución cumple los requisitos exigidos.

Como señala B. Meyer [12], "la formación universitaria de un profesional del software involucra **principios** (conceptos básicos en los que se sustenta la disciplina), **prácticas** (técnicas de resolución de problemas que domina un profesional de un área), **aplicaciones** de esos principios y prácticas en diversas áreas, **herramientas** (que facilitan la aplicación de los principios y las prácticas), y **teorías matemáticas** que constituyen la base formal que permite integrarlo todo".

Como señalamos en el apartado 2.1. las asignaturas de programación y de ingeniería del software deben diseñarse conjuntamente, por lo que es necesaria una coordinación entre los grupos que se encarguen de organizar estas dos áreas. Además, muchos conocimientos y destrezas que el alumno aprende o adquiere en asignaturas de programación, los aplica y mejora su dominio a través del trabajo en asignaturas de otras áreas. Por ello, también es conveniente la coordinación con estas otras asignaturas.

6. Algunas cuestiones en el diseño del plan de estudios

En este apartado vamos a comentar algunas de las cuestiones a abordar en el diseño de la parte de un plan de estudios relacionada con la programación. Para cada una de ellas existen varias alternativas de diseño igualmente válidas. No hay un único camino para conseguir los objetivos. En primer lugar, vamos a discutir cuestiones que afectan al diseño del plan de estudios en su conjunto y luego le dedicaremos especial atención al diseño del primer curso de programación por su importancia y por existir una gran variedad de propuestas.

6.1 Cuestiones generales

¿Qué contenidos de programación deberían enseñarse?

La obligatoriedad de los contenidos adscritos en el apartado anterior a los cursos IP1, IP2, ADA y EDA siempre ha estado clara y los cursos de introducción a la programación y de algoritmia y estructura de datos son incorporados por todas las universidades. Actualmente tampoco hay duda sobre la necesidad de que el alumno conozca bien el paradigma orientado a objetos y adquiera experiencia en programar con un lenguaje orientado a objetos que tenga una rica librería de clases. Como se ha indicado en el apartado anterior existen varias posibilidades para cumplir este objetivo: comenzar la enseñanza de la programación con el paradigma orientado a objetos, enseñarlo en el segundo semestre del primer año o dentro de la asignatura de estructuras de datos del segundo año, disponer de una asignatura específica en el segundo o tercer año para enseñar orientación a objetos desde un punto de vista independiente del lenguaje junto al aprendizaje de un lenguaje concreto. También parece existir un alto grado de consenso en incorporar un curso semestral de informática teórica en el que se estudie computabilidad y complejidad algorítmica.

Para el resto de contenidos de programación no existe un consenso sobre su obligatoriedad, según se desprende del análisis comparativo realizado. En España, la materia "procesadores de lenguajes" es troncal en el segundo ciclo según las directrices propias de 1990, pero es discutible que deba serlo en el futuro grado, sobre todo si se tiene en cuenta que en el CC2001

es considerada una materia optativa y que en la mayoría de universidades del resto de Europa y de EE.UU. no es incluida entre las materias obligatorias.

Según el CC2001 sería obligatorio un curso de programación concurrente y distribuida y no lo sería uno sobre programación declarativa (funcional y/o lógica), aunque son muchas las universidades que incorporan este último y pocas el primero. El estudio de los lenguajes de programación, sus mecanismos y el contraste entre varios paradigmas, es considerado obligatorio en un 25% de las universidades que hemos consultado.

Es necesario un trabajo con los responsables de organizar los contenidos del área de Ingeniería del Software para cubrir dos aspectos importantes en el desarrollo de software como son la validación (pruebas) y evolución. Ambos aspectos no son cubiertos en su justa medida en la mayoría de planes actuales.

La cuestión es decidir cuántos créditos pueden asignarse al área de Programación y entonces separando los contenidos cuya obligatoriedad está clara (IP1, IP2, EDA, ADA, POO e Informática Teórica) decidir cómo proporcionar la mejor formación posible a los alumnos, considerando la enseñanza de programación concurrente y distribuida, programación funcional, lenguajes de programación, procesadores de lenguaje, pruebas del software y técnicas avanzadas de programación (al estilo del curso "Principios de Programación" de CMU). Entre todas las asignaturas de programación deberían enseñarse todos los principios, prácticas (método y técnicas) y teorías fundamentales del área, así como que los alumnos manejen herramientas para desarrollar programas en varios dominios de aplicación.

En el diseño de los cursos habría que tener en cuenta la posibilidad mencionada en CC2001 de diseñar cursos híbridos que abarquen más de un área.

Cuando se observa la secuencia de cursos recomendada en CMU y se revisan sus cursos obligatorios, se tiene la sensación de que se le dedica un buen número de créditos a la programación (50.5 y no incluye compiladores) y que el alumno recibe una enseñanza que cubre aspectos teóricos y avanzados de programación, lo que le permite adquirir una buena formación. Se trata de una buena referencia a tener en cuenta.

¿Qué técnicas formales se deben enseñar?

Desde el principio hay que transmitir que la programación es una labor de ingeniería que tiene una base matemática y que es necesario un trabajo riguroso para construir buenos programas que sean correctos y eficientes. Hay que inculcar rigor a los ingenieros en informática.

Para ello hay que enseñar las bases teóricas de la programación y debería huirse de presentar la disciplina del software como el arte de construir programas con los lenguajes disponibles. También hay que enseñar técnicas y métodos para abordar la programación de un modo sistemático.

Además de las teorías asociadas a los contenidos obligatorios de programación como las del análisis de algoritmos y computabilidad, es necesario identificar qué métodos formales se deberían enseñar a lo largo de la carrera. Entre los métodos formales candidatos a ser contenidos obligatorios estarían la derivación formal de programas, la especificación formal de tipos abstractos de datos y la derivación de algoritmos a partir de ellas, y el diseño por contrato.

Más adelante comentaremos cómo abordar el primer curso de programación a partir de un enfoque semiformal con la utilización de técnicas como el diseño iterativo y recursivo.

¿Cuál debe ser el número de cursos de matemáticas?

Existe un reconocimiento de la importancia de los cursos de matemáticas en la formación de un ingeniero en informática, al igual que sucede con el resto de ingenierías, como se señala en el CC2001. En esta recomendación, "Matemática discreta" es el área de conocimiento a la que se asigna un mayor peso (incluye lógica), un 15% de contenido obligatorio.

Los cursos de matemáticas contribuyen a que el alumno adquiera conocimientos que son necesarios para comprender principios y técnicas fundamentales de la programación y a potenciar su capacidad de razonamiento matemático formal de tanta utilidad en programación.

Parece existir consenso en la conveniencia de que los planes incorporen un curso de *cálculo* (dos semestres), *álgebra*, *matemática discreta*, *estadística* y *probabilidad*, y *lógica* (éste a veces se incluye en matemática discreta), lo que debería suponer un mínimo de unos cuarenta créditos. En la mayoría de las universidades españolas se imparten estos cursos.

El curso de matemática discreta y el de lógica se debería impartir en el primer semestre del primer año, junto al primer semestre de cálculo. Los cursos de estadística y probabilidad y el de álgebra lo antes posible.

La combinación de la programación con las teorías matemáticas, como la que se ofrece en el curso "*Grandes ideas teóricas en Informática*" de CMU, es muy interesante y se puede aprovechar en las asignaturas de programación, aunque no exista una asignatura concreta como en el caso de CMU, mediante la realización de trabajo práctico que exija aplicar cierta teoría matemática para resolver problemas atractivos para los alumnos.

¿Qué lenguajes se deben enseñar en la carrera?

En general, el lenguaje ideal para una asignatura de programación sería aquel que posea buenas propiedades, para el que existan entornos apropiados, que sea muy utilizado en las empresas y en la industria, y que se disponga de buena documentación. En cuanto a las buenas propiedades del lenguaje nos referimos a que:

- Se ajuste a los objetivos de la asignatura. Por ejemplo, si se quiere enseñar diseño modular el lenguaje deberá incorporar el concepto de módulo o si se deben representar estructuras de datos sería apropiado que soportase genericidad.
- Refleje claramente los conceptos del paradigma que se trate. Por ejemplo, lenguajes orientados a objetos puros son más apropiados que los lenguajes híbridos como C++ para enseñar orientación a objetos.
- Que sea tipado estáticamente para favorecer la fiabilidad y legibilidad.
- Sea pequeño pero que tenga gran potencia expresiva, como sucede por ejemplo con los lenguajes funcionales y lenguajes dinámicos orientados a objetos como Ruby o Smalltalk.

En cuanto a los entornos deberían tener todas las facilidades necesarias, ser sencillos de usar y robustos, bien documentados y gratuitos.

Es evidente, que a la hora de elección del lenguaje es muy difícil encontrar un lenguaje que satisfaga todos los requisitos deseables, y a menudo hay que optar entre las diversas características. Por ejemplo, es habitual disponer de un lenguaje apropiado desde un punto de vista académico pero que su uso está poco extendido en la industria, como sucede con Pascal o Modula en el caso de la programación imperativa o con Eiffel en el caso del paradigma orientado a objetos, o bien la situación contraria que se da con un lenguaje como VisualBasic, o tener que elegir entre un lenguaje pequeño y potente pero que no es tipado y otro más complejo pero que sí es tipado.

A lo largo de la carrera el alumno debería conocer, al menos, los lenguajes C y Java. El primero por ser el lenguaje más utilizado en el contexto de programación de sistemas y Java por ser un lenguaje que refleja bien los conceptos de la orientación a objetos, y que está muy extendido ya que es la base de una de las principales plataformas de desarrollo software. A partir de Java es fácil aprender otros lenguajes orientados a objetos como C#, lenguaje de Microsoft para su plataforma .Net.

También habría que valorarse la conveniencia de enseñar un lenguaje funcional (la cuestión sería si se debe incluir un curso de programación funcional) y un lenguaje dinámico orientado a objetos como Ruby, ya que permiten enseñar técnicas avanzadas de programación muy útiles como las que se enseñan en el curso "Principios de Programación" de CMU comentado en el apartado 3 de este documento: introspección, intercesión, iteradores, bloques de código, etc.

No cabe duda de que en el diseño de un plan de estudios es importante determinar en qué lenguajes debe tener experiencia el alumno al acabar sus estudios y en qué secuencia se enseñarán, lo que exige una coordinación entre diferentes asignaturas.

¿Qué herramientas de programación deberían utilizarse?

A lo largo de la carrera un alumno debe adquirir experiencia en herramientas de programación básicas y algún entorno de programación. Las herramientas que debería dominar son: utilidades del sistema operativo que se ejecutan como órdenes, compilador y sus opciones, depuradores, analizadores de memoria dinámica, generadores de documentación (como doxygen), gestores de versiones (como CVS o Subversion), constructores de un sistema (como make o ant). Probablemente, también debería tener conocimientos en algún lenguaje de script. En cuanto a entornos de programación, debería conocer bien los entornos Unix y Eclipse.

¿Cuándo se enseñan la validación y evolución de software?

Una crítica común a los planes de estudio es que no se presta atención a dos aspectos clave en el desarrollo del software como son las pruebas y la evolución, que son considerados unidades *core* en el CC2001.

Desde el primer curso de programación, se debería concienciar al alumno de la importancia de las pruebas y habría que ver cómo se consigue que el alumno sea capaz de realizar pruebas de forma sistemática. Debería ser obligatorio el dominio de herramientas de prueba como JUnit.

En cuanto a la evolución, los alumnos deberían enfrentarse a algún proyecto en el que tuviesen que realizar cambios a código existente y reutilizar software para satisfacer alguno de los requisitos. Esto último implicaría elegir el componente adecuado, adaptarlo e integrarlo.

¿Es conveniente una asignatura específica de "Proyecto de Programación"?

Algunas universidades, como UPC y UNIZAR han incorporado en sus planes de estudio una asignatura cuyo objetivo es que el alumno desarrolle un proyecto de programación en grupos de dos o tres alumnos. Estos proyectos sirven para aplicar e integrar los conocimientos adquiridos en varias asignaturas de programación en los dos primeros años y desarrollar una aplicación de tamaño mediano. Además, estos proyectos también sirven para llevar a la práctica aspectos como las pruebas y la evolución, comentados anteriormente.

El trabajo de los alumnos es tutelado por un profesor que guía a los alumnos y con el que mantienen reuniones semanales de seguimiento.

También podría tener sentido que se realizase otro proyecto de esta naturaleza en el último o penúltimo semestre, una vez ha estudiado todas las asignaturas de programación e ingeniería del software, aunque se puede considerar que el proyecto fin de carrera sirve para ese propósito.

Estos proyectos también pueden organizarse dentro de una asignatura que tenga créditos prácticos suficientes, aunque es más difícil a no ser que sea anual. Reconocida la conveniencia de estos proyectos, las universidades deben analizar cuidadosamente cómo integrarlos en el plan de estudios.

¿Conviene que algunas asignaturas tengan duración anual?

En algunas universidades españolas, como UPV, UCM y UMU, coexisten asignaturas anuales con semestrales. Cuando los centros prepararon sus planes de estudio de acuerdo a las directrices de la LRU aprovecharon para organizar todas las asignaturas como semestrales. Sin embargo, cuando estos planes fueron reformados posteriormente, algunos centros consideraron oportuno organizar algunas asignaturas como anuales. Estas asignaturas suponían juntar en una sola lo que antes eran dos asignaturas semestrales íntimamente ligadas, como sucedía en el caso de las dos asignaturas semestrales que conformaban el primer año de programación, o las dos en las que se estudiaban algoritmos y estructuras de datos o las dos de sistemas operativos, entre otras. En el caso de la UCM parece que se ha querido asignar una carga lectiva considerable a varias materias, mediante asignaturas anuales en las que sólo se imparten clase teóricas y de resolución de ejercicios.

Las principales razones que motivaron la vuelta a asignaturas anuales fueron, entre otras, que los alumnos tuviesen todo un curso para madurar una materia que no tenía sentido separar, con

lo que se evitaba el examen final del primer cuatrimestre cuando apenas habían tenido tiempo de asimilar y comprender los primeros conceptos, y también se permitía al profesor organizar mejor las prácticas, ya que se elimina la interrupción entre el fin de la asignatura del primer semestre y el comienzo de la del segundo.

Por mi experiencia en la enseñanza del primer curso de programación defendí la existencia de asignaturas anuales, aunque ahora no lo tengo claro, ya que el cambio no ha mejorado los resultados académicos y cuando un alumno suspende alguna parte de la asignatura debe matricularse de todos los créditos.

6.2 Cuestiones específicas del primer año de programación

¿Qué asignaturas de informática debería haber en el primer año?

En CC2001 se discute con detalle cuáles deben ser las asignaturas de informática que deben enseñarse en el primer año de la carrera. Allí se señala la conveniencia de que el alumno adquiera una visión global de la disciplina, con una introducción a todas las áreas, además de recibir una formación inicial en programación.

En el informe “Computing as a Discipline” [3], que sentó las bases del CC2001, se proponía un curso introductorio en el que además de los temas propios de programación se incluía “Organización de Computadores”, “Límites de la computación”, “Sistemas Operativos y Seguridad”, “Redes y Sistemas Distribuidos”, “Modelos de Inteligencia Artificial, “Ficheros y Bases de Datos”, “Paralelismo” e “Interfaces Persona-Ordenador” para impartir en tres cursos semestrales.

Hasta que se elaboraron los planes de estudio de acuerdo a las directrices de las ingenierías en informática (BOE, 20 de noviembre de 1990), en el primer año de la carrera de muchas universidades españolas, además de un curso de programación se impartía una asignatura de introducción a la informática. En la Universidad de Murcia, como en otras, esta asignatura proporcionaba al alumno una visión global de la informática a través del enfoque de máquina multinivel. En ella se enseñaba la evolución de la informática, representación de la información, sistemas digitales, la estructura y funcionamiento del ordenador, periféricos, sistemas operativos y redes, estructura de un compilador, ficheros y bases datos, y una visión del ciclo de vida del desarrollo de una aplicación software.

Sin embargo, los planes de estudio españoles cambiaron y la mayoría optó por incluir en el primer año un curso de programación junto a otro sobre la estructura y funcionamiento de los ordenadores, y en muchas universidades esta asignatura se acompaña de otra sobre sistemas digitales. Si observamos los planes de estudio de las universidades del resto de Europa y de EE.UU, notamos que esta organización es poco frecuente. De hecho, en los planes revisados sólo en el ETH se imparte en el segundo semestre un curso de sistemas digitales.

La elaboración de los planes para el grado en informática supone una ocasión para reflexionar sobre si en los planes actuales hay demasiada “estructura y arquitectura de computadores” y analizar si es conveniente que regrese la **asignatura de introducción a la informática**” y cuáles debían ser sus contenidos. Quizá sea importante que los alumnos adquieran al principio esa visión global que aporta el beneficio, entre otros, de que el alumno no crea que “todo se reduce a programar, esto es, informática = programación”, como se expone en [1,3].

¿Cómo se debe organizar el primer curso de programación?

Si aceptamos como objetivo de este primer curso que “debe proporcionar a alumnos sin ninguna experiencia en programación los mecanismos necesarios para enfrentarse a la creación de programas que resuelvan problemas pequeños, en el marco de expresividad de un paradigma de programación, enseñándoles un lenguaje de programación y los conceptos, métodos y técnicas que les permitan abordar los problemas y llegar de un modo riguroso desde la especificación a un programa correcto, eficiente y legible”, entonces nos surgen una serie de cuestiones como:

- ¿Qué paradigma de programación elegimos?
- ¿Qué lenguaje utilizamos?
- ¿Cómo inculcamos el rigor a los alumnos?
- ¿Qué contenidos enseñamos?

Por supuesto, hay que elegir la duración del curso y en cuántas asignaturas se organiza. Según hemos visto, en las universidades españolas se ha optado por dos asignaturas semestrales según el esquema de IP1 e IP2 que hemos comentado en el apartado anterior, y en menor medida por reunir las dos asignaturas en una anual. En Europa hemos encontrado universidades con un único curso semestral de duración variable (ETH, UNIMI, UNL).

En cuanto a la duración, en España se asignan de 9.6 a 18 ECTS, mientras que en Europa hemos encontrado universidades con una asignación de 8 ECTS (UNL, ETH) y otras con 18 (TUM, UNIMI).

A continuación, vamos a analizar la lista de cuestiones anteriores.

¿Qué paradigma debe utilizarse?

Cualesquiera que sea el paradigma elegido se puede conseguir que el alumno adquiera la formación adecuada en programación para proseguir sus estudios. En la mayoría de universidades se sigue eligiendo el paradigma imperativo para comenzar, son muy pocas en las que se utiliza el paradigma orientado a objetos y no conozco ninguna universidad española en la que se comience con el paradigma funcional o lógico.

Personalmente, defiendo el enfoque imperativo por los argumentos que se exponen en [8], entre los que destacan los siguientes:

- En el primer año, el alumno debe preocuparse principalmente de diseñar y escribir algoritmos iterativos y recursivos que manejan secuencias, strings y arrays (posiblemente multidimensionales) de tipos básicos. La introducción del concepto de objeto complica innecesariamente estos algoritmos, al ser necesario incluir los conceptos de clase y mensaje, en vez de escribir una simple secuencia de instrucciones imperativas.
- Un alumno debe dominar un lenguaje imperativo (especialmente C) al acabar la carrera, y es más fácil enseñar primero el paradigma imperativo y luego el orientado a objetos, que hacerlo al revés. Es más fácil pasar de invocación de rutinas a mensajes, de monomorfismo a polimorfismo, de la ligadura estática a la ligadura dinámica que al contrario.
- Cuando el alumno tiene experiencia en diseño descendente y modular valora mejor las ventajas de la orientación a objetos.

Cada vez gana terreno la idea de introducir la programación orientada a objetos lo antes posible, en el segundo semestre del primer año o en el primero del segundo año, pero en la mayoría de universidades sigue utilizándose el paradigma imperativo en el curso de introducción del primer semestre, aunque en muchas ocasiones se usa un lenguaje orientado a objetos, normalmente Java. Sin duda la propuesta más seria sobre cómo empezar con el paradigma orientado a objetos es la que sigue Bertrand Meyer en su curso del ETH, que más adelante comentaremos, y que está reflejando en un excelente libro de introducción a la programación todavía inacabado [13]. En UNIMI se imparte un curso en el que se enseña orientación a objetos en Java en el primer semestre del primer año, después de haber construido los primeros algoritmos mediante un diseño procedural [16].

En la actualidad los tres enfoques utilizados en las universidades españolas en el primer año de programación son:

- Paradigma imperativo y uso de un lenguaje imperativo (C, Pascal, Ada, Modula) en los dos semestres.
- Paradigma imperativo y uso de un lenguaje orientado a objetos (Java o C++) en los dos semestres.

- Paradigma imperativo en el primer semestre y paradigma orientado a objetos en el segundo semestre (Java o C++).

No tengo constancia de universidades españolas en las que la introducción a la programación se desarrolle en el marco del paradigma orientado a objetos o del paradigma declarativo.

¿Cómo inculcamos el rigor a los alumnos desde el principio?

Como señalamos anteriormente, desde el principio hay que transmitir que la programación no es un arte sino que como toda labor de ingeniería se sustenta en unas teorías, y que sólo el trabajo riguroso puede conducir a la creación de buenos programas.

El primer curso no debe limitarse a la enseñanza de un lenguaje de programación y a mostrar ejemplos de algoritmos simples y otros más avanzados que involucran recursión y/o estructuras de datos lineales o arborescentes. En cambio, la preocupación principal debe ser enseñar de forma rigurosa, a una persona que no tiene conocimientos previos de programación, los conceptos básicos de programación y algunas técnicas y métodos que ayuden a encontrar soluciones a problemas algorítmicos. Es importante incorporar alguna técnica formal para que el alumno sea consciente de que la programación debe abordarse desde una perspectiva ingenieril.

J. Van Amstel describió muy bien los tres enfoques que es posible aplicar en un curso de introducción a la programación [18]:

- *No formal*: Basado en mostrar ejemplos escritos en un lenguaje.
- *Semiformal*: Se utiliza el concepto de invariante para deducir el algoritmo, pero no se aplica una derivación formal.
- *Formal*: Se aplica el método de derivación formal de programas a partir de la especificación propuesto por E. W. Dijkstra [5].

El enfoque no formal se basa en describir un lenguaje de programación con ejemplos de programas que ilustran la sintaxis y semántica de sus instrucciones y una presentación de los algoritmos clásicos de un primer curso expresados en el lenguaje elegido, pero sin prestar atención especial al proceso creativo que lleva a idear un algoritmo ni a inculcar el rigor en la tarea de programar. Los libros del estilo “Introducción a la Programación en el Lenguaje X” suelen reflejar este estilo de enseñanza.

El enfoque formal, que puede ser ilustrado por el texto “*A Method of Programming*” de Dijkstra y Feijen, plantea la derivación formal de los programas a partir de la especificación. En el debate surgido en 1989 tras la publicación por E. W. Dijkstra de su famoso artículo “*On the cruelty of really teaching computing science*” [4], este enfoque recibió duras críticas ya que, entre otros inconvenientes, suponía: un alejamiento de la práctica real de la programación; la obtención, escritura y manipulación de las especificaciones formales resulta muy complicada y no se tienen en cuenta importantes habilidades y conocimientos que deben enseñarse. T. Winograd señaló que “no hay duda de que hay que introducir a los estudiantes en el pensamiento riguroso, pero de ahí a encumbrar a la manipulación de abstracciones formales como objetivo primordial media un abismo”.

El enfoque semiformal tiene en cuenta algunos de los conceptos básicos del enfoque formal, como son la deducción de los algoritmos iterativos a partir del invariante de bucle, aunque sin desarrollar una derivación formal de los programas, y prestar atención a la especificación con precondiciones y postcondiciones. Se pretende introducir desde el comienzo un razonamiento riguroso en el diseño de algoritmos pero sin alejarse de la práctica habitual con la que se escribe código.

No conozco ninguna universidad española en la que los alumnos aprendan a programar siguiendo los postulados planteados por Dijkstra y sí demasiadas en las que se sigue el enfoque no formal. En [7] aparece un análisis sobre los contenidos del curso de introducción a la programación en 33 titulaciones de 25 universidades españolas, referidos al curso 2001/02, y en la mayoría de universidades se seguía un enfoque no formal. Sólo en la UPC y en la UMU (asignatura adscrita al Departamento de Informática y Sistemas en Ingeniería Técnica de

Informática de Gestión) se sigue un enfoque semiformal desde principios de los noventa. En ambas universidades se aborda el diseño iterativo y transformaciones recursivo-iterativo. En la actualidad, la situación española es parecida a la encontrada en [7].

El enfoque semiformal seguido en la UMU desde 1991 se plasma en el texto [9]. Está inspirado en el enfoque de diseño iterativo plasmado en el libro "Esquemas Algorítmicos Fundamentales" de Scholl y Peyrin [17] y en el enfoque de la resolución de problemas expuesto en el magnífico libro "*How solve it by computer?*" de Richard Dromey [6]. Los algoritmos iterativos se obtienen a partir del invariante aplicando un razonamiento inductivo.

Recientemente, Bertrand Meyer en su curso del ETH también ha apostado por seguir un enfoque semiformal basado en el manejo de los invariantes y variantes de bucles, y en la técnica del *Diseño por Contrato* ideada por él mismo y soportada por el lenguaje Eiffel.

¿Cuáles deben ser los contenidos en un primer año de programación?

En la mayoría de universidades de todo el mundo, en el primer año se enseñan los contenidos que en el apartado 4 hemos adscrito a los cursos IP1 e IP2. A continuación, exponemos los principales conceptos que debería enseñarse en un primer curso de programación, de forma independiente a que el paradigma elegido sea el imperativo o el orientado a objetos:

- Paradigma de programación elegido
- Lenguaje de Programación
- Tipo de dato. Tipos simples.
- Tipos estructurados básicos: tabla, registro y secuencia
- Abstracción: Operacional y de datos
- Separación especificación-implementación.
- Ocultación de información
- Especificación de operaciones: precondiciones y postcondiciones
- Diseño iterativo. Inducción e Invariante.
- Esquemas algorítmicos de recorrido y búsqueda en tablas y secuencias.
- Diseño recursivo.
- Eficiencia de los algoritmos.
- Representación estática y dinámica de estructuras de datos.
- Estructuras de datos lineales (pilas, colas, listas)
- Estructuras de datos arborescentes (árbol binario)

Al principio el alumno cree que toda la dificultad del proceso de programación es la originada por el dominio del lenguaje de programación, ya que supone un medio nuevo para expresar sus ideas. Esto se acentúa por la naturaleza simple de los primeros problemas que resuelve. Sin embargo, una vez se siente cómodo con la utilización del lenguaje, descubre que la programación es una actividad muy difícil, incluso para problemas elementales o pequeños debido a la dificultad de descubrir un algoritmo o de encontrar uno que satisfaga ciertas restricciones relacionadas con algún recurso como tiempo de ejecución o uso de memoria, o debido al excesivo nivel de detalle de la especificación, o a la dificultad de encontrar una representación adecuada de los datos.

Los métodos y técnicas de la ingeniería del software son muy útiles para abordar el desarrollo industrial de aplicaciones software, pero ya en el primer curso hay que dotarles de algunas herramientas conceptuales que le ayuden a abordar la construcción de sus primeros programas y que constituyan una sólida base para futuros métodos y técnicas más avanzados, como son:

- El razonamiento inductivo basado en el concepto de invariante como técnica para el diseño de iteraciones.
- El refinamiento por pasos sucesivos
- El diseño recursivo y las transformaciones recursivo-iterativas.

- Esquemas algorítmicos de recorrido y búsqueda en tablas y secuencia.
- La descripción minuciosa del razonamiento que lleva a encontrar la estrategia de solución de cada algoritmo.
- Comparar algoritmos y determinar cuál es mejor según cierto criterio.

¿Qué lenguaje se debe utilizar?

En cuanto a la elección del lenguaje tenemos varias posibilidades, según el enfoque previsto, que podemos resumir en:

- Un lenguaje imperativo.
- Un lenguaje orientado a objetos si se enseña el paradigma orientado a objetos como sucede en los cursos del ETH y UNIMI (quizá Java y Eiffel son las mejores opciones). No parece muy apropiado utilizar un lenguaje orientado a objetos si se sigue el paradigma imperativo.
- Un lenguaje imperativo en el primer semestre y uno orientado a objetos en el segundo si se introduce el paradigma orientado a objetos. Se puede pensar que para no introducir dos lenguajes, se puede enseñar el mismo lenguaje orientado a objetos en los dos semestres (un lenguaje híbrido, en concreto C++).

En el caso de un lenguaje imperativo para todo el año, ADA podría ser el lenguaje más apropiado por sus buenas propiedades: es un lenguaje que facilita la programación estructurada, favorece la legibilidad y soporta tipos genéricos. La elección de C es discutible ya que no es un lenguaje claro y sencillo que favorezca la legibilidad y la aplicación de los principios de la programación estructurada. No obstante, la combinación C/C++ puede ser útil si se sigue el tercer enfoque.

Durante los primeros meses es conveniente utilizar sólo una notación algorítmica para liberar al alumno de prestar excesiva atención en detalles particulares de un lenguaje de programación y centrarse en el diseño de los algoritmos. De este modo, el alumno escribe y prueba sus primeros algoritmos sobre papel lo cual es una forma de educarlo en el "pensar antes de codificar".

¿Cómo motivamos a los alumnos del primer año?

El nivel de fracaso académico en las asignaturas programación es alto, especialmente en las asignaturas de introducción a la programación y de algoritmos y estructuras de datos, con tasas de alumnos que superan la asignatura entre un 10% y un 20%. Los profesores de esos cursos critican la falta de motivación de los alumnos. Como se señalaba en [10], en los últimos años los profesores de introducción a la programación han tenido que abordar el reto de enseñar a programar a alumnos que ya habían tenido un contacto con los ordenadores y con los juegos de ordenador (*la generación del Nintendo*) y a los que no les llamaba la atención los problemas habitualmente utilizados para enseñar a programar. En el año 1994 unos alumnos de la asignatura de introducción a la programación del primer semestre, que habían aprobado con buena nota, me comentaban como profesor de la asignatura que les había gustado la asignatura aunque les había frustrado no resolver problemas en los que la parte de la interfaz de usuario hubiese sido importante, ellos pensaban en algún tipo sencillo de juego. Me criticaban que todos los problemas requerían una salida por pantalla muy pobre. Me imagino que la frustración de los alumnos en años sucesivos fue creciendo.

Problemas como la búsqueda binaria, la justificación de un texto o la generación de las permutaciones de un orden *n* no fascinan a unos jóvenes que vienen de manejar juegos llenos de gráficos y animación en la videoconsola, aunque sean planteados por un buen profesor y se trate de problemas que requieren de creatividad e ingenio para su resolución. Y la motivación de los alumnos también se ve influenciada negativamente si programan con un lenguaje que "sólo se usa en la universidad".

Para motivar a los alumnos, en [10] se propone utilizar un lenguaje como Squeak, una moderna versión de Smalltalk, en la que es posible que los primeros algoritmos que crean los alumnos manejen datos multimedia gracias a una potente librería de clases y al estilo de programación orientada a objetos (mensajes). Por otra parte, B. Meyer está aplicando lo que denomina el *curriculum invertido* en su asignatura del ETH [14] en la que los alumnos empiezan utilizando (reutilizando) clases de una potente librería denominada *Traffic* que incluye clases para manejar gráficos avanzados y datos multimedia. *Traffic* es, en realidad, un pequeño framework en el dominio de la gestión de tráfico en una gran ciudad.

Con *Traffic*, B. Meyer pretende motivar a los alumnos planteándoles problemas en un dominio que les resulta cercano y conocen bien, y que exigen crear interfaces de usuario con gráficos y animación. Los principios en los que se basa el curso del ETH son los siguientes:

- Se utiliza el paradigma orientado a objetos
- Los alumnos empiezan escribiendo código muy sencillo (secuencia de pocos mensajes) con la librería *Traffic*.
- Los contenidos típicos de un primer curso de programación (variable, asignación, tipo, estructuras de control, etcétera) se introducen después de los conceptos orientados a objetos de clase, objeto, interfaz, atributo, método y mensaje.
- Los alumnos construyen código aplicando el diseño por contrato como formalismo que los educa en el rigor: precondiciones y postcondiciones de métodos, invariante de clase, invariante y variante de bucles.
- A lo largo del curso los alumnos deben construir una pequeña aplicación que consideren útil y de un dominio que les resulta familiar. Esta aplicación involucra una interfaz de usuario con gráficos y animación y el diseño de algoritmos no triviales.

¿Buenas prácticas para enseñar a programar?

Como es lógico, cuando se desea aprender a programar no es suficiente con los conocimientos teóricos, sino que el aprendizaje, como sucede con cualquier actividad constructiva, requiere de un trabajo práctico en el que alumno se enfrente a la creación de programas.

Además de las clases teóricas, deben impartirse clases de ejercicios en las que el profesor resuelve problemas para un grupo reducido de alumnos (no debería superar los veinte alumnos) y clases de laboratorio tuteladas por el profesor en las que el alumno escribe y prueba sus programas. En las clases de ejercicios, los alumnos podrán participar activamente al ser grupos reducidos.

Cuando un alumno escribe sus primeros programas le resulta muy difícil el paso del enunciado del problema a organizar el programa. La programación les supone la entrada en un universo de ideas y pensamiento totalmente nuevo, y es necesario proporcionarles de forma cuidadosa los elementos que le permitirán afianzar esas ideas y adquirir destreza en esa nueva forma de razonar. Por ello, es necesario que en clases de ejercicios se resuelvan muchos problemas para transmitirles el tipo de razonamiento que lleva del problema al esbozo de la solución, analizando cada paso dado. Luego, serán ellos los que deban enfrentarse de forma individual a ejercicios propuestos en las clases de laboratorio.

Una buena fuente de conocimiento para el alumno es la lectura y comprensión de código escrito por buenos programadores, por ejemplo el que se encuentra en muchos libros de texto o código preparado por el departamento que imparta la asignatura, como en el caso de la librería *Traffic* del ETH. Un ejercicio interesante es que el alumno modifique código escrito por otros.

Hay que evitar crear programadores compulsivos, ansiosos por sentarse delante de la máquina para escribir directamente el código y establecer con el ordenador una batalla estúpida hasta obtener el programa que se supone obtiene el resultado deseado, a través de la técnica “prueba-error”. Por ello, como señalamos antes, sería conveniente que durante un tiempo se les exigiese que expresasen sus soluciones en una notación algorítmica no ejecutable, y analizasen la corrección, como paso previo a escribir el programa. Esto también supone una forma de educar en el rigor.

Dos posibles estrategias a la hora de enseñar el lenguaje de programación son: que se enseñe al tiempo que se avanza con la notación o comenzar a introducirlo cuando han transcurrido varios meses, por ejemplo en el tercer mes. La primera estrategia es obligada en el caso de una asignatura semestral, y en ella habría que exigir a los alumnos que escribiesen primero la solución con la notación antes de codificarla en el lenguaje elegido. Con la segunda estrategia está claro que hasta que conozcan el lenguaje, los alumnos sólo podrán utilizar la notación. Durante ese intervalo de tiempo los algoritmos se ejecutan utilizando una tabla que muestra la evolución de los valores de las variables a lo largo de la ejecución. Una vez que conocen el lenguaje pueden traducir los algoritmos y ejecutarlos sobre un ordenador, utilizando un depurador como herramienta para mostrar la evolución de los valores de las variables y analizar el comportamiento del programa.

Sería muy útil reunir un conjunto de buenas prácticas docentes que estuviesen disponibles para los departamentos que enseñan programación. Como se puede comprobar cada año en las Jornadas de Enseñanza Universitaria de la Informática (JENUI), muchos profesores o departamentos han probado diferentes métodos para mejorar la enseñanza de la programación en el primer año, por ejemplo propuestas de aprendizaje basado en problemas o técnicas para motivar a los alumnos.

7. Estructura del documento final

Tras el debate en la sesión del SINDI y cuyas aportaciones se han recogido en un documento que acompaña a esta ponencia, se podría crear un grupo de trabajo formado por expertos en la enseñanza de la programación para que elaborasen unas recomendaciones relacionadas con el diseño del área de Programación en los próximos planes de estudio. Este grupo debería trabajar coordinado con el grupo de Ingeniería del Software. La recomendación curricular final sería encargada a un grupo de trabajo que se encargaría de integrar las recomendaciones de cada área.

Para elaborar la propuesta del área de Programación, habría que tener en cuenta los documentos de análisis que algunas universidades han preparado recientemente durante el proceso de diseño de sus planes de estudio o de preparación de la adaptación al EEES, y también los trabajos que muchos profesores han presentado en las JENUI relacionados con la enseñanza de la programación.

El documento que preparase el grupo de Programación, podría tener la siguiente estructura. Primero se analizarían recomendaciones curriculares existentes. Luego se presentaría un análisis sobre los planes de estudio españoles y de otros países. A continuación se describiría en detalle el área de Programación: contenidos y objetivos formativos y se expondrían un conjunto de recomendaciones que darían respuesta a las cuestiones que surgen durante el diseño del plan; también se explicaría como se consiguen los objetivos básicos. Se definirían varios ejemplos de cursos y se mostrarían varios ejemplos de posibles organizaciones de los cursos en asignaturas. Finalmente, se listarían y comentarían libros para cada una de las unidades temáticas (sería interesante tener referencias a los numerosos textos que se han escrito en los últimos años en las universidades españolas).

8. Referencias

1. *Computing Curricula 2001*, Computer Science, IEEE y ACM, diciembre, 2001, http://www.acm.org/education/curricula.html
2. *Computing Curricula 2005: The Overview Report*, IEEE y ACM, septiembre 2005, http://www.acm.org/education/curricula.html
3. Peter J. Denning et al., *Computing as a discipline*, Comm. of the ACM, 32 (1): 9-23, 1989.
4. Peter J. Denning, *A debate on teaching computing science*, Comm. of ACM, 32 (12): 1397-1414, 1989.

5. Edsger W. Dijkstra y W. H. J. Feijen, “A Method of Programming”, Addison Wesley, 1988.
6. Richard Dromey, “How to solve it by computer”, Prentice-Hall, 1982.
7. J.L. Fernández Alemán, Proyecto Docente TEU, Universidad de Murcia, 2002.
8. Jesús J. García Molina, *¿Es conveniente la orientación a objetos en un primer curso de programación?*, VII Jornadas sobre Enseñanza Universitaria en Informática (JENUI), pags. 293-299, ed. UIB, Mallorca, 2001, disponible en http://dis.um.es/~jmolina/publi.html.
9. J. García Molina, F.J. Montoya, J.L Fernández y M.J. Majado, “Una Introducción a la Programación. Un enfoque algorítmico”, Thomson, 2005.
10. Mark Guzdial y Elliot Soloway, *Teaching the Nintendo Generation to Program*, Comm. Of the ACM, 45(4): 17-21, 2002.
11. *Libro Blanco del Título de Grado en Ingeniería en Informática*, Proyecto EICE, ANECA, mayo, 2004, http://www.aneca.es/activin/docs/ libroblanco_jun05_ informatica.pdf
12. Bertrand Meyer, *Software Engineering in the Academy*, IEEE Computer, 34(5): 28-35 (2001).
13. Bertrand Meyer, “Touch of class. Learning to program with Object Technology and Design by Contract”, borrador disponible en www.inf.ethz.ch/personal/meyer
14. Michela Pedroni y Bertrand Meyer, *The Inverted Curriculum in Practice*, SIGSE’06, March 2006.
15. Ricardo Peña Marí, “Diseño de Programas. Formalismo y Abstracción”, tercera edición, Pearson, 2005.
16. G. Pighizzini e M. Ferrari. “Dai fondamenti agli oggetti. Corso di programmazione Java (seconda edizione)”. Pearson Education, 2005.
17. Pierre C, Scholl y Jean P. Peyrin, *Schémas algorithmiques fondamentaux: séquences et itérations*, Masson, 1988 (Edición española: *Esquemas Algorítmicos Fundamentales: Secuencias e Iteración*, Masson, 1991).
18. Jan J. van Amstel, *Teaching programming at various levels of formality*, IFIP, 1985.