Aplicación práctica de un proceso basado en UML

Jesús García Molina, Marcos Menárguez Tortosa, Joaquín Nicolás Ros
Dept. de Informática y Sistemas
Universidad de Murcia
30071 Murcia
e-mail: (jmolina | marcos | jnr)@um.es

Resumen

En las anteriores JENUI de 2002 presentamos una propuesta para organizar la enseñanza de la orientación a objetos a través de dos asignaturas obligatorias, una de *Introducción a la Programación Orientada a Objetos* y otra de *Análisis y Diseño Orientado a Objetos*. Ahora nos centraremos en el principal objetivo de la segunda asignatura: los alumnos deben conocer y aplicar un proceso software basado en UML. En este trabajo discutimos cómo hemos planificado las prácticas de dicha asignatura para conseguir el objetivo formativo propuesto, presentando también una valoración de los resultados.

1. Introducción

En la actualidad, la mayoría de universidades incluyen asignaturas relacionadas con la orientación a objetos, OO. En el caso de las universidades españolas, si se observan los programas de las asignaturas relacionadas con la programación y la ingeniería del software OO, resulta evidente que no existe un consenso en cuanto a la forma de plantear la enseñanza de la construcción de software OO. En las JENUI de 2002 presentamos una propuesta [5] sobre cómo organizar la enseñanza de esa materia y ahora nos centraremos en un aspecto esencial de ella: los alumnos deben conocer y aplicar un proceso software basado en UML.

El objetivo del presente trabajo es describir cómo hemos organizado, durante los últimos tres cursos, la enseñanza de un proceso UML, en concreto el proceso presentado por Craig Larman en su libro *"UML y Patrones"* [7], que se ajusta al Proceso Unificado (UP) [6].

Las aportaciones que pretendemos realizar con el trabajo son: i) mostrar una posible organización docente para enseñar un proceso UML, ii) valorar el proceso de Larman, iii) ofrecer guías que ayuden a aplicar nuestro enfoque o uno similar.

Hemos comprobado que todos los planes de estudio de la titulación *Ingeniero en Informática*, incluyen al menos una asignatura en la que se enseña, de uno u otro modo, "métodos OO" y que casi siempre aparece el libro de Larman en la bibliografía, por lo que creemos que este trabajo puede ser de interés para un buen número de profesores que enseñan una materia que todavía no está bien definida en el plano docente.

El trabajo se organiza del siguiente modo. En la siguiente sección se describe la asignatura en la que se enmarca la enseñanza del proceso UML. En la sección 3 se describe cómo se desarrolla el trabajo práctico de la asignatura y en la sección 4 se ofrece una serie de recomendaciones para cada una de las etapas del proceso. Finalmente se incluyen las conclusiones y las referencias.

2. Una asignatura sobre Análisis y Diseño OO

La propuesta curricular *Computing Curricula 2001* de ACM/IEEE [1] ha reconocido la importancia adquirida por la programación OO en la pasada década y ha añadido este área al núcleo de conocimientos básicos de la disciplina, estableciendo que cualquier estudiante de informática debería tener una formación en Programación OO, Análisis y Diseño OO, y Patrones de Diseño.

En [5] planteamos una organización de la enseñanza de la OO, que se ajustaba a [1], a través de dos asignaturas cuatrimestrales de seis créditos:

Introducción a la Programación OO y *Análisis y Diseño OO*. En la primera se introducen los conceptos básicos que caracterizan al paradigma de programación OO y la segunda se dedica al estudio y aplicación práctica de un proceso software UML y de los patrones de diseño GoF [3]. En [5] se presentó una visión general de la asignatura de *Análisis y Diseño OO* (nos referiremos a ella como ADOO), por lo que en esta sección profundizamos en aspectos relacionados con sus contenidos teóricos, que establecen la base para analizar la organización de las prácticas.

En nuestro caso, ADOO es una asignatura troncal de cuarto curso de seis créditos (3 teóricos y 3 prácticos). Sus contenidos teóricos incluyen los siguientes temas: i) El Lenguaje UML (7 horas), ii) Un proceso basado en UML (8 horas), iii) Patrones de Diseño (12 horas), y iv) Persistencia OO (3 horas).

Actualmente es indiscutible que UML se ha convertido en un lenguaje de modelado estándar y que cualquier proceso software OO debe utilizarlo como notación. Se han definido muchos procesos para UML, siendo RUP (*Rational Unified Process*) [9] el más conocido y extendido. En realidad, RUP es una materialización particular del *Proceso Unificado* (*Unified Process*, UP) [6]. UP establece un proceso más genérico que ha sido ampliamente aceptado, de modo que cualquier proceso basado en UML debería encajar o ser compatible con UP.

Sin embargo, RUP es un proceso muy grande y complejo que no puede ser abordado en el contexto de una asignatura. Es necesario un proceso UML más simple, a ser posible también compatible con UP. En este sentido creemos que el proceso descrito por C. Larman en su excelente libro *"UML y Patrones"* [7] es muy apropiado, ya que encaja con UP y es muy sencillo, permitiendo abordar de una manera sistemática y realista la construcción de software OO. Además posee las propiedades que deben caracterizar a todo proceso UML: dirigido por casos de uso, iterativo, incremental y centrado en la arquitectura.

Creemos que el libro de Larman es el mejor texto publicado para un curso de análisis y diseño OO, ya que combina un enfoque práctico (utiliza un único caso de estudio que se sigue a lo largo de todo el libro), con un tratamiento muy riguroso de las cuestiones que van surgiendo durante todo el proceso de desarrollo en sus diferentes etapas, y además está escrito de un modo muy didáctico.

En la asignatura se sigue en gran medida este texto pero con algunas variaciones: i) comenzamos con una descripción completa de UML, en vez de introducir los modelos cuando se necesitan , ii) se introduce una etapa de modelado de negocio previa al modelado de requisitos, iii) se desdobla la etapa de *Modelado de Diseño* en dos: *Análisis* y *Diseño*, y iv) se estudian en profundidad todos los patrones GoF.

Nosotros complementamos el proceso de Larman con una etapa de *Modelado de Negocio* que precede al *Modelado de Casos de Uso y Modelado del Dominio*. El *Modelado del Negocio* es una disciplina incluida en UP que tiene como objetivo identificar y describir los procesos de negocio (casos de uso del negocio). Mostramos cómo este modelado es útil para identificar los casos de uso y vocabulario del sistema, aplicando las técnicas descritas en [4]: los procesos del negocio se describen a través de escenarios (diagramas de secuencia) y diagramas de proceso (diagramas de actividad) en los que se muestra cómo interactuan los actores (agentes que participan) y qué tareas o actividades ejecuta cada actor; finalmente, se extraen los *casos de uso del sistema*, *conceptos del dominio* y *reglas de negocio* a partir de los diagramas de proceso, aplicando unas reglas muy simples.

Por otra parte, descomponemos el *Modelado de Diseño* de Larman en dos etapas: una más cercana a los requisitos, que llamamos *Modelo de Análisis*, en la que el objetivo es obtener un diseño preliminar del sistema, determinando las colaboraciones, entre objetos de las clases del dominio, que implementan los casos de uso; y otra más cercana a la implementación, que llamamos *Modelo del Diseño*, en la que se refina el modelo del análisis introduciendo los requisitos no funcionales, y considerando cuestiones como la definición de la arquitectura del sistema, aplicación de patrones de diseño, la organización en paquetes, la persistencia o la distribución.

El estudio de los patrones GoF ocupa una buena parte de las clases teóricas del curso. Nos centramos en los aspectos de *motivación*, casos de *aplicabilidad*, *estructura* de clases, *colaboración* entre objetos participantes y *consecuencias* de aplicación, pero no se aborda la *implementación* de los patrones en un determinado lenguaje. El

objetivo es que los alumnos reconozcan situaciones en las que se puede y debe aplicar un determinado patrón, conozcan bien la estructura y comportamiento de cada patrón para su correcta aplicación, y las ventajas y problemas de su uso. Antes de comenzar con la discusión de los patrones GoF se presentan los patrones GRASP de Larman como patrones generales para el reparto de responsabilidades entre clases que siempre hay que tener en mente.

Finalmente, señalar que en el segundo ciclo de nuestro plan de estudios de Ingeniero en Informática se imparten otras dos asignaturas troncales de ingeniería del software: una centrada en ingeniería de requisitos, sobre todo en estándares y guías, y otra de gestión y planificación de proyecto informáticos. Conviene destacar que la primera de ellas se imparte antes de ADOO, pero no se ocupa del modelado de requisitos con casos de uso.

3. Organización del trabajo práctico

El objetivo principal del trabajo práctico que deben desarrollar los alumnos es la aplicación del proceso de Larman descrito en las clases teóricas, prestando también atención a la aplicación de los patrones GoF.

Las clases prácticas comienzan la tercera semana del inicio del cuatrimestre y se dedican a sesiones de ejercicios de modelado con UML de una hora y media de duración: tres de modelado de caso de uso, una de modelado conceptual, tres de modelado de colaboraciones, y una sesión para comentar la práctica del curso anterior y la documentación a entregar. Al acabar de explicar el proceso UML en clases de teoría, se dedican tres sesiones de laboratorio de hora y media cada una a explicar la herramienta de modelado que utilizarán los alumnos, en nuestro caso *Rational Rose 2001*. A mitad del cuatrimestre, entregamos a los alumnos la especificación del caso práctico para el que deben aplicar el proceso (se forman grupos de dos alumnos). Como se puede observar, se combinan prácticas presenciales junto con trabajo práctico que los alumnos realizan de acuerdo a su propia planificación. La Figura 1 muestra la ordenación temporal de las clases teóricas y prácticas (dos horas semanales de teoría y hora y media semanal para clases prácticas).

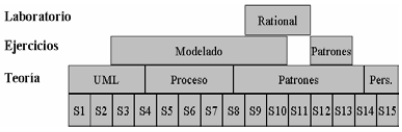

Figura 1. Ordenación temporal de los contenidos

En las sesiones de ejercicios de modelado se discute el ejemplo del terminal de punto de venta del libro de Larman y partes interesantes de trabajos prácticos de cursos anteriores, junto con otros ejercicios cuyo enunciado se les entrega con antelación. Estas sesiones finalizan ilustrando a través de un caso de uso cómo aplicar el proceso completo.

Dentro del curso, la herramienta de modelado no juega un papel central ya que nuestra principal preocupación es el aprendizaje de las técnicas relativas al modelado OO. Nosotros utilizamos *Rational Rose 2001*, pero igualmente se podría utilizar otras como *Together*, *Poseidon* o *ArgoUML*. En las tres sesiones de laboratorio con la herramienta se muestra a los alumnos cómo construir los diferentes diagramas de UML a través de ejemplos. También se presta atención a cómo organizar los modelos en paquetes y a la generación de código.

En cuanto a la aplicación del proceso por parte de los alumnos, el trabajo se organiza en las tres etapas que hemos comentado anteriormente: *Modelado de Requisitos*, *Modelado de Análisis* y *Modelado del Diseño*. La figura 2 ilustra las relaciones entre las tres etapas. Cómo se puede observar se ha excluido la etapa de *Modelado del Negocio* comentada en la anterior sección. Esta etapa fue obligatoria en el primer año, pero notamos que consumía mucho tiempo y que no era posible abordarla teniendo en cuenta la carga de créditos prácticos de la asignatura, por lo que decidimos que fuese opcional. Un modelado de negocio de interés, por muy simple que sea, requiere un esfuerzo importante, que es preferible dedicarlo a las otras etapas.

Si el caso práctico se presta a un *Modelado del Negocio* (involucra procesos de negocios con agentes que cooperan para cumplimentarlos), el alumno reconoce el interés de realizar los diagramas de procesos ya que les facilita la

identificación de los casos de uso del sistema. El caso práctico del pasado curso fue la gestión de un videoclub con máquinas automáticas de alquiler y ningún grupo optó por realizar esos diagramas de procesos, a diferencia de hace dos cursos en el que la mayoría de grupos decidieron realizar dichos diagramas, ya que el caso práctico era la gestión de pedidos en una empresa de fabricación bajo demanda.

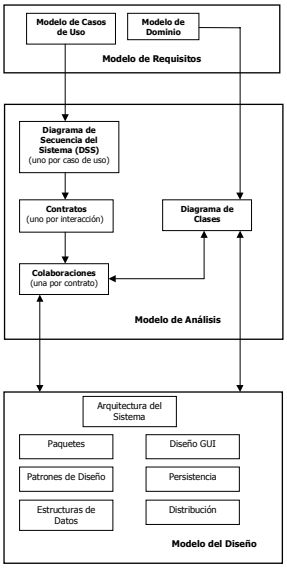

Figura 2. Etapas del proceso

En cuanto al tamaño del caso práctico, planteamos problemas en los que el alumno puede identificar de quince a veinte casos de uso, pero hay cinco o seis, que son los que corresponden a los *objetivos básicos,* en los que centra su atención; el resto son casos de uso *accesorios* (en [8] se puede encontrar esta diferenciación entre casos de uso básicos y accesorios). El problema de la gestión de un videoclub ilustra bien este tamaño de práctica.

Cada grupo debe mantener una entrevista con el profesor al final de la primera y segunda etapas, en la que se revisan los modelos realizados y se establece una lista de cambios obligatorios, cuestiones a repasar y posibles mejoras. El grupo entrega los documentos del modelado, el profesor los revisa y entonces acuerdan una cita. El alumno no puede pasar a la siguiente etapa si no es con la aprobación del profesor.

Los grupos se enfrentan primero al *Modelado de Requisitos*, debiendo realizar el *Modelo de Casos de Uso* y el *Modelo del Dominio*. Los casos de uso son especificados con una plantilla que incluye los campos: *Nombre, Objetivo, Actor Principal, Precondiciones, Flujo Principal* y *Flujos Alternativos*, siguiendo el estilo de Larman. En realidad, para esta etapa los grupos mantienen dos entrevistas con el profesor. Una para comprobar que la identificación de casos de uso es correcta y otra al final, una vez se han completado los dos modelos. El objetivo de la primera entrevista es evitar el esfuerzo inútil de escribir casos de uso que no tienen sentido.

No se tienen en cuenta los requisitos no funcionales, salvo algunos relacionados con la extensibilidad y reutilización del sistema final, que posibilitan la aplicación de patrones.

Una vez realizado el *Modelado de Requisitos*, se realiza el *Modelado del Análisis*. Durante esta etapa se especifican los *Diagramas de Secuencia del Sistema* (DSS), los *Contratos*, las *Colaboraciones* y el *Modelado de Clases*, tal y como propone Larman. Para cada caso de uso se define un DSS (diagrama de secuencia UML) que muestra la secuencia de eventos generados por el actor sobre el sistema a lo largo del caso de uso (en realidad sería un DSS para el flujo principal y otro para cada flujo alternativo).

Los contratos describen el comportamiento del sistema asociado a cada evento de un DSS (objetos y asociaciones que se crean o destruyen, cambios de estado en los objetos). Antes de pasar a realizar las colaboraciones, una para cada contrato, algunos grupos prefieren discutir con el profesor los DSS y los contratos, para tener la certeza de que han comprendido su utilidad.

Conforme desarrollan las colaboraciones, los alumnos van construyendo el diagrama de clases a partir del diagrama conceptual del *Modelo del Dominio*, incorporando los métodos a las clases, añadiendo nuevas asociaciones, y estableciendo la navegabilidad y visibilidad entre clases. Las colaboraciones siempre tienen el mismo esquema: un evento de la interfaz de usuario es recibido por

un objeto controlador, que invoca una interacción entre objetos de las clases del dominio.

Cada grupo mantiene una entrevista con el profesor para repasar las colaboraciones realizadas. En realidad sólo discutimos las cuatro o cinco más significativas que corresponden a los casos de uso básicos. Esta entrevista requiere más tiempo que la primera, ya que el diseño de las colaboraciones no resulta sencillo a los alumnos.

Los únicos aspectos de diseño que se consideran en la etapa del *Modelado del Diseño* son los patrones y la organización en paquetes (lógicamente no sería la definitiva), dejando a un lado las cuestiones relacionadas con las estructuras de datos, persistencia, distribución y diseño de interfaces de usuario. Finalmente se acaba escribiendo el código Java de las clases incluidas en el diagrama de clases final, expresando la secuencia de mensajes de las colaboraciones cómo código de los métodos.

En una última entrevista se revisa todo el modelado realizado por el grupo. Al haber mantenido, al menos, dos revisiones previas, los grupos presentan trabajos que se ajustan al diseño "ideal" previsto por el profesor, quedando sólo por discutir las cuestiones relacionadas con la aplicación de patrones de diseño y la implementación.

En las últimas semanas del curso se dedican dos sesiones prácticas a discutir ejercicios de patrones (puede encontrarse una colección de estos ejercicios en http://dis.um.es/~jmolina), que complementan el trabajo con patrones realizado para el caso práctico.

La documentación que entregan los alumnos refleja el proceso seguido: *Modelo de Casos de Uso* y *Modelo del Dominio*; *DSS*, *Contratos* y *Colaboraciones* (ordenados por caso de uso); *Diagrama de Clases* (Modelo y Controladores); descripción de los patrones de diseño aplicados y código de las clases.

En cuanto a la evaluación, se valora cada entrevista y la documentación final. En la nota final, la teoría tiene un peso del 40% y las prácticas del 60%. El examen teórico consta de un conjunto de ejercicios sobre patrones de diseño.

4. Valoración de la aplicación del proceso

En esta sección presentaremos una valoración del trabajo práctico de nuestros alumnos descrito en la sección anterior, fruto de nuestra experiencia en los últimos tres cursos en los cuales hemos revisado unas 120 prácticas de modelado con UML. Como resultado de este trabajo de revisión y de las entrevistas con los alumnos hemos detectado las principales dificultades que encuentran, los errores más comunes y los beneficios e inconvenientes de nuestro enfoque para el aprendizaje de los alumnos. Hemos organizado la discusión de esta sección de acuerdo con la etapas del proceso y al final comentamos la valoración de los alumnos.

4.1. Modelado de Casos de Uso

En la explicación del proceso se debe insistir mucho en que especificar casos de uso no es una actividad de dibujar un diagrama de casos de uso sino de escribir con precisión el flujo principal y los flujos alternativos: "centrado en la escritura en vez del dibujo", tal y como refleja el título de la principal referencia sobre casos de uso [2]. También hay que insistir en no preocuparse demasiado por las relaciones entre casos de uso (*inclusión*, *extensión* y *generalización*), ni de las relaciones entre actores, ya que ello supone una perdida de tiempo en discusiones que, normalmente, no aportan nada.

Hay que conseguir que el alumno comprenda que el objetivo inicial es identificar los actores y a partir de sus objetivos encontrar los casos de uso, siendo el diagrama de casos de uso una ayuda visual que permite ver de forma simple y directa qué actores hay y cuáles son sus objetivos.

No obstante, unos pocos grupos dedican mucho tiempo a encontrar relaciones de inclusión y extensión, llegando a establecer redes complicadas, por entender que su modelado de casos de uso tiene mayor valor si logran obtener un complicado diagrama.

A través de los ejemplos que se discuten en las sesiones de ejercicios, se consigue que los alumnos adquieran una idea de la *granularidad* o tamaño de un caso de uso. Creemos que la introducción al modelado de negocio ayuda a que adquieran esa idea, al conocer la existencia de procesos de negocio que involucran tareas (objetivos del usuario). Por otra parte, al principio muchos alumnos identifican funciones elementales como casos de uso; por ejemplo, dentro de un sistema de compra por internet,

modelan como caso de uso *"Añadir producto al carro de la compra",* en vez de definir un caso de uso *"Realizar Pedido"*.

Un error común es no encontrar los casos de uso que corresponden a actividades que lanza el sistema de forma automática; por ejemplo *"Anular reservas pasado cierto número de días"* o *"Enviar catálogo al inicio del mes"*. Otro error frecuente es englobar bajo un único caso de uso *"Gestión de X"* todas las operaciones *CRUD* sobre un objeto de negocio (alta, consulta, borrado, actualización). Les aclaramos que eso no es un caso de uso, y que lo indiquen aparte como una función del sistema, y sólo incluyan como casos de uso aquellas operaciones *CRUD* relevantes para el sistema: por ejemplo, *"Alta de un cliente en una tienda virtual"* sí debería modelarse como un caso de uso, pero no *"Alta de un producto"*.

Un problema frecuente en la identificación de actores es no establecer bien la frontera del sistema, de modo que se definen actores que realmente no interactúan con el sistema, como es el caso frecuente de clientes de un negocio que no tienen acceso a la aplicación. Los alumnos suelen caer en este error, aunque el caso de estudio del libro de Larman, el terminal de punto de venta, sirve para ilustrar bien el problema, ya que el cliente que realiza la compra no es un actor del sistema, sino que el único actor es el cajero.

También hemos conseguido que los alumnos escriban casos de uso independientes de la interfaz de usuario, al nivel *esencial* que comenta Larman. Los casos de uso que tienen en cuenta las interfaces de usuario se escribirían antes de la implementación, pero en nuestro caso no se hace ya que no llegamos a esa etapa.

Finalmente, destacamos que el libro de Larman incluye un excelente capítulo sobre casos de uso que permite a los alumnos elaborar una lista de las cuestiones básicas a tener en cuenta en el modelado de casos de uso.

4.2. Modelo del Dominio

Los alumnos no tienen problemas con el modelado del dominio ya que tienen experiencia de asignaturas anteriores con los modelos de datos *Entidad-Relación*. Les recomendamos que: i) realicen el modelado del dominio al mismo tiempo que escriben los casos de uso (conforme aparecen los conceptos se incluyen en el diagrama), ii) no se preocupen demasiado por completar las asociaciones y iii) siempre que identifiquen una clasificación, la modelen como una generalización, sin preocuparse si luego dará lugar o no a una herencia de clases.

Los alumnos no identifican todos los conceptos la primera vez; les insistimos en que incluyan los conceptos del dominio sin plantearse si darán lugar o no a clases. Otro error común es no distinguir entre la especificación de un concepto y un ítem de ese concepto; por ejemplo distinguir entre la descripción de una película y una cinta concreta de esa película. También encuentran dificultad para ver la conveniencia de modelar como un concepto, en vez de cómo atributos, informaciones tales como cantidades con unidades o ciertos identificadores (película, cuenta bancaria) que están formados de varias partes, a los que más tarde tendrá sentido asociarles alguna operación.

En general, a los alumnos les cuesta trabajo entender que no se trata de *pensar en clases* sino en el vocabulario del dominio extraído de los requisitos, a partir del cual se establecerán las clases del *Modelo* (o del *Negocio*) del sistema.

4.3. Modelo del Análisis

Los DSS resultan útiles para identificar los eventos generados por un actor hacia el sistema durante un caso de uso. Asociar a cada evento una operación a realizar por el sistema, descrita por un contrato, constituye un modo sistemático de identificar las colaboraciones que realizan un caso de uso y que finalmente darán lugar al código de la aplicación. De esta forma se rompe la falsa idea de que un caso de uso se realiza mediante una única colaboración.

La especificación de un DSS no suele presentar problemas; el alumno lee el flujo de eventos del caso de uso (el principal o uno alternativo) para encontrar cada evento lanzado al sistema por el actor.

Aunque en la segunda edición de su libro Larman comenta que los contratos sólo son de interés en aquellas situaciones en los que *"los detalles y complejidad de los cambios de estado requeridos son difíciles de capturar en los casos de uso",* nosotros obligamos a los alumnos a especificar siempre el contrato, ya que creemos

que son de gran ayuda para modelar las colaboraciones y no suponen mucho trabajo extra.

Sólo deben escribir la precondición y postcondición de la operación del sistema. Lo más importante es, sin duda, la especificación de la postcondición. En cuanto a la forma de escribir estos asertos, les aclaramos que no es necesario usar un lenguaje formal, sino que el lenguaje natural es suficiente. La mayoría prefieren el modo informal y sólo unos pocos grupos escogen OCL o la notación de asertos de Eiffel que ya conocen de la asignatura de *Introducción a la Programación OO*.

También se debe insistir en no dediquen mucho tiempo a escribir postcondiciones lo más completas posibles, sino que lo consideren una primera aproximación que les ayudará a establecer la colaboración; esto es una alternativa a comenzar directamente con el diagrama de colaboración: "*pensar antes de dibujar*".

El diseño de las colaboraciones es la tarea más compleja a la que se enfrentan los alumnos durante su trabajo práctico y a la que dedican más tiempo. Esto es lógico ya que es bien conocido que la distribución de responsabilidades entre clases y el establecimiento de las interacciones entre objetos es la parte más difícil del modelado OO [3,7], y su dominio es imprescindible para una buena programación OO.

Cabe señalar que aquellos alumnos que no han recibido un curso de *Introducción a la Programación OO*, ya que proceden de universidades en las que no es obligatorio dentro del plan de estudios o proceden de las titulaciones técnicas de nuestro Centro (hasta el nuevo plan que ha comenzado a impartirse este curso, dicha asignatura era optativa), se encuentran con grandes problemas cuando deben modelar las colaboraciones y con frecuencia abandonan la práctica (estos alumnos tienen muchas dificultades para aprobar la asignatura, necesitando más de un curso, ya que deben dedicar un esfuerzo personal para comprender los principios de la OO).

Se recomienda que primero construyan diagramas de colaboración en vez de diagramas de secuencia, ya que pueden especificar los enlaces entre objetos y conocer si es posible que un objeto pueda enviar cierto mensaje a otro objeto que es accesible, aunque finalmente también manejen diagramas de secuencia por resultarles más útiles al mostrar la secuencia ordenada de mensajes.

Los errores más comunes que cometen los alumnos al diseñar colaboraciones son: i) un objeto envía un mensaje a un objeto inaccesible, ya que no tiene visibilidad sobre él; el problema puede ser que falta una asociación entre las clases o que es necesario delegar la responsabilidad en una cadena de objetos o es necesario que el controlador del caso de uso obtenga acceso a ese objeto, ii) se viola el patrón GRASP *Experto*, no asignando una responsabilidad a la clase que tiene la información para cumplimentarla, iii) no se obtiene el objeto que corresponde a un *string* de entrada capturado a través de la interfaz, y se pasa dicho *string* como argumento a métodos que en realidad requieren el objeto, iv) se viola el patrón GRASP *Creador*, un objeto no es creado por quién debería hacerlo, v) se olvida la creación de cierto objeto, y luego se accede a él, vi) un objeto se envía sobre sí mismo un mensaje para obtener una información de la que dispone, asociada a un atributo, vii) se confunde la clase catálogo con la estructura de datos que almacena los objetos del catálogo. Sin duda, los tres primeros errores son los más importantes y los que se cometen con facilidad cuando no se tiene experiencia modelando colaboraciones o en programación OO.

Siempre procuramos que el caso práctico propuesto suponga el modelado de cinco o seis colaboraciones complejas (siete u ocho objetos) que requieran que el alumno realmente adquiera destreza en la técnica de encontrar la colaboración apropiada para un caso de uso.

Hemos observado que en muchos cursos sobre análisis y diseño OO la aplicación de un proceso se reduce a obtener casos de uso y diagramas de clases, pero no se presta atención al diseño de colaboraciones, sobre todo en cursos de ingeniería del software que abordan tanto los métodos estructurados como los procesos OO. Esto supone no realizar un verdadero modelado OO.

4.4. Modelo del Diseño e Implementación

Debido a la carga de créditos prácticos de la asignatura, sólo es posible aplicar una iteración y no se tratan cuestiones importantes del diseño, las que realmente dan quebraderos de cabeza en el desarrollo real de una aplicación: incorporar al diseño todas las restricciones impuestas por los requisitos no funcionales. Al menos sería

conveniente que cada grupo llegase a construir un prototipo de la aplicación (sería un primer prototipo evolutivo no desechable) a partir del modelado que han realizado, pero no se les exige ya que no hay tiempo para ello. Los alumnos tan sólo codifican parcialmente los métodos, siguiendo la secuencia de mensajes de las colaboraciones.

Como hemos comentado anteriormente, la práctica permite aplicar los patrones GoF más comunes en el caso de un problema real. La codificación en Java a partir de los modelos creados no les presenta ninguna dificultad y los patrones que suelen aparecer son: *Iterador*, *Composite*, *Estrategia*, *Estado*, *Factoria*, *Método Plantilla* y por supuesto los GRASP.

Con la separación entre *Modelo de Análisis* y *Modelo de Diseño* se consigue que el alumno tenga conciencia de la existencia de una visión ideal del sistema obtenida al contemplar solo los requisitos funcionales y su realización a través de un conjunto de colaboraciones, frente a una visión más detallada del sistema, previa a la implementación, obtenida mediante un refinamiento de la anterior al considerar los requisitos no funcionales.

Los alumnos valoran muy positivamente las prácticas de la asignatura y entienden que han estudiado conceptos y técnicas que les van a ser de una utilidad práctica. En cuanto al proceso, consideran que realmente les ayuda a construir software y valoran sobre todo su trazabilidad: es posible ir desde un caso de uso al código que realmente lo implementa a través de las colaboraciones. También lo consideran un método simple que ofrece un modo sistemático de llegar desde los requisitos a la implementación.

No obstante los alumnos son conscientes que han aplicado el proceso de una forma simplificada, que sirve de punto de partida para su aplicación en un caso real y para que tengan un conocimiento de las técnicas y conceptos que subyacen al análisis y diseño OO.

5. Conclusiones

Creemos que el enfoque presentado en este trabajo es adecuado para que los alumnos adquieran un buen conocimiento de los aspectos básicos que subyacen a la aplicación de un proceso UML en el desarrollo de software OO. A través de la aplicación de un proceso simple pero realista a un caso práctico no trivial, los alumnos adquieren destreza en las técnicas básicas del modelado OO con UML: modelado de casos de usos, modelado conceptual, modelado estructural y de comportamiento (colaboraciones).

El enfoque ideal sería aplicar el proceso a un problema real, llegando a la implementación a través de varias iteraciones y prestando atención al diseño. Nuestro planteamiento supone un compromiso entre el ideal y las restricciones que impone la asignatura en cuanto a carga docente. Con una carga práctica de seis créditos se podría alcanzar ese ideal.

Referencias

[1] *Computing Curricula 2001*, Final Report, December 2001, ACM e IEEE.

[2] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, 2001.

[3] E. Gamma et al., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] J. García Molina et al. *Towards Use Case and Conceptual Models Through Business Modeling*, ER2000: 19th Int. Conf. on Conceptual Modeling, USA, 2000.

[5] J. García Molina et al., *Una Propuesta para Organizar la Enseñanza de la Orientación a Objetos*, VIII Jornadas de Enseñanza Universitaria de la Informática (JENUI'02), Cáceres, 2002.

[6] I. Jacobson et al., *El Proceso Unificado de Desarrollo de Software*, Addison-Wesley, 2000.

[7] C. Larman, *UML y Patrones*, 2ª edición, Prentice-Hall, 2002.

[8] S. Lilly, *How to Avoid Use-Case Pitfalls*, 2000, http://www.sdmagazine.com

[9] RUP. Debe ser adquirido de *Rational Software*: documentación on-line, http://www.rational.com/products/rup