¿Podemos darle la vuelta a la enseñanza del desarrollo del software?

Josep M. Marco-Simó, Isabel Guitart Hormigo, M. Jesús Marco-Galindo, Àngels Rius Gavidia, M.Elena Rodríguez González
Estudios de Informática, Multimedia y Telecomunicación
Universitat Oberta de Catalunya
Rambla del Poblenou, 156. 08018 Barcelona
{jmarco, iguitarth, mmarcog, mriusg, mrodriguezgo}@uoc.edu

Resumen

Se describe el trabajo realizado por un grupo pluridisciplinar de once profesores del área de desarrollo del software (programación, bases de datos, ingeniería del software y sistemas de información) de la Universitat Oberta de Catalunya (UOC) en relación a la organización de la enseñanza de esta área. El trabajo ha estudiado las posibilidades de reordenar e incorporar nuevos contenidos, además de detectar solapamientos y contenidos que se alejan de las necesidades actuales de formación a nivel universitario.

En este artículo se expone por un lado el proceso de reflexión y por otro el resultado obtenido en forma de mapa del área y la justificación, tanto de los cambios propuestos, como de las decisiones de mantenimiento de algunos aspectos más clásicos.

Las conclusiones y las propuestas presentadas más que ser definitivas, pretenden estimular el desarrollo de reflexiones similares en el contexto de cambio que implica el EEES.

1. Motivación

Tradicionalmente, la docencia de los aspectos de desarrollo del software (DS) se ha impartido siguiendo una visión ascendente, es decir, empezando por la programación (P) hasta culminar con las tareas más complejas de la ingeniería del software (IS). El origen de este enfoque posiblemente hay que buscarlo en el propio desarrollo histórico del software. Sin embargo, en la actualidad este enfoque ascendente es precisamente inverso al proceso del DS y de los ciclos de vida indiscutiblemente aceptados [5].

En otras áreas de la informática, como por ejemplo la de redes de computadores, la docencia también había seguido este enfoque ascendente, a pesar de que el interés profesional de un estudiante fuera justo el contrario: mucho más centrado en la utilización de las capas OSI medias y altas que en el movimiento de los bits por la red. Hoy por hoy, ya existen propuestas para el área de redes de computadores [3] que abandonan el enfoque tradicional planteando una nueva visión descendente, visión que se está implantando en muchos currículos universitarios y que supone, en ocasiones, la pérdida de relevancia de algunos contenidos y/o la aparición de otros nuevos.

Por otro lado, parece claro que las discusiones actuales sobre: a) hasta qué punto un ingeniero en informática debe formarse en programación [4]; b) cómo adaptar los currículos de las diferentes áreas de la informática al EEES; c) cuáles son las competencias diferenciales de un ingeniero versus otros profesionales del sector; favorecían una reflexión acerca de la docencia en el área del DS, intentando descubrir aspectos a descartar y nuevas necesidades de formación, así como plantear nuevos itinerarios formativos de adquisición de las competencias dentro del currículo.

Así pues, con el objetivo final de plantear la docencia del DS desde lo general a lo particular, esto es, de manera descendente, un grupo de profesores de la UOC ha realizado a lo largo de 2006 una serie de sesiones de trabajo y discusión. El proceso de reflexión así como los resultados obtenidos se exponen en este artículo. Así, la sección 2 explica la situación actual de la docencia en el ámbito del DS en la UOC; la sección 3 resume el proceso de reflexión seguido y propone un nuevo mapa del área. Y la sección 4 expone las conclusiones y el trabajo futuro.

2. Punto de partida

En general, y también en el caso de la UOC como puede verse en la Figura 1, la docencia en el

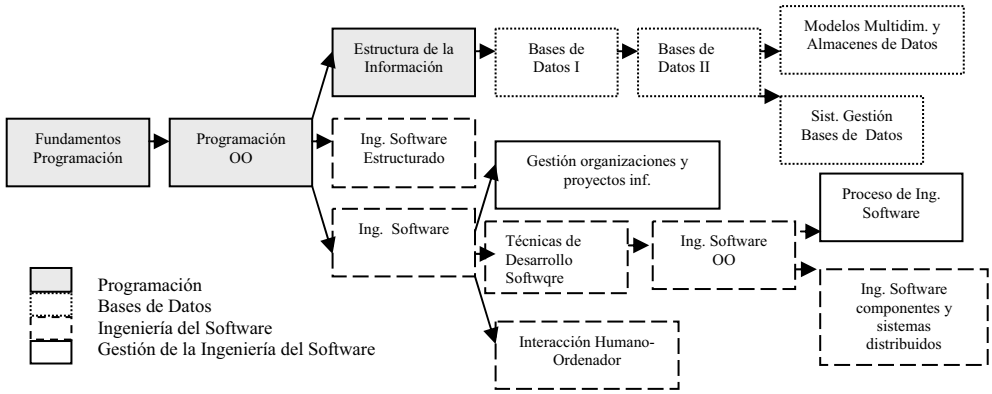

Figura 1. Organización curricular actual de DS en la UOC

ámbito del DS agrupa las competencias en dos grandes grupos: por un lado, las relativas a P, y por otro las que se sitúan en la IS, entendida en un sentido amplio (es decir, incluyendo los aspectos de bases de datos (BD) y de gestión del proceso de desarrollo del software (G)).

Los objetivos *propios* de aprendizaje de P se podrían sintetizar en los siguientes:

- PO1. Conocer y dominar las herramientas básicas de programación que permitan la concepción de algoritmos correctos.
- PO2. Saber definir las estructuras de datos necesarias para dichos algoritmos.
- PO3. Conocer y saber llevar a cabo el proceso de codificación-compilación-prueba de programas.
- PO4. Tener conocimiento de los paradigmas de P (estructurado, orientado a objetos, distribuido) para escoger el más adecuado, dado un entorno de programación y un problema concreto a resolver.

Por su parte, los objetivos *propios* de la IS podrían resumirse en:

- ISO1. Saber analizar requisitos de cualquier sistema de información.
- ISO2. Saber diseñar soluciones, tanto desde el punto de vista del proceso (IS), como desde el punto de vista de los datos (BD).
- ISO3. Conocer los paradigmas relacionados, y saber adaptarlos a cada caso concreto.
- ISO4. Ser capaz de llevar a cabo la planificación y control de proyectos (G).

Asimismo, el estudiante de DS debe desarrollar una serie de *habilidades genéricas* a P y a IS como son la capacidad de abstracción, la capacidad lógica y de análisis, y la capacidad para seguir métodos y aplicar técnicas generales a casos concretos. Por supuesto, estas habilidades, aún siendo comunes a P e IS, se aplican a problemas de diferente magnitud y ámbito. También puede observarse que la adquisición de estas capacidades genéricases objetivo de asignaturas de otras áreas del currículo (lógica, matemática discreta y/o informática teórica) que posiblemente el estudiante no haya cursado completamente cuando afronta las asignaturas de DS.

Por otro lado, intrínsecos al área de DS, los paradigmas de desarrollo (objetivos PO4 y ISO3), también son comunes a las dos áreas y, consecuentemente, en ambas se realizan esfuerzos para introducirlos. Ocurre, sin embargo, que estos esfuerzos no están siempre alineados.

Otra observación necesaria es que en los primeros cursos de P el estudiante además de aprender los conocimientos básicos, debe trabajar simultáneamente aspectos relativamente alejados: por un lado los objetivos más metodológicos (PO1, PO2 y PO4), y por otro el objetivo de habilidad práctica (PO3).

Todas estas consideraciones, junto con una organización curricular que prima la introducción rápida de PO3, y la distribución de objetivos que requieren maduración y practica como son PO1, PO2 y PO4 en asignaturas de alrededor de trece semanas útiles de docencia, pueden explicar en parte la dificultad para lograr todos los objetivos

de aprendizaje así como el bajo rendimiento de estas materias en los primeros cursos.

Tras constatar esta visión, las primeras preguntas que se plantearon fueron las siguientes:

- ¿Podríamos darle la vuelta al aprendizaje del DS empezando por la parte común (habilidades de análisis, diseño y paradigmas) y posponiendo la introducción de PO3?
- Esta inversión del orden de aprendizaje, ¿evitaría repeticiones y solapamientos de objetivos y contenidos entre distintas asignaturas, y, por tanto, permitiría disponer de más tiempo para profundizar en el aprendizaje de ciertas competencias?
- ¿Existen otras asignaturas en los currículos que, rediseñadas, puedan jugar un papel relevante en este replanteamiento?

3. El proceso de reflexión

Como esquema de trabajo para iniciar una reflexión más profunda que permitiera responder a las cuestiones anteriores (y a otras que pudieran aparecer posteriormente), se planificaron las siguientes fases que, mediante un proceso de aproximaciones sucesivas, nos debían llevar a nuevas propuestas y a posibles resultados:

- F1. Buscar experiencias innovadoras de ingeniería curricular en relación al DS.
- F2. Profundizar más allá de la visión tradicional del área de DS, buscando un cierto consenso sobre las debilidades y oportunidades de la docencia actual.
- F3. Plantear cambios y reorganizar los objetivos, con el fin de esbozar una posible ordenación curricular.
- F4. Establecer un mapa de contenido que facilite su inclusión en la organización que resulte de la aplicación del EEES.

El equipo de profesores implicados incluía personas de los cuatro ámbitos indicados: dos de P, dos de BD, tres de IS y dos de G, así como dos más de asignaturas de síntesis. El trabajo se realizó en seis sesiones conjuntas, cada una de tres horas, para las que previamente cada profesor, individualmente o en grupo, había realizado un trabajo/análisis previo según las directrices que se marcaban. Durante las sesiones, se ponían en común las reflexiones de cada profesor o pareja (mediante un sistema de tarjetas *colgables* en una pared o pizarra) y se sometían a discusión con el resto de profesores. Aunque, evidentemente, las discusiones detalladas sobre objetivos y contenidos a menudo no podían producirse entre profesores de las distintas subáreas, sí que permitían emerger ciertas ideas preconcebidas, a la vez que fluían otras más novedosas, fruto del "*brainstorming* controlado" que surgía de la interacción entre profesores de diferentes especialidades y sensibilidades.

A continuación se resumen los aspectos abordados en cada fase. No se entra, por motivos de espacio, ni en el detalle de cada sesión, ni en su dinámica, a veces diferente de la prevista.

3.1. F1: Buscando otras experiencias

Consultada la organización curricular de universidades que nos son cercanas (UPC, UPM, UPV, UCIII), es importante destacar que no encontramos experiencias reales de implantación que abordaran este tema desde una perspectiva similar a la que nos planteábamos.

Como era de esperar, sí que se encontraron algunas experiencias expuestas en anteriores ediciones de las JENUI que planteaban, bien desde otra perspectiva, bien parcialmente, temáticas parecidas [1].

En el contexto internacional la fuente más clara es el currículo sobre ingeniería del software propuesto por la ACM [2]. En una de sus estructuras curriculares propone empezar por IS. Sin embargo, el análisis de su propuesta permite constatar que dentro de IS incluye P, así que, de hecho, realmente empieza por P.

3.2. F2: Más allá de la visión clásica

Para intentar superar los tópicos más habituales del DS se intentó que dos cuestiones nos acompañaran durante toda la reflexión, a pesar de que, entre nosotros, tampoco existía un consenso claro sobre ellas.

- ¿Es necesario retrasar la codificación? ¿Es ventajoso o contraproducente "codificar-compilar-probar" desde el primer día? ¿Entenderían los estudiantes no hacerlo así?
- ¿Es posible empezar directamente introduciendo en genérico el análisis de

problemas y el diseño de soluciones sin conocer ninguna de las *palabras nucleares clave* de P e IS (como *variable, parámetro, tipo, procedimiento, compilar, depurar, entorno, descomponer, llamar, abstraer*…)?

Empezamos, pues, revisando conjuntamente y de forma plenaria todos los contenidos que impartíamos en el área del DS, con el fin de confirmar o no la visión más clásica y, sobre todo, de descubrir otros problemas de los que no éramos conscientes. Y surgieron los siguientes aspectos:

- La aparición explícita de todo el ciclo de vida de IS no aparece hasta la primera asignatura de IS, esto es, hasta claramente avanzada la carrera.
- El estudio sucesivo de UML, desde diferentes perspectivas o usos. Esto es, el riesgo de repetición de contenidos y, de nuevo, la falta de visión unitaria.
- La aparición reiterada de los conceptos de eficiencia y calidad en P, IS, BD, G.
- La repetición del modelado de datos tanto en BD como en IS, con el problema añadido de explicarse con métodos diferentes.
- La repetición de algunos contenidos en BD e IS, debido a la historia de desarrollo y puesta en marcha de algunas asignaturas.
- La falta de conexión entre IS y P. En IS se llega hasta el diseño que es el punto de partida de P, pero no se establece el enlace entre las dos áreas de manara natural.
- El exceso de contenido con poco tiempo para ser asumido, sobre todo en P. No se trata tanto de recortar el contenido, como de disponer de más tiempo para su asimilación.
- La aparición de temas de implementación muy *tecnológicos* (tanto en BD como en P) como, por ejemplo, aspectos internos de SGBD o de compiladores.
- La no inclusión de conceptos sobre diseño y programación distribuida hasta prácticamente las asignaturas de síntesis final.
- La incorporación de interacción de persona-ordenador como un aspecto colateral, no integrado en todo el ciclo de vida.
- La confirmación de la necesidad de disponer de asignaturas de síntesis que cubran todo el ciclo de vida del DS.

3.3. F3: Planteando los cambios

Teniendo en cuenta todos los aspectos surgidos en las dos fases previas, se planteó, para los cuatro ámbitos (P, IS, BD, G), una discusión liderada por los profesores especialistas de cada uno y confrontada con una actitud crítica del resto de profesores. El objetivo era proponer y debatir los cambios de cada subárea.

Para IS:

- El principal cambio que se plantea, consiste en limitar el estudio del paradigma estructurado, con el fin de maximizar la dedicación al paradigma orientado a objetos. Aunque este mayor énfasis en las metodologías de orientación a objetos ya se da en nuestra organización curricular actual, el remanente histórico de dedicación a las metodologías estructuradas aún es alto.
- Se propone un primer bloque curricular que tenga por objetivo la introducción de los conceptos de la IS y la importancia de garantizar la *calidad* en todo el proceso de DS, siempre *centrado en el usuario* –requisitos e interacción-, para continuar después con el análisis del sistema basado en UML.
- En un segundo bloque se trabajarían los objetivos de diseño para continuar con la IS distribuida (centrada en aspectos de diseño, pues los de análisis ya se han estudiado y sólo sería necesaria su revisión).

Para BD:

- En esta área la propuesta de cambio consiste más en reorganizar objetivos que en incluir nuevos.
- Los objetivos fundamentales incluyen el estudio del modelo de datos relacional y el álgebra relacional, los SGBD, aspectos de concurrencia desde un punto de vista de diseño, SQL (interactivo y programado), creación de BD partiendo de UML, bases de datos distribuidas, desnormalización y almacenes de datos. Como objetivos fundamentales de nueva incorporación se incluyen datos semiestructurados y XML.

- Como objetivos de aprendizaje optativos se proponen conocer los aspectos relacionados con el control de concurrencia y recuperación en BD, optimización y *tunning*. Como objetivos de nueva incorporación, conocer alternativas a las BD relacionales.

Para P:

- Las habilidades en programación, entendida como la técnica de codificación-ejecución-prueba-depuración que se implementa con diferentes lenguajes, deberían tener su propio lugar. Este lugar, en ocasiones llamado *tecnología de la programación*, debería ser independiente del otro lugar propio de P, el de la *algorítmica*. De esta forma, se podría intentar desactivar la discusión clásica sobre qué lenguajes utilizar en cada asignatura (Java, C, Pascal, ensamblador).
- En este lugar reservado a la técnica de la programación, se debería insistir en la búsqueda de la calidad del software, a partir de la importancia de la representatividad de las pruebas a realizar, la depuración y las cuestiones de eficiencia.
- Para facilitar la comprensión de estructuras de datos complejas, se confirma la necesidad de reintroducir conceptos de recursividad.
- La aceptación de esta área de *tecnología de la programación* separada de los contenidos sobre *algorítmica*, facilitará la reformulación y reorganización de las competencias objetivo de P
- Entre las ventajas de empezar por IS en lugar de por P, están las de incluir en el currículo una sola vez el estudio de las características de la OO, para luego ir desarrollándolas en amplitud y profundidad según las necesidades particulares de P y de IS. Adicionalmente, otra ventaja sería que el estudiante ya comprendería un análisis UML, en el momento en que tuviera que afrontar el aprendizaje del diseño e implementación de la solución.
- Por otra parte, parecería natural tender hacia una dedicación de dos cursos académicos completos para asimilar los conceptos de algorítmica, y de tecnología de la programación.

Para G:

- En la organización curricular actual los aspectos de calidad ya se abordan aunque en las asignaturas finales. Por eso sería necesario asegurar que se introducen antes, una vez conocido el ciclo de vida así como generalizarlos.
- Tradicionalmente, los contenidos propios de esta subárea se hayan mezclados con otros más cercanos a la dirección de sistemas y tecnologías de la información (STI). Se deben considerar temas relacionados, aunque no incluidos propiamente en el desarrollo del software. La razón de que tiendan a mezclarse está en que a menudo en el entorno profesional, un jefe de de departamento de STI acostumbra a haber sido, o a ser a la vez, responsable de proyectos de desarrollo de software.

3.4. F4: Dibujando un nuevo mapa

Del debate de F3 sobre las cuatro subáreas, parecía que se podía esbozar alguna respuesta a las preguntas inicialmente planteadas (ver final sección 2).

Así, si aceptamos la necesidad de disponer de más tiempo para aprender las habilidades expuestas para P, posponer la enseñanza de P hasta haber introducido IS puede afectar a la planificación curricular de otras áreas de conocimiento. En consecuencia, además de las dudas que plantea iniciar la docencia con una visión descendente, parece claro que hay otros condicionantes, como la estructura curricular, las expectativas de los estudiantes, y la necesidad de compartir las competencias de programación con otras asignaturas (p.e. redes o sistemas operativos). Posiblemente, pues, la reordenación debería pasar por realizar en paralelo la docencia de IS y P, en lugar de invertirla.

Adicionalmente, sin embargo, sí que podría ser posible dar una visión global de todo el DS, introduciendo los conceptos fundamentales y creando un marco de referencia general en la primera asignatura de DS. Este marco de referencia sería el hilo conductor al que el estudiante acudiría durante la profundización en las cuatro subáreas y en sus materias asociadas,

ayudándole a tener siempre presente esta visión descendente del proceso de DS.

Para dibujar el mapa final, se decidió aplicar las siguientes simplificaciones:

- La idea inicial era hacer un mapa genérico de competencias del área con independencia de en qué oferta formativa se incluyeran (ingeniería, ingeniería técnica, grado, máster). Sin embargo, resultaba muy difícil no tener en cuenta los objetivos/competencias diferenciales de cada oferta formativa. Así pues se optó por plantear un mapa para un hipotético currículo de *Grado en Informática*.
- Otra simplificación fue describir el mapa en forma de contenidos como se había venido haciendo hasta ahora y no de competencias. La experiencia en la descripción por competencias era desigual entre el grupo de profesores y desviaba la atención del objetivo principal del trabajo. Más adelante ya se plantearía una descripción por competencias en consonancia con las directrices del EEES.
- Calcularlo en créditos tradicionales, no ECTS. La experiencia en el cálculo de ECTS era también demasiado heterogénea en el grupo de profesores y podía dificultar la comprensión del resultado global.
- Establecer unidades de aprendizaje mínimas (a fin de no pensar en asignaturas) que facilitaran su ensamblaje posterior en la organización temporal que finalmente resultara del grado EEES. Estas unidades las definimos como "el conjunto máximo de contenidos mínimos (indivisibles) desde el punto de vista docente en un periodo lectivo y para una titulación de Grado".

Finalmente, el mapa resultante se presenta en la Figura 2 (unidades mínimas con indicación de su peso en créditos convencionales, así como su interrelación) y en la Figura 3 (contenidos asociados a cada unidad de aprendizaje mínima).

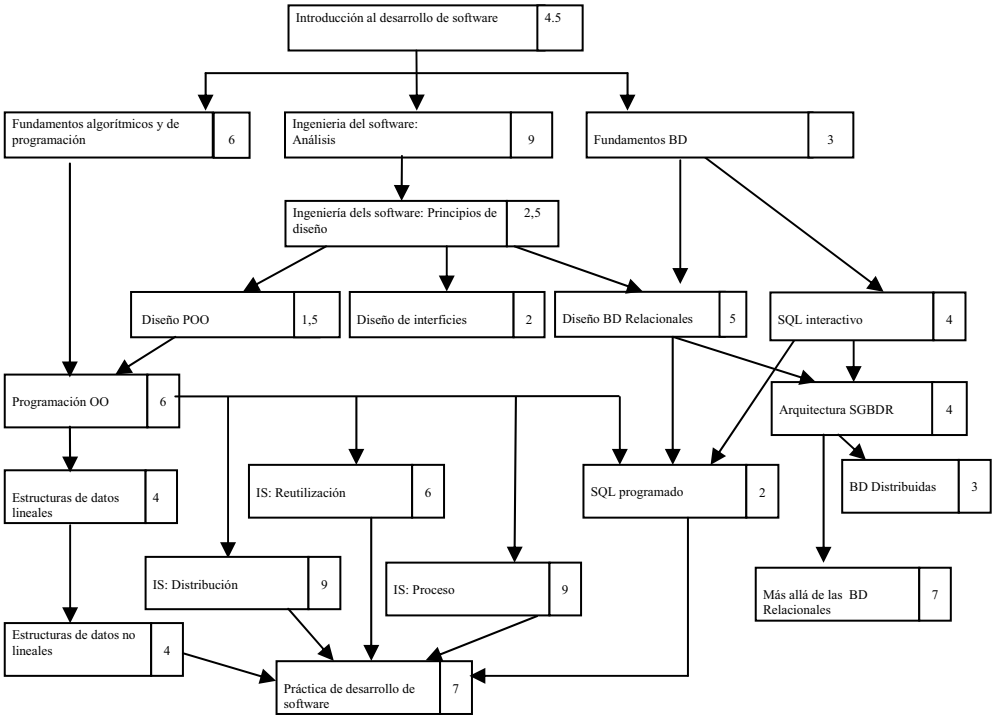

Figura 2. Mapa de contenidos propuesto

Introducción al desarrollo del software	- Etapas del ciclo de vida - Metodologías de desarrollo de software (estructurada, orientada a objetos, distribuida) - Algorítmica básica (control de flujo, tipos elementales, entrada/salida) - Tecnología de la programación básica (Codificación, depuración, test en C y pseudocódigo)
Fundamentos algorítmicos y de programación	- Estructuras de datos: Tablas y tuplas - Secuencias: Recorrido y búsqueda - E/S: Ficheros - Eficiencia temporal y espacial
Ingeniería del software: análisis	- Características OO - Metodología (Unified Process) - Captura de requisitos y análisis (UML y OCL) - Testing funcional - Análisis de la persistencia (Diseño conceptual UML. Modelo de datos)
Ingeniería del software: principios de diseño	- Diseño de casos de uso - Diseño arquitectónico - UML asociado
Fundamentos de BD	- Principios de la gestión de la información y evolución histórica - Funcionalidades de un sistema gestor de BD - Modelo de datos relacional y álgebra relacional.
Diseño POO	- Consolidación diseño (adaptación al lenguaje de programación)
Programación OO	- Justificación de la programación OO - Codificación POO (traducción del diseño a código OO). Herramientas Java. - Pruebas unitarias y de integración - Interfaces
Diseño de Interfaces	- Diseño interfaz de usuario en el contexto de la ingeniería del software
Diseño BD relacionales	- Diseño lógico - Normalización y desnormalización - Diseño físico (índices, tunning, esquema interno)
SQL interactivo	- Creación BD y tablas, consultas y cambios de la BD (inserción, borrado, modificación) - Vistas, disparadores, procedimientos almacenados, transacciones, autorizaciones y roles
SQL programado	- JDBC y SQL-J
Estructuras de datos lineales	- Introducción a los TAD - Revisión de eficiencia - Secuencias: pilas, colas y listas - Colas prioritarias - Librería de TAD en Java
Estructuras de datos no lineales	- Recursividad - Árboles, funciones y conjuntos, relaciones y grafos - Diseño de TADs complejos - Librería de TADs de Java
Arquitectura sistemas gestores de BD relacionales	- Gestión de vistas - Gestión de la seguridad - Optimización de consultas - Control de concurrencia y recuperación BD
Más allá de las BD relacionales	- BD OO y BD Object-relational - Almacenes de datos (multidimensionales, FIC, OLAP) - Datos semiestructurados y XML
BD distribuidas	- Acceso a BD distribuidas - Arquitectura SGBD distribuidas
IS: Distribución	- Especificación (distribución en general: RM-ODP) - Arquitecturas distribuidas - Ingeniería de componentes (UML asociado) - Diseño distribuido OO (UML asociado) - Tecnologías de implementación distribuidas
IS: Reutilización	- Principios y objetivos de la reutilización - Patrones y componentes - Otras técnicas de reutilización
IS: Proceso	- Calidad - Gestión de la configuración - Mantenimiento - Métricas
Práctica de desarrollo del software	- Aplicación práctica global de desarrollo de software

Figura 3. Relación de contenidos