Metodología y Tecnología de la Programación: una propuesta para mejorar el aprendizaje del diseño de sistemas de información

Sánchez P., Letelier P., Molina A., Carsí J.A.
Escuela Universitaria de Informática[1]
Universidad Politécnica de Valencia
e-mail: {ppalma, letelier, amolina, pcarsi}@dsic.upv.es

Resumen

La enseñanza de las fases del ciclo de vida de construcción de software sigue un orden inverso respecto de su aplicación en el plano profesional. Es aceptado el hecho de comenzar enseñando programación y continuar, en cursos posteriores, con actividades de diseño y análisis. Esta manera de formar a los futuros ingenieros de software puede ser una razón importante para explicar que posteriormente en la industria el uso de métodos de análisis y diseño en el desarrollo de sistemas muchas veces sea una excepción. MTP (asignatura Metodología y Tecnología de la Programación) es el primer contacto que el alumno tiene con técnicas de diseño de software tras una formación en programación. El alumno debe afrontar el diseño de sistemas de tamaño medio. Observamos dos aspectos pedagógicos cruciales en MTP: la motivación del alumno y el cambio contextual que le significa pasar de codificar a diseñar sistemas. Este trabajo presenta algunas recomendaciones docentes para abordar el diseño estructurado a partir de la experiencia contrastada de los últimos cursos académicos.

1. Introducción

La manera tradicional de realizar la docencia en ingeniería del software es proporcionar los contenidos teóricos y prácticos al alumno en un orden inverso al que deberían usarse en la resolución de situaciones reales. Así, el primer contacto del alumno con la informática son los algoritmos y estructuras de datos básicas construyendo pequeños programas en un determinado lenguaje. Después, la formación se orienta hacia técnicas y métodos para el diseño de programas y de datos. Por último, el alumno se introduce en el análisis de sistemas. Aunque este enfoque docente parece ser el más recomendado, esta manera de formar a los futuros ingenieros de software puede explicar el hecho de que posteriormente en la industria la aplicación de fases de análisis y diseño en el desarrollo de sistemas muchas veces sea una excepción. La formación inicial como programadores determina, en gran medida, la motivación posterior del alumno. Además, la funcionalidad de los actuales entornos de programación hace que el interés se concentre en programar para obtener cuanto antes el producto funcionando, relegando o descartando cualquier análisis y diseño serio del sistema. Suele suceder que las técnicas de análisis y diseño son utilizadas para documentar sistemas cuando ya han sido construidos.

Los enfoques de diseño basan su éxito en un buen particionamiento del problema como una forma efectiva de enfrentar su complejidad. Todo sistema de información tiene dos perspectivas: dinámica (asociada al procesamiento de la información) y estática (asociada a la estructura de la información). En diseño estructurado el particionamiento de la dinámica se realiza en módulos que además se organizan jerárquicamente, el particionamiento de la estructura se aborda con técnicas de modelado de bases de datos (Diagramas E-R y modelo relacional). En diseño orientado a objetos el particionamiento es a través de clases y objetos, incluyendo los aspectos dinámicos y estáticos a la vez.

En la *Escuela Universitaria de Informática (EUI)* de la *Universidad Politécnica de Valencia (UPV)* figura en sus planes de estudios la asignatura *Metodología y Tecnología de la Programación (MTP)* como troncal de 3 créditos teóricos y 3 créditos prácticos en el cuarto semestre y para ambas titulaciones de *Ingeniero Técnico en Informática* de *Gestión* y de *Sistemas*.

MTP es el primer contacto que el alumno tiene con métodos de diseño de software tras una formación en programación. El alumno se enfrenta a la

[1] Este trabajo ha sido parcialmente financiado por la Escuela Universitaria de Informática de la Universidad Politécnica de Valencia.

problemática de definir arquitecturas de programas cuyo tamaño supera a los previamente estudiados. La perspectiva *metodológica* de la asignatura se centra en el estudio del diseño de programas (utilizando el Diseño Estructurado). La parte *tecnológica* considera el uso de una herramienta CASE y, por último, la parte de *programación* aborda la implementación modular del software en un lenguaje imperativo.

Hemos constatado dos puntos críticos en la docencia de MTP: (1) la **motivación** del alumno, y (2) el **cambio de contexto** que significa pasar de escribir líneas de código a diseñar sistemas. La presentación de ejemplos de diseño ha demostrado no ser suficiente para que el alumno, por sí mismo, pueda abordar el diseño de un sistema.

En este curso 97/98, a diferencia de los anteriores, hemos experimentado usando una herramienta CASE en el laboratorio de prácticas dedicando más esfuerzo al diseño y menos a la programación correspondiente. Una encuesta realizada a 700 alumnos revela la necesidad de que esta fase de diseño se realice de forma "activa". En este trabajo, y a partir de una revisión de los inconvenientes y desafíos para el éxito de la asignatura, se plantean recomendaciones docentes para abordar la enseñanza del diseño de programas y para asegurar los dos puntos críticos señalados anteriormente.

2. El diseño en los entornos de programación actuales

Teóricamente, la programación a partir del diseño debería ser una tarea sencilla e incluso casi totalmente automatizable. Sin embargo, la realidad nos muestra lo contrario. Usando programación orientada a objetos difícilmente se consigue una implementación fiel al diseño por algunas de las siguientes razones: (1) se usa una base de datos relacional para la persistencia de los objetos, (2) el modelado de la interfaz del sistema normalmente no es parte del diseño del sistema, y (3) no se realiza una programación orientada a objeto pura.

Usando programación estructurada, con lenguajes de tercera o cuarta generación, el diseño estructurado ofrece una transición directa: cada módulo es un procedimiento (o el nombre que esto reciba en el lenguaje); cada especificación del módulo es el cuerpo del procedimiento asociado (en sintaxis del lenguaje, incluyendo las invocaciones entre módulos como llamadas a procedimientos). Al trabajar con un lenguaje de cuarta generación (4GL) el código asociado al acceso a bases de datos se reduce, disminuyendo el tamaño de los módulos e incluso disminuyendo el número de ellos (en comparación a una misma solución sin usar un lenguaje de cuarta generación). Respecto de la programación de la interfaz, un inconveniente al usar diseño estructurado es el gran abanico de salida obtenido frecuentemente en módulos asociados a formularios. Sin embargo, esto no presenta problemas al programar pues los procedimientos de eventos del formulario se definen fácilmente en el entorno.

Por lo anterior, cuando el entorno de desarrollo no es orientado a objeto, indudablemente el diseño estructurado es lo más adecuado. Incluso usando un entorno de programación orientado a objetos, si la implementación no es fiel al diseño entonces toda la parte de la aplicación que no es orientada a objetos (incluyendo la interfaz) puede ser diseñada siguiendo el enfoque estructurado.

3. Introducción a la asignatura MTP

3.1. Contenidos

El contenido teórico de la asignatura viene desglosado en los siguientes puntos:

1. Ingeniería del Software: paradigmas del ciclo de vida de desarrollo de software, análisis, diseño y prueba de sistemas, factores de calidad, etc.
2. El Diagrama de Flujo de Datos (DFD) como herramienta gráfica del análisis: interpretación y enlace con el Diccionario de Datos (DD).
3. El Diagrama de Estructura (DE) como herramienta gráfica del diseño estructurado.
4. Conceptos fundamentales del diseño: modularidad, cohesión, acoplamiento, etc.
5. Derivación del DE a partir del DFD. Relación con los conceptos de diseño.
6. Diseño de casos de Prueba del software.

El 70% de la docencia se centra en los puntos 3, 4 y 5. El material bibliográfico fundamental del que disponen los alumnos para la asignatura está formado por los libros [1][2]. Como bibliografía complementaria se les recomienda [4][5].

3.2. Evolución de las prácticas en los últimos cursos

El tiempo total destinado a las prácticas es de 30 horas distribuidas en 15 semanas de 2 horas en cada sesión. Veremos cómo han evolucionado los siguientes items en cuatro cursos académicos:

1. El control de asistencia a prácticas.
2. El porcentaje dedicado a programación frente a diseño.
3. La evolución del porcentaje de asistencia a lo largo del curso.
4. El método de evaluación de los contenidos de dichas prácticas.

Cursos 1994/95 y 1995/96
La asistencia fue obligatoria y la evaluación consistió en el desarrollo de un programa en el lenguaje C a partir de un DE que el alumno obtenía desde el DFD proporcionado como ejemplo.

La Figura 1 muestra la distribución de los contenidos de las prácticas.

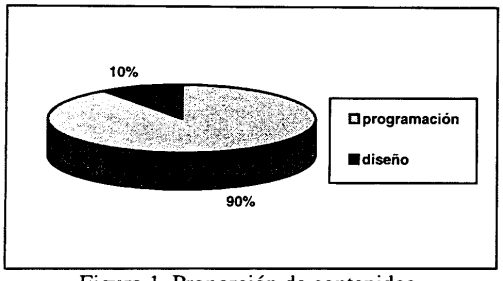

Figura 1. Proporción de contenidos.

La Figura 2 muestra la evolución de la asistencia a las prácticas a lo largo del curso.

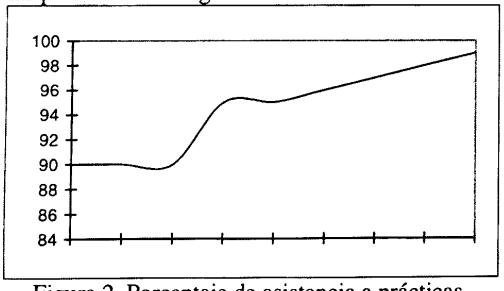

Figura 2. Porcentaje de asistencia a prácticas.

Comentarios
El alumno asiste por ser obligatoria la práctica. Se distinguen claramente dos niveles de aprendizaje: (1) quienes asimilan muy bien el entorno de programación con máximo provecho, y (2), quienes se limitan a cumplir las exigencias mínimas. Al final la asistencia aumenta pues coinciden con la entrega de los trabajos de prácticas. El alumno busca entregar un trabajo "como sea" y acude a otro horario distinto del asignado.

Curso 1996/97
La asistencia fue voluntaria pero la evaluación de las prácticas estaba incluida en el examen final de la asignatura con una parte relativa al lenguaje C. La práctica consistió en modificaciones y programación para un caso de diseño dado.

La Figura 3 muestra la distribución de los contenidos de las prácticas.

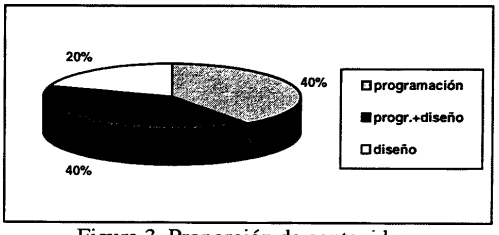

Figura 3. Proporción de contenidos.

La Figura 4 muestra cómo fue la evolución de la asistencia a las prácticas.

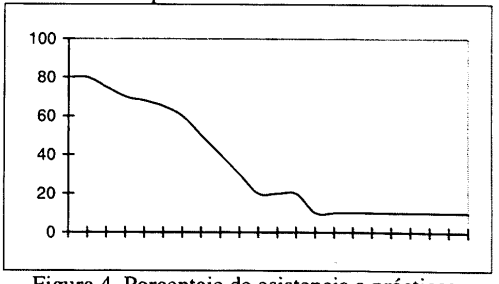

Figura 4. Porcentaje de asistencia a prácticas.

Comentarios
El alumno asiste al principio por ver cómo se desarrollan las prácticas pero limita el esfuerzo al mínimo. Esto hace que destine su tiempo a otras asignaturas en las que la asistencia es obligatoria y se exige entrega de trabajos. Al final del curso, quedan únicamente aquellos alumnos con una motivación propia.

Curso 1997/98
No se pasa lista de asistencia, pero se realizaron dos evaluaciones de prácticas en el laboratorio durante el curso, comunicadas al comienzo del

mismo. Dichos controles proporcionaron el aprobado o no en cuanto a las prácticas para cada alumno. Se reduce el tiempo dedicado a relacionar diseño con programación. Se llevaron a cabo dos tipos de diseño: (1) un diseño asistido en el que el alumno recibe un sistema ya construido y se limita a introducirlo[2] en la herramienta CASE, y (2) un diseño que el alumno de forma *activa* e *incremental* va desarrollando por sí mismo con la ayuda del profesor. Los dos controles fueron relativos a la segunda parte de diseño.

La Figura 5 muestra la distribución de los contenidos de las prácticas.

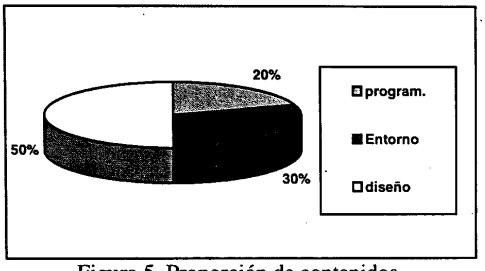

Figura 5. Proporción de contenidos.

La Figura 6 muestra la evolución de la asistencia a las prácticas.

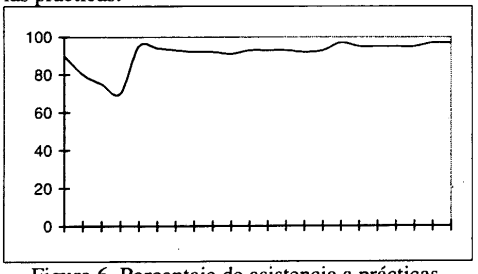

Figura 6. Porcentaje de asistencia a prácticas.

Comentarios
La asistencia disminuyó cuando se dedicó el tiempo a la programación y aumentó hasta el máximo cuando se comienza a hacer uso de la herramienta CASE. Aunque la obligatoriedad implícita a los dos puntos de evaluación en el último mes condicionó la asistencia, no era de esperar que se mantuviese en un nivel tal alto durante todo el curso. Esto nos permite afirmar que se ha logrado una mayor motivación del alumno, lo cual se confirma con los resultados de la encuesta que se detalla más adelante. Dos innovaciones importantes introducidas este último año pueden explicar lo anterior: 1) un enfoque de enseñanza práctica mediante el uso de patrones de diseño, 2) la utilización de una herramienta CASE.

[2] Esto permite al alumno el aprendizaje del entorno CASE.

4. Uso de patrones de diseño

El alumno normalmente comienza su aprendizaje de programación estudiando pequeños ejemplos. Posteriormente la resolución por sí mismo de nuevos programas se basa en gran medida en su experiencia y en la aplicación de casos típicos de programación ya aprendidos. En el diseño el aprendizaje mediante ejemplos no parece tener la misma efectividad, pues posteriormente el alumno no consigue fácilmente abordar nuevos diseños. Hemos observado un mayor éxito cuando se consideran ciertos ***patrones de diseño***. Estos patrones cubren situaciones típicas del diseño además de tener un enlace y justificación con los contenidos teóricos.

Con todo esto, el modelo de prácticas que consideramos adecuado para el diseño de sistemas de información posee las siguientes propiedades:

- Se dispone de patrones de diseño. Se formulan ejemplos que resalten la utilización de cada uno de los patrones y que muestran cómo resolver dichos ejemplos de manera sencilla.
- Se proponen ejemplos de complejidad cada vez mayor para ir combinando los distintos patrones de diseño.

Nuestro objetivo final es que el alumno sea capaz de detectar qué patrones de diseño necesita y cómo aplicarlos. Este aprendizaje debe hacerse extensible para sistemas de complejidad mayor. Consideramos que esto es posible dado que la descomposición jerárquica del sistema es intrínseca a la metodología estructurada.

5. Algunos patrones de diseño

Para cada concepto del diseño estructurado que consideremos fundamental se proporciona un patrón de diseño. A continuación se presentan algunos de los patrones utilizados.

Bifurcación de la entrada
Frente a un dato de entrada (que puede ser estructurado) se hacen, por ejemplo, validaciones individuales para producir un dato valido. La

Figura 7 es representativa del patrón de diseño correspondiente en un DFD.

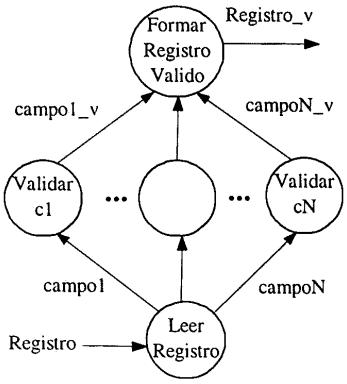

Figura 7. Patrón bifurcación a la entrada.

La Figura 8 muestra el DE solución para dicho patrón.

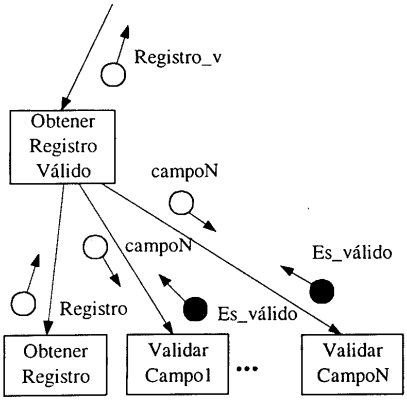

Figura 8. DE para bifurcación a la entrada.

Edición por niveles

El patrón correspondiente fija, en términos generales, que los aspectos sintácticos se validen antes que los semánticos[3]. El patrón obliga a reestructurar el orden de las acciones indicadas en el DFD para preservar dicho aspecto.

Para este caso aportamos en la Figura 9 un DE representativo de los aspectos de edición para el nivel sintáctico/semántico. El alumno debe distinguir (de entre los procesos del DFD) el tipo de validación que se hace: sintáctica o semántica, y representarlas en distintos niveles.

[3] Este patrón se extiende considerando otros aspectos tales como: validar lo simple antes que lo cruzado y lo interno antes que lo externo [4].

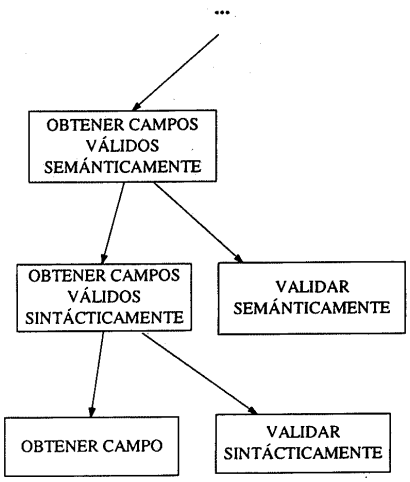

Figura 9. Aspectos de edición.

Interfaz E/S

La comunicación con los almacenes de datos debe hacerse siguiendo el patrón de la Figura 10. El acceso a las entidades externas es similar aunque por lo general debe sujetarse al interfaz específico de la entidad con el sistema. Tradicionalmente los alumnos no encuentran la forma de ubicar las llamadas a sentencias SQL cuando, por ejemplo, programan en 4GL una sentencia de *insert* o *select*. Eligiendo un interfaz apropiado se pueden describir dichos aspectos.

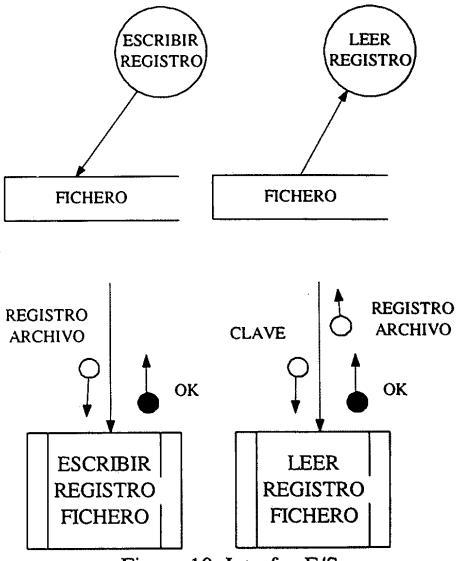

Figura 10. Interfaz E/S.

Accesos repetidos a ficheros
La inclusión de cúmulos de información (clusters) permite mejorar el rendimiento por accesos repetidos a archivos. Además disminuye el acoplamiento de los módulos que hacen uso del cluster y puede eliminar flujos de datos vagabundos o evitar el uso de variables globales. El patrón ejemplo de la Figura 11 permite disminuir el acoplamiento por estampado y mejorar los accesos repetidos (para el "registro empleado") y debe aplicarse cuando se detecte tal situación.

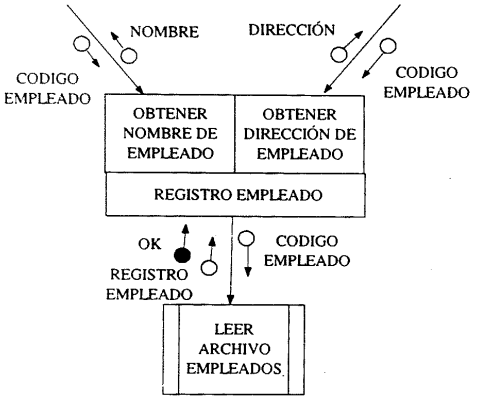

Figura 11. Consideración de un cluster.

Otros aspectos
Por simplicidad, se han incluido sólo algunos de los patrones que hemos usado en la asignatura.

Otros patrones importantes aplicables a la hora de derivar el DE desde el DFD pueden ser: dar formato a la información de salida, realizar las validaciones e informar debidamente de los errores detectados, aspectos relativos a la inicialización y terminación de la ejecución de actividades del sistema, estructura general del DE detectando ramas aferentes, de transformación y eferentes, etc.

6. Encuesta realizada

Se encuestó al alumnado para conocer su opinión acerca de la asignatura en general y de las prácticas realizadas con la herramienta CASE. La encuesta fue respondida por 640 alumnos.

A continuación se comentan los resultados obtenidos en dicha encuesta.

Figura 12. Resultados pregunta 1.

El gráfico de la Figura 12 ilustra que la dificultad principal para el alumno es conseguir derivar los DE a partir de la especificación del análisis que se le proporciona así como aplicar los criterios de diseño impartidos en las clases teóricas.

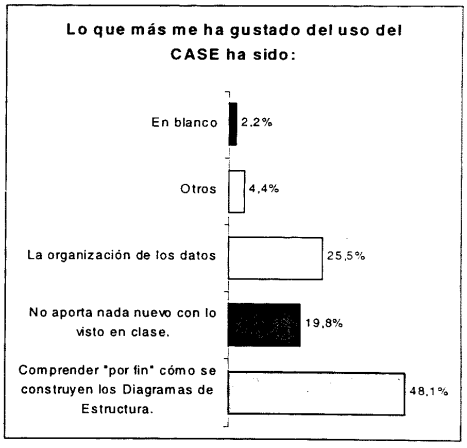

Figura 13. Resultados pregunta 2.

La Figura 13 señala que, a pesar de las dificultades enunciadas en la pregunta 1, para un porcentaje elevado de alumnos (48%), el uso del CASE y una participación activa en las sesiones prácticas les ha ayudado en la comprensión de la construcción de Diagramas de Estructura.

La Figura 14 ilustra el hecho de que la mayoría de alumnos (70%) se han sentido especialmente atraídos por la utilización del CASE. El uso de una herramienta de clara utilidad práctica consigue motivar al alumno, lo que también repercute en

una mayor asistencia a las clases prácticas, como se ha detallado antes.

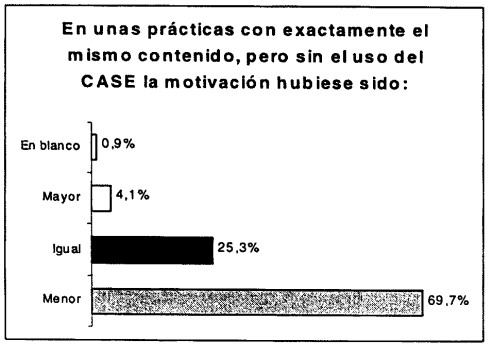

Figura 14. Resultados pregunta 3.

7. Conclusiones

La enseñanza exitosa del diseño de sistemas de información debe afrontar varios inconvenientes. Tras aprender técnicas de programación en un lenguaje dado, el alumno tiende a quedar condicionado sólo a esa visión para construir software. Esto se refuerza con las facilidades de los actuales entornos de programación visual, que suelen incentivar la construcción de sistemas comenzando directamente por su programación. Por otra parte, el cambio de contexto entre escribir líneas de código y diseñar la arquitectura del sistema parece ser un gran obstáculo para el alumno. Todo ello repercute negativamente en la motivación del alumno.

Se han presentado los resultados obtenidos en cuatro años de docencia en MTP, la primera asignatura de las del grupo de ingeniería del software de la Escuela de Informática en la Universidad Politécnica de Valencia. En un afán por conseguir un aprendizaje efectivo con un adecuado nivel de motivación, se han aplicado distintas estrategias para llevar a cabo la parte práctica de la asignatura. Este último año académico el resultado ha sido satisfactorio y nos permite proponer el enfoque utilizado para que sirva de antecedente en asignaturas similares. La propuesta se basa en tres aspectos: 1) asistencia opcional pero con un par de evaluaciones preestablecidas, realizadas en el laboratorio de prácticas, 2) diseño incremental, con la ayuda del profesor, usando patrones de diseño y 3) utilización de una herramienta CASE. Los resultados han sido confirmados por una encuesta realizada a 700 alumnos.

8. Bibliografía

[1] Molina A., Letelier P., Sánchez P., Sánchez J., *Metodología y Tecnología de la Programación*, Servicio de Publicaciones UPV-97.498, ISBN 84-7721-519-7, 1997.

[2] Sánchez P., Molina A., Sánchez J., Letelier P., Vivancos E., *Ejercicios Solucionados de Metodología y Tecnología de la Programación*, Servicio de Publicaciones UPV-97.960, 1997.

[3] Ministerio para las Administraciones Públicas (MAP), *Métrica v.2.1. Metodología de Planificación y Desarrollo de Sistemas de Información*, 1995.

[4] Page Jones M., *The Practical Guide to Structured Systems Design*, Prentice Hall, 1988.

[5] Pressman R.S., *Ingeniería del Software: Un Enfoque Práctico*, Tercera edición, Mc Graw Hill, 1994.