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

Josep Ma. Marco-Simó, Isabel Guitart Hormigo, Mª Jesús Marco-Galindo, Angels Rius, Ma. Elena Rodríguez González, Joan Arnedo-Moreno, Jordi Cabot, Santi Caballé, Daniel Riera

Estudios de Informática, Multimedia y Telecomunicación, Universitat Oberta de Catalunya, Barcelona

<{jmarco, iguitarth, mmarcog, mriusg, mrodriguezgo, jarnedo, jcabot, scaballe, drierat} @uoc.edu>

Este artículo fue seleccionado para su publicación en *Novática* entre las ponencias presentadas a las XIII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2007) celebradas en Teruel de las que ATI fue entidad colaboradora.

Resumen: *con la llegada del Espacio Europeo de Educación Superior (EEES) a las universidades españolas, el ámbito de conocimiento de desarrollo del software, al igual que otros, tiene una interesante oportunidad para replantear su organización curricular, así como las competencias que pretende desarrollar en los estudiantes. Un grupo pluridisciplinar de once profesores de éste ámbito (de las áreas de programación, bases de datos, ingeniería del software y sistemas de información) de la Universitat Oberta de Catalunya se ha planteado la posibilidad de reordenar, invirtiéndola, la aproximación a las competencias de este ámbito, incorporando nuevos contenidos y eliminando solapamientos y conceptos que se alejan de las necesidades actuales de formación a nivel universitario. El proceso de análisis descrito en este artículo así como los resultados propuestos no pretenden ser definitivos sino estimular el desarrollo de reflexiones similares que ayuden a mejorar la calidad de la enseñanza en el contexto de cambio actual.*

Palabras clave: *desarrollo de software, Espacio Europeo de Educación Superior, organización curricular.*

1. Motivación

Tradicionalmente, la docencia de los aspectos que intervienen en el desarrollo del software (DS) se ha impartido siguiendo una visión ascendente, es decir, empezando por la programación (P), siguiendo con los aspectos de análisis y diseño de Ingeniería del Software (IS) -en paralelo a menudo con los de bases de datos (BD)- hasta finalizar con las tareas de gestión de proceso del DS (G). El origen de este enfoque posiblemente hay que buscarlo en la propia evolución histórica del proceso de DS. Sin embargo, en la actualidad este enfoque ascendente es precisamente inverso al proceso típico del DS y de los ciclos de vida ampliamente aceptados [1][2].

En otras áreas de la informática la docencia también había seguido este enfoque ascendente. En el caso, por ejemplo, de la de redes de ordenadores se empezaba (y se centraba) la docencia en las capas más bajas del modelo OSI a pesar de que el interés profesional sobre el tema para un ingeniero informático fuera precisamente el contrario (las capas altas). Hoy por hoy, ya existen propuestas para esta área [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 informática al EEES; y c) cuáles son las competencias diferenciales de un ingeniero informático respecto a otros profesionales del sector; favorecían una reflexión acerca de cómo articular y ordenar la docencia en el ámbito del DS, intentando determinar qué aspectos más obsoletos descartar y qué nuevas necesidades de formación incluir.

Así pues, con el objetivo final de plantear la docencia del DS desde lo general a lo particular, un equipo de once profesores de nuestra universidad provenientes de las cuatro áreas en que consideramos constituido el ámbito del DS (P, BD, IS y G) y dispuestos a realizar y someterse a crítica constructiva, participaron en unas sesiones conjuntas en las que se replantearon los tópicos habituales de cada área (para defenderlos o rebatirlos), intentando al mismo tiempo concretar nuevas propuestas.

2. Búsqueda de experiencias similares

En primer lugar intentamos localizar si existían propuestas previas en este sentido. En nuestro contexto, consultada la organización curricular de universidades *próximas* (Universitat Politècnica de Catalunya, Universidad Politécnica de Madrid, Universidad Politécnica de Valencia, Universidad Carlos III), no supimos encontrar experiencias reales de implantación que abordaran este tema. Sí se encontraron algunas, como era de esperar, en las actas de las *Jornadas de Enseñanza Universitaria de la Informática (JENUI)* que planteaban temáticas parecidas, aunque, o bien lo hacían desde otra perspectiva, o bien lo hacían parcialmente. De entre ellas, queremos destacar la de Cachero et al. [5] donde se apunta que "*la falta de interconexión explícita en los temarios de ambas asignaturas* [Programación Orientada a Objetos e Ingeniería del Software] *crea en los alumnos lagunas de conocimiento y menoscaba su motivación a la hora de aprender técnicas de diseño, que no perciben como verdaderamente útiles para la codificación de software*", afirmación con la que estamos, como se verá posteriormente, claramente de acuerdo.

En el contexto internacional la fuente más clara es el currículo sobre ingeniería del software propuesto por la ACM [6]. En una de sus estructuras curriculares propone empezar por IS. Sin embargo, un análisis más profundo de su propuesta permite constatar que dentro de IS incluye P, así que, de hecho, realmente empieza por P.

Mención aparte merecen las aportaciones de Meyer quien ya en 1993 [7] asumía para la enseñanza del DS la filosofía del *inverted curricula* o *progressive opening of black boxes* propuesto por Cohen para el ámbito de la ingeniería eléctrica. No es tanto una visión *top-down* como una visión *outside-in*, esto es, basada en introducir al estudiante primero como consumidor-usuario de software para ir madurando hacia un rol de productor-desarrollador.

Sus propuestas se basan en universalizar el uso exclusivo y sin ambigüedades de la orientación a objetos, con el diseño por contrato como estrategia y apoyado en un potente entorno de librerías de trabajo. Sin embargo parece que han quedado principalmente limitadas a los cursos iniciales de programación [8] a pesar de que ya ha expuesto unas interesantes líneas maestras para un diseño curricular completo [9].

Independientemente de su alcance final real, las razones que alega Meyer sobre la necesidad de replantear profundamente la organización curricular de la enseñanza del DS nos resultan también muy cercanas.

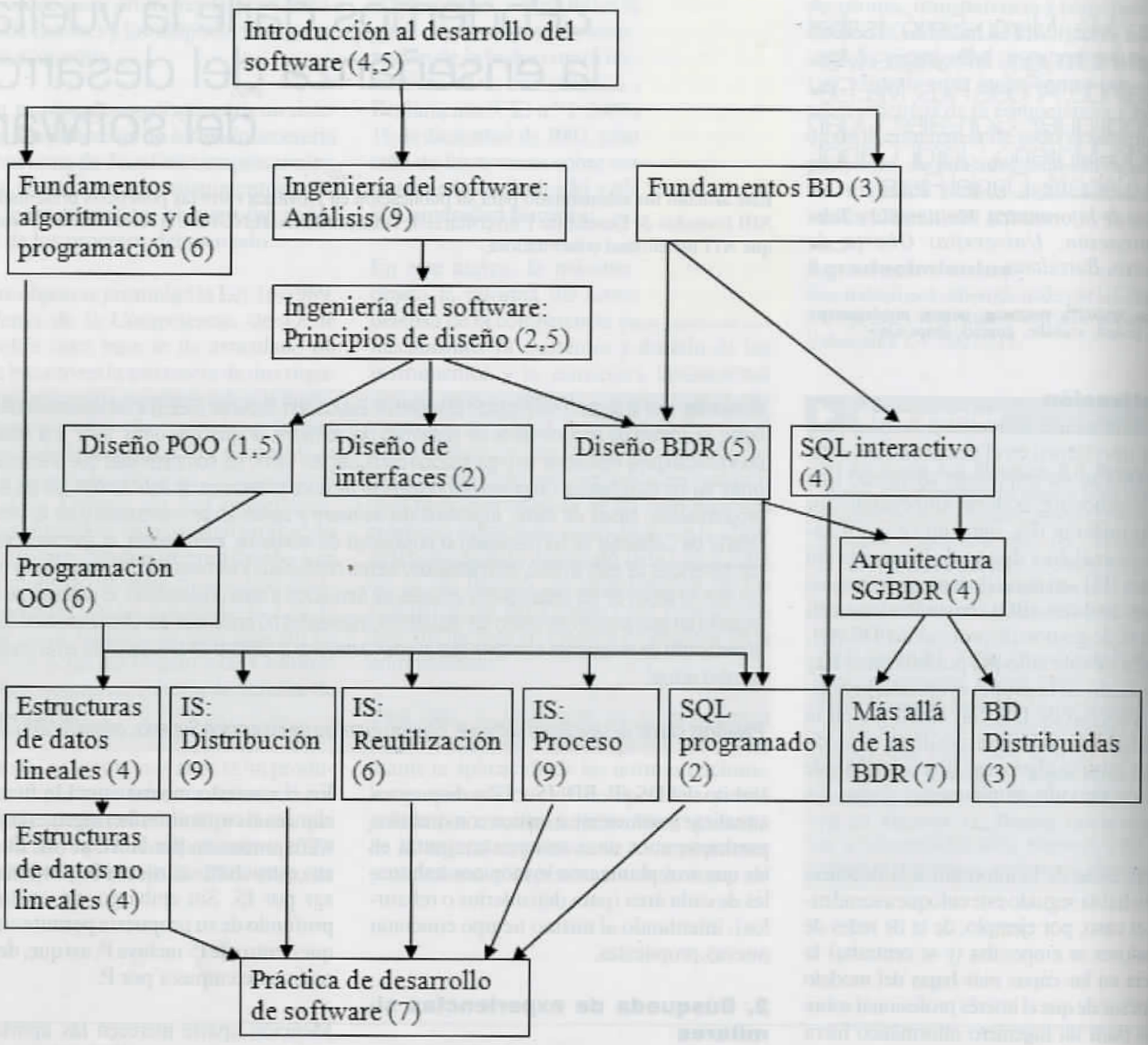

Figura 1. Mapa del ámbito.

3. Críticas a la organización curricular actual

En una segunda fase revisamos todos los contenidos que impartíamos con el fin de intentar hacer explícitos los problemas (conocidos ya o no) de la organización actual. Detectamos:

- La tardía descripción explícita de todo el ciclo de vida del DS, que no aparece hasta la primera asignatura de IS, esto es, hasta claramente avanzada la carrera.
- La aparición reiterada y desconectada del paradigma de orientación a objetos y de la notación UML tanto en P como en IS.
- La aparición reiterada y desconectada de los conceptos de eficiencia y calidad en P, IS, BD y G.
- La repetición de explicaciones sobre modelado de datos en BD y IS, con el problema añadido de abordarse con métodos y notaciones diferentes.
- La introducción en P y BD de contenidos propios de IS que no pueden justificarse suficientemente y que, por simplificados, a menudo derivan en malos hábitos o concepciones.
- La falta de orden en la organización de la docencia que *parece* ascendente pero que no lo es estrictamente: después de P se sube a IS pero no por el diseño sino por el análisis. Esta falta de orden redunda en una falta de conexión entre IS y P dado que no se presenta el enlace entre las dos áreas de manera natural.
- La introducción inmediata de P y en pocas semanas útiles, exigiendo la adquisición de un conjunto de competencias que requieren un tiempo de maduración superior.
- La exigencia de que el estudiante de P trabaje simultáneamente aspectos relativamente alejados: por un lado los propios de la concepción algorítmica (estructuras de control y de datos, acciones y funciones) y por otro los propios de la tecnología de la programación (codificación, compilación, pruebas).
- En BD la aparición de temas muy *tecnológicos* de implementación interna de SGBD.
- La no inclusión de conceptos sobre diseño y programación distribuida hasta prácticamente las asignaturas de síntesis final.
- La incorporación de interacción persona-ordenador y del diseño centrado en el usuario como un aspecto colateral, no como un aspecto integrado en todo el ciclo de vida del DS.
- La necesidad de que el estudiante de DS desarrolle una serie de habilidades genéricas y comunes a P y a IS -como son la capacidad de abstracción (clave para muchos en la profesión [10]), la capacidad lógica y de análisis, y la capacidad para seguir métodos y aplicar técnicas generales a casos concretos- que son también objetivo competencial de otras áreas del currículo (lógica, matemática discreta y/o informática teórica, entre otras) pero que posiblemente el estudiante no haya cursado completamente cuando afronta las asignaturas de DS.

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 - 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 del 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

Tabla 1. Detalle de contenidos de los objetivos competenciales.

4. Diseño de un nuevo mapa

Vistas estas debilidades, y discutidas y analizadas las opciones de cambio en la organización curricular que permitieran superarlas, ¿qué respuesta damos a nuestra pregunta sobre la posibilidad de invertir la enseñanza del software?

Por un lado, si aceptamos la necesidad de disponer de más tiempo para aprender las habilidades expuestas para P, es evidente que posponer la enseñanza de P hasta haber introducido IS puede afectar a la planificación curricular de otras áreas de conocimiento (e.g. redes o sistemas operativos) donde son necesarias unas habilidades mínimas. Por otro lado, la experiencia docente nos hace *percibir* (sólo intuitivamente) que los estudiantes no tienen suficiente maduración para afrontar, al inicio de su formación universitaria, temas muy abstractos (propios de la IS).

Adicionalmente, parece claro que los estudiantes, al incorporarse a los estudios, tienen muchas expectativas en formarse cuanto antes como programadores, un aliciente profesional. En consecuencia, no parece muy viable invertir radicalmente la docencia en DS. Sin embargo, una posible solución de compromiso entre ambas orientaciones podría pasar por:

- Introducir, en primer lugar, una visión global de todo el DS, presentando 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 áreas, ayudándole a tener siempre presente esta visión descendente del proceso de DS.
- Superada esta presentación global, iniciar en paralelo la docencia de IS y P, en lugar de invertirla estrictamente.

Con todo esto, el dibujo final del mapa propuesto para un hipotético currículo de Grado en Informática es el que se presenta en la **figura 1**. Esta figura representa unidades mínimas de aprendizaje (no asignaturas) con indicación de su peso en créditos ECTS, así como su interrelación. Esta figura se complementa con la **tabla 1** que indica los contenidos de los objetivos competenciales asociados a cada unidad de aprendizaje mínima.

5. Conclusiones

Las conclusiones más relevantes, resultado de la propuesta realizada así como de la organización, dinámica y discusión (imposible de transcribir aquí por motivos de espacio) de las sesiones realizadas, podríamos resumirlas en:

- La necesidad de dar una visión global del proceso descendente del DS se puede cubrir mediante la creación de un contenido inicial marco que sirva de referencia al estudiante.
- Diferentes motivos (la dependencia curricular con otras áreas de conocimiento, las expectativas de formar rápidamente a programadores, las percepciones sobre las capacidades de los estudiantes y las propias dudas de los equipos docentes sobre la viabilidad del replanteamiento) desaconsejan la inversión drástica del orden de docencia (empezar por IS y seguir por P).
- Sin embargo, aunque no se produzca esta inversión en el orden, sí puede realizarse en paralelo la introducción tanto de IS como de P. Esto permite reforzar e insistir en los aspectos que son comunes y fundamentales, minimizar las repeticiones de contenido y facilitar la incorporación como un continuo de aspectos hasta ahora no tratados en forma integrada como la calidad del software o el diseño centrado en el usuario.
- Es necesario reducir el tiempo dedicado a los temas que ya no se consideran demasiado útiles para el profesional actual (metodologías de IS estructuradas, aspectos excesivamente teóricos de algorítmica, etc.).

Constatamos también que nuestra propuesta del mapa del ámbito incorpora un volumen de créditos que nos lleva a un grado claramente orientado al DS, con lo que, para un planteamiento más generalista, sería necesaria su revisión.

En cuanto al proceso seguido para llegar a esta propuesta, las mayores dificultades que nos hemos encontrado podríamos resumirlas en:

- La ausencia de experiencias similares obliga a superar reticencias al cambio en la acción docente.
- El conocimiento de las dificultades del estudiante para afrontar temas que requieren capacidad de abstracción no ayuda a ser más atrevidos en el nuevo diseño curricular.
- Otras asignaturas deben contribuir a aportar las bases para la enseñanza del DS. Esta discusión, que implica reconsiderar contenidos de otras áreas de conocimiento, requiere una estrategia compleja para que sea operativa.

En cuanto a las posibilidades de implantación real de una propuesta de este estilo y de la superación de las limitaciones percibidas, sólo tres preguntas finales, quizá de difícil respuesta:

- ¿Cuántas de estas limitaciones quedarían superadas en un grado muy centrado en IS, como el propuesto, por ejemplo, por Martín y Ruiz [11]?
- ¿Cuántas de las ideas sobre lo que quiere el estudiante (aprender a programar) o es capaz de hacer (aplicar abstracciones y capacidad de análisis) no son más que tópicos? Si en otros ámbitos más clásicos se logra que los estudiantes aprendan conceptos mucho más complejos o abstractos y los apliquen, ¿por qué no es posible hacerlo en el, además, estimulante ámbito del DS?
- ¿Qué esfuerzo de coordinación y sacrificio puede (o quiere, o está autorizado a) realizar el profesorado universitario para asegurar la coherencia docente de este extenso ámbito?

Por supuesto, no pretendemos haber planteado una propuesta definitiva, sino sólo volver a promover reflexiones similares entre los foros de profesorado. Sin duda, las revisiones que implica el EEES son otra ocasión para reconsiderar el papel en los currículos universitarios de este fundamental ámbito del DS.

Agradecimientos

Los autores quieren agradecer las indicaciones de los revisores de las JENUI así como a las aportaciones, preguntas y comentarios recibidos en su presentación en dichas jornadas. Asimismo quieren mencionar a los profesores de la UOC, Atanasi Daradoumis y Enric Mor, quienes formaban parte, activa y fundamental, del grupo de trabajo implicado en la experiencia aquí descrita.

Referencias

[1] I. Sommerville. *Software Engineering* (7ª. Ed.). Harlow: Addison-Wesley, 2005. ISBN: 0321210263.

[2] R.S. Pressman. *Software engineering: a practitioner's approach.* McGraw-Hill Professional, 2005. ISBN: 0072853182.

[3] J.F. Kurose, K.W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet, 2/e.* Addison Wesley, 2005.

[4] N. McBride. *The death of computing (Member view).* British Computer Society, 2007.

[5] C. Cachero, O. López, M.J. Durá. Del Diseño a la Implementación del Software: Una metodología de cohesión. En *X Jornadas de Enseñanza Universitaria de la Informática. Actas del congreso.* Thomson, 2004.

[6] IEEE & ACM. *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering A Volume of the Computing Curricula*, 2004.

[7] B. Meyer. Towards an object-oriented curriculum. *Journal of Object-Oriented Programming, 6(2)*: 76–81, mayo 1993.

[8] M. Pedroni, B. Meyer. The Inverted Curriculum in Practice. *Proceedings of SIGCSE 2006, ACM*, Houston, Texas, 1-5 marzo 2006.

[9] B. Meyer. Software engineering in the academy. *Computer, 34 (5)*: 28-35, mayo 2001.

[10] J. Kramer. Is abstraction the key of computing? *Communications of the ACM, 50 (4)*: 36-42, abril 2007.

[11] G. Martín, E. Ruiz. Ingeniería Informática: ¿más que solo un título, menos que sólo una profesión? *Novática, 188*: 51-63, julio 2007.