Ingeniería del Software: una propuesta de contenidos para titulados superiores

Juan Sánchez Díaz, M. Carmen Juan Lizandra

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{jsanchez, mcarmen}@dsic.upv.es

Resumen

En este trabajo se presenta una propuesta de contenidos para la asignatura Ingeniería de la Programación (IDP) de la Facultad de Informática (FI), de la Universidad Politécnica de Valencia (UPV). La propuesta se fundamenta, por una parte, en consideraciones relativas a las funciones que van a desarrollar los ingenieros de informática en el ámbito laboral, y por otra en la demanda existente de expertos en metodologías y herramientas relacionadas con el campo de la orientación a objetos.

1. Introducción

Con la aparición de los nuevos planes de estudio se ha producido un aumento singular de la presencia de la materia ingeniería de la programación dentro de los currícula de informática. Lo que antes era una asignatura ha pasado a desdoblarse en un grupo de asignaturas. El principal inconveniente que esto presenta es el posible solapamiento de contenidos. En particular, dentro de la titulación Ingeniero en Informática de la UPV, existen las siguientes asignaturas relacionadas con ingeniería de la programación: Metodología y Tecnología de la Programación (MTP, 4 semestre), Ingeniería de la Programación (IDP, 7 semestre), Ingeniería de Requisitos (IDR, 7 semestre) y Laboratorio de Ingeniería de la Programación (LIP, 8 semestre).

MTP es el primer contacto que tienen los estudiantes con ingeniería, pasan de la programación a pequeña escala al diseño de sistemas de información basándose en técnicas estructuradas (diseño estructurado).

En IDR, que se imparte en paralelo con IDP, tiene como objetivo principal el estudio de métodos formales de especificación de requerimientos. Se estudian también exhaustivamente los distintos paradigmas de ciclo de vida, haciendo un especial hincapié en la programación automática y en el prototipado generado a partir de especificaciones formales.

LIP, por último, se ocupa principalmente de aspectos tecnológicos: herramientas de programación y de desarrollo.

Para evitar solapamientos de contenidos, el proceso de desarrollo de software se aborda en cada asignatura desde una perspectiva diferente. MTP utiliza el ciclo de vida clásico con metodologías estructuradas. IDP emplea metodologías orientadas a objetos con el ciclo de vida en espiral, estructurado en tono a los casos de uso del sistema. Finalmente, IDR utiliza entornos de prototipación formal y programación automática.

2. Objetivos de IDP

Los objetivos de IDP están fuertemente relacionados con las funciones que se esperan de los ingenieros en informática dentro del ámbito laboral y con las demandas existentes dentro del mercado de trabajo. Estos deben poseer los suficientes conocimientos para abordar las distintas fases de un proyecto software: análisis, diseño, implementación y validación.

En la parte práctica, deben utilizar y conocer al menos una herramienta CASE que facilite la documentación y seguimiento de ese proceso de desarrollo.

IDP pretende proporcionar un bagaje de conocimientos tal que el alumno sea capaz de afrontar el desarrollo de sistemas software con una metodología adecuada, con la mayor calidad posible, en un tiempo mínimo y ayudado por las herramientas más actuales.
El objetivo principal que se persigue en IDP es hacer que el alumno conozca las metodologías orientadas a objetos más usuales y que sea capaz de utilizarlas a la hora de analizar, diseñar e implementar sistemas software.
Como objetivos parciales, cabe citar: la utilización de una herramienta CASE, que le ayude en su trabajo; el dominio de un lenguaje de programación orientado a objetos; y la utilización de un entorno de desarrollo que permite al programador crear el esqueleto básico de una aplicación, sin programar una sola línea de código y que además le facilite su trabajo con Asistentes.

3. Estructura de IDP.

IDP es troncal y se imparte en el séptimo semestre dentro de la titulación Ingeniero en Informática, con una carga docente de 4 créditos teóricos y 2 créditos de laboratorio.

4. Teoría de IDP

En la parte teórica de IDP se aborda la problemática del análisis y desarrollo de grandes sistemas, utilizando técnicas de ingeniería y desde el punto de vista de las metodologías orientadas a objetos. Haciendo un especial énfasis en las distintas fases y en la documentación que se genera en cada una de ellas.
La asignatura esta dividida en tres grandes módulos: introducción a la ingeniería del software y a la orientación a objetos (10% curso), metodologías de desarrollo (70%) y técnicas de prueba de programas (20%). Concretamente, el temario es el siguiente:
Módulo I: Introducción a la ingeniería del software y a la orientación a objetos.
Capítulo 1: Ingeniería del Software.
Capítulo 2: Lenguajes orientados a objetos.
Módulo II: Metodologías de desarrollo.
Capítulo 1: El proceso de desarrollo de OMT.
Capítulo 2: El modelo de objetos.
Capítulo 3: Los modelos dinámico y funcional.
Capítulo 4: Diseño del sistema y de objetos.
Capítulo 5: Implementación.
Capítulo 6: Comparación con otras metodologías Orientadas a Objetos: UML, Objectory.
Módulo III: Técnicas de prueba de programas.
Capítulo 1: Fundamentos de la prueba de programas.
Capítulo 2: Prueba de software basado en objetos.
Seguidamente, se comentará el contenido de cada uno de estos módulos en que se divide la asignatura.

4.1 Módulo I: Introducción a la ingeniería del software y a la orientación a objetos.

El capítulo 1, dedicado a la ingeniería del software, se estudia la evolución histórica del proceso de desarrollo de software. Presentándose distintos paradigmas de ciclo de vida. Se insiste en la necesidad de abordar este proceso desde una perspectiva científica e ingenieril.
El capítulo 2 introduce toda la terminología de la orientación a objetos, desde el punto de vista de los lenguajes de programación (objetos, clases, instanciación, herencia, tipos de polimorfismo, tipos de enlace, metaclases, etc.). Todo ello acompañado de ejemplos en los lenguajes Simula, Smalltalk, Object Pascal, C++ y Eiffel.
Con esto no se pretende que el alumno domine la programación en todos estos lenguajes, únicamente que conozca su sintaxis y las características, en cuanto a programación orientada a objetos, que cada uno de ellos posee.

4.2 Módulo II: Metodologías de desarrollo

Este módulo constituye la parte central y más extensa de la asignatura. Esta dedicado casi en su totalidad a presentar la metodología Object Modeling Technique (OMT).
OMT surgió a finales de los 80 dentro de General Electric, fuertemente influenciada por la modelización conceptual. De hecho, las primeras versiones de OMT presentaban una metodología para el diseño de bases de datos relacionales, posteriormente se le incorporaron conceptos relativos a la orientación a objetos.
El capítulo 1 presenta el proceso de desarrollo de OMT (véase la figura 1), que puede adaptarse al ciclo de vida clásico o al ciclo de vida en espiral.

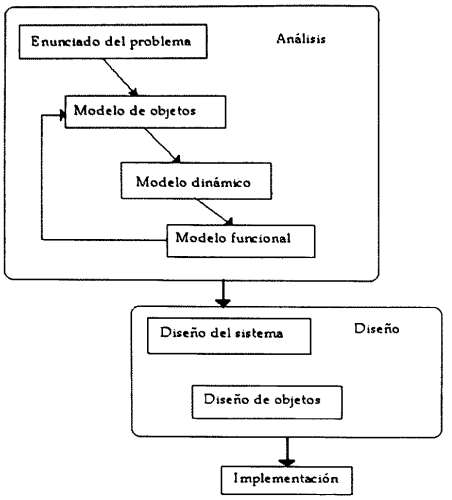

Figura 1. Proceso de desarrollo de OMT.

En la etapa de análisis se crean tres modelos que describen las tres visiones (figura 2) de un sistema: estática, dinámica y de procesamiento

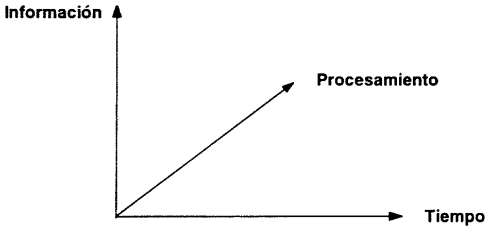

Figura 2. Visiones de un sistema.

La parte estática se describe en el modelo de objetos que contiene clases y relaciones entre las mismas (agregación, asociación, generalización). La parte dinámica, relacionada con el tiempo, utiliza diagramas de transición entre estados. Finalmente, la parte de procesamiento emplea el paradigma funcional (diagramas de flujo de datos, especificación por pseudocódigo, etc.).
El capítulo sirve también para comparar el ciclo de vida basado en técnicas estructuradas con el desarrollo orientado a objetos. Resaltando la discontinuidad de modelos que se da en las técnicas estructuradas entre la etapa de análisis y la de diseño.
También se presentan ciclos de desarrollo que direccionan directamente la reutilización de componentes, como es el caso del propuesto por la metodología BON (figura 3).

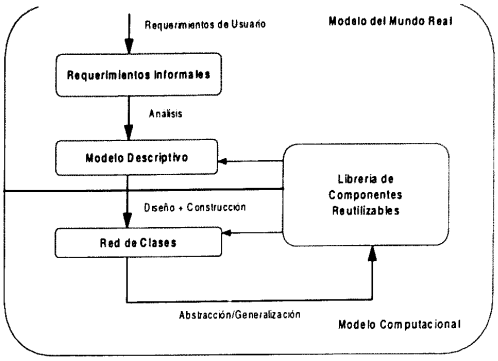

Figura 3. Proceso de desarrollo de BON.

El capítulo 2 está dedicado a presentar la notación gráfica, llamada modelo de objetos, que refleja la estructura estática del sistema. La notación es demasiado rica por lo que es aconsejable intercalar ejemplos de pequeños sistemas a la vez que esta se va presentando.
El capítulo 3 contiene los otros dos modelos que se crean en la etapa de análisis: el modelo dinámico y el modelo funcional. El modelo dinámico da cuenta de los ciclos de vida de los objetos del sistema, es decir de los estados por los que pasan desde que se crean hasta que se destruyen. El formalismo a utilizar es una variante de los diagramas de transición entre estados de D. Harel.
Para construir el modelo dinámico se siguen los siguientes pasos:

- Se identifican los actores y los **escenarios** o **casos de uso** del sistema (figura 4).

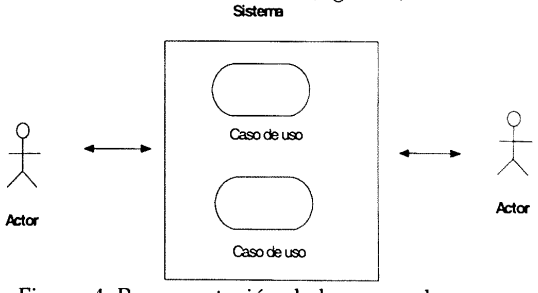

Figura 4. Representación de los casos de uso.

- Se identifica, a partir de cada escenario, los eventos entre objetos (emisor, receptor) preparándose una **traza de eventos**.
- A partir de todos los casos de uso, se construye un **diagrama de flujo de eventos**: muestra todos los eventos que pueden enviarse todos los objetos del sistema.

- Se construye, a partir de los escenarios, los **diagramas de transición** (figura 5) para clases de objetos con comportamiento dinámico relevante.

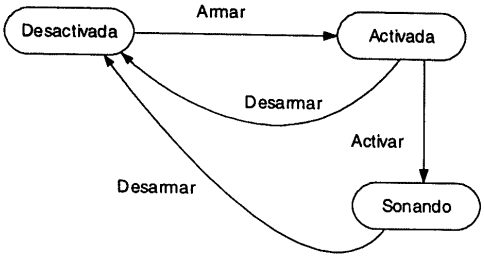

Figura 5. Diagrama de Transición.

Hay que resaltar que en este punto no se estructuran los casos de uso (con la relación extiende y usa) tal como aparece descrito en Objectory.
El capítulo concluye con la presentación del modelo funcional, que o bien puede emplear la notación de los diagramas de flujo de datos o bien cualquier otra notación como pueden ser las especificaciones pseudoformales.
El capítulo 4 está dedicado a presentar la etapa de diseño que está formada, a su vez, por dos etapas: diseño del sistema y diseño de objetos.
En la etapa de diseño del sistema se toman decisiones de alto nivel relativas a la arquitectura del mismo, identificación de la concurrencia, división en subsistemas, etc. La etapa de diseño de objetos elimina los constructores de análisis dentro del modelo de objetos utilizando "punteros". En este punto se crea una versión del modelo de objetos que es independiente del lenguaje de programación elegido para la implementación.
En el capítulo 5 se presenta la última fase, la de implementación. En esta fase se tienen que traducir las estructuras del diseño a las estructuras del lenguaje de programación a utilizar. Hay tres opciones: lenguaje de programación orientado a objetos, no orientado a objetos o que se vaya a utilizar una base de datos relacional. En este capítulo, se establece la anterior clasificación y se explican los pasos a seguir según el lenguaje a utilizar.
Finalmente, el capítulo 6 está dedicado a presentar la notación del método unificado (UML). La diversidad de modelos que este propone se comparan con los modelos que utiliza OMT. Como proceso de desarrollo, con UML, se emplea el propuesto por I. Jacobson en Objectory. Los casos de uso se estructuran con las relaciones "extiende" y "usa". Una visión del proceso de desarrollo y de la influencia de los casos de uso en cada etapa puede verse en la figura 6 .

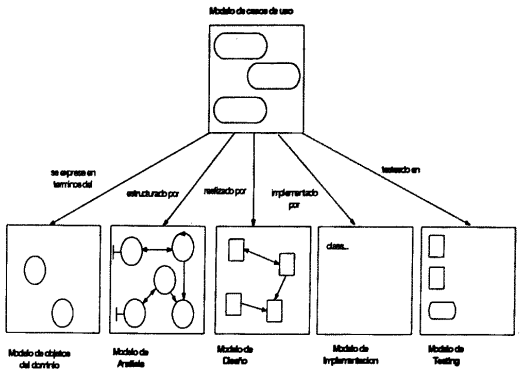

Figura 6. Objectory y casos de uso.

4.3 Módulo III: Técnicas de prueba de programas

En este último módulo está dividido en dos capítulos, el primero se presentan los distintos tipos de test que pueden realizarse en un sistema (de unidades, de integración, de aceptación). En el segundo se incide en el test de unidades, en este caso clases de un lenguaje orientado a objetos.

5. Prácticas de IDP

En el laboratorio de IDP (2 créditos) se pretende que los alumnos pongan en práctica los conocimientos teóricos aprendidos en la asignatura. Las prácticas constituyen una base muy importante en su formación académica, dado que les permiten emplear distintos tipos de herramientas y afianzar la base teórica.
Las prácticas posibilitan que el alumno programe en un lenguaje de programación orientado a objetos como es C++, utilizando un entorno de desarrollo, cada vez más difundido, como es Visual C++. Por otra parte, también se considera necesario que los alumnos manejen una herramienta CASE como es System Architect.

5.1 Herramientas

Veamos con un poco más de detalle las características de las herramientas utilizadas.
Visual C++, v. 4.2, es un entorno de desarrollo visual que contiene facilidades que permiten la construcción y la depuración de aplicaciones y librerías. Con Visual C++ se puede programar en C, en C++, realizar programas en C para Windows, usando la API que se introdujo originalmente con el SDK de Windows, y programar utilizando la librería de clases Microsoft Foundation Class Library (la Librería de Clases). La Librería de Clases esencialmente constituye una interfaz de programación de C++ para Windows, en la que se incluyen una serie de clases que permiten la creación de cualquier aplicación para Windows de una forma sencilla y cómoda, facilitando enormemente la labor del programador. Además, las MFC de Visual C++ v. 4.2. poseen características que incluyen APIs de Internet Win32 y las API ActiveX, para soportar la tecnología ActiveX.
Se permite el acceso a bases de datos, pudiéndose utilizar SQL/DAO/ODBC. Además, Visual C++ incluye una serie de asistentes que hacen que la creación de aplicaciones o el añadir clases, métodos o atributos sea una labor sencilla. Por ejemplo, utilizando el AppWizard se puede crear el esqueleto de una aplicación sin implementar una sola línea de código, código que si hubiera que programar no sería una tarea inmediata. Así pues, se puede decir que Visual C++ es una herramienta que facilita y aumenta la productividad del programador.
Además de poder utilizar la Biblioteca de Clases, en Visual C++ se pueden añadir una serie de librerías estándar así como librerías creadas por el usuario, permitiendo la utilización de nuevas clases, sin necesidad de que el programador las implemente. Por ejemplo, la librería OpenGL, que en esencia es una librería de gráficos 3D, permite crear visualizaciones en 3D. LEADTOOLS, que es una librería que permite incorporar compresión de imagen, soporte a distintos formatos de ficheros, y procesamiento de imagen (filtros, rotaciones, etc.). Esta librería facilita el tratamiento de ficheros que contienen imágenes, permite recuperar o almacenar cualquier formato de fichero y su posterior visualización en pantalla.
System Architect (SA), es una herramienta CASE que permite la creación de modelos gráficos ya sea con metodologías estructuradas (Yourdon,, DeMarco, etc) o con metodologías orientadas a objetos. (Booch 94, Coad/Yourdon, OMT, UML). Permite importar y exportar diagramas e intercambiar información con Microsoft Repositoy de Microsoft. Asimismo, genera e importa código de lenguajes como C++, Smalltalk, Java, CORBA IDL, Visual Basic y Delphi Object Pascal. Por último, posee un modulo que permite crear modelos relacionales a partir de modelos entidad-relación.

5.2 Programación

Las prácticas, que se realizan en grupos de dos personas, están divididas en 6 sesiones de 2 horas de duración cada una. Se organizan del siguiente modo:
SESIÓN 1: Introducción a Visual C++. Ejemplo de aplicaciones creadas utilizando Visual C++ y que demuestran la potencia del mismo. Primera aplicación utilizando AppWizard. Incorporación de algunas primitivas de dibujo, cajas de diálogo, opciones de menú y botones.
SESIÓN 2: Creación de clases base y derivadas para dibujar figuras en pantalla: cuadrados y círculos. En primer lugar, las figuras se dibujarán en una posición fija en la pantalla. Al seleccionar una entrada de menú o un botón se dibujarán las figuras en la pantalla. Después, se modificará el código para que las figuras aparezcan en la pantalla al pulsar el botón izquierdo del ratón. Por último, se creará una lista en la que se incluirán todos los elementos que se vayan dibujando. En este punto se utiliza una clase genérica.
SESIÓN 3: Permitir al usuario seleccionar el color y el radio de las figuras. Hacer que se puedan almacenar y recuperar todas las figuras que aparecen en un documento.
SESIÓN 4: Introducción y primera toma de contacto con System Architect. En esta sesión, el alumno da un paseo por SA para conocer todas sus posibilidades y crea la primera enciclopedia con distintos tipos de diagramas, utilizando los modelos de OMT.
SESIÓN 5: Dedicada a la resolución de ejercicios propuestos de modelización de sistemas de información.
SESIÓN 6: Presentación y realización de la práctica final. En la misma, el alumno debe realizar el análisis, diseño e implementación de un sistema de información propuesto, documentando las distintas fases con OMT.
Para las sesiones de Visual C++ se dispone de material que permite al alumno crear las aplicaciones paso a paso. En cuanto a las sesiones con System Architect, el alumno también dis-

pone de material de apoyo para la realización de dichas prácticas.

Conclusiones.

Consideramos que el temario propuesto se adapta perfectamente a lo que se espera de un ingeniero en informática a finales de los años 90: un conocimiento profundo de alguna metodología orientada a objetos (OMT), y una visión de las nuevas tendencias en el desarrollo OO (UML + Rational Objectory Procces). En la parte práctica de la asignatura, se muestra un entorno actual de desarrollo como es Visual C++ y una herramienta CASE que soporta la tecnología orientada a objetos.

Referencias

BLOQUE 1:

Budd, T., Introducción a la programación orientada a objetos, Addison-Wesley Iberoamericana, 1994

Graham, I., Object oriented methods, Addison-Wesley, 1991

Masini, G., Object Oriented Languajes, Academic Press 1991.

Meyer, B., Object Oriented Software Construction, Prentice Hall, 1991

BLOQUE 2:

Blaha, M., Object-Oriented Modeling and Design for Database Applications, Prentice Hall 1997

Booch, G., Análisis y diseño orientado a objetos, Addison Wesley Iberoamericana, 1995

Eriksson, H.E., Penker, M., UML Toolkit, John Wiley & Sons, 1997

Fowler, M., UML Distilled. Addison-Wesley 1997.

Jacobson, I., Object-oriented software engineering: A use case driven approach, Addison Wesley, 1995

Martin, J., Análisis y diseño orientado a objetos, Prentice Hall, 1994

Muller, P., Instant UML, Ed. Wrox Press Inc, 1997

Rumbaugh, J., Modelado y diseño orientado a objetos, Prentice Hall, 1996

BLOQUE 3:

Kit, E., Software testing in the real world, Addison Wesley, 1995

Marick, B., The craft of software testing, Prentice Hall, 1995

PRÁCTICAS

Ceballos Sierra, F. J., Visual C++ : aplicaciones para Windows, Ra-ma, 1995

Devis Botella, R., Programación orientada a objetos en C++, Paraninfo, 1993

Hernández Orallo, E., Hernández Orallo, J., Programación en C++, Paraninfo, 2ª Edición, 1995

Kruglinski, D. J., Progrese con Visual C++, McGraw-Hill, 1993