METODOLOGÍA DE LA PROGRAMACIÓN: ESTUDIO DE LA FASE DE PRUEBAS EN EL CICLO DE VIDA DEL SOFTWARE

Manuel Velasco de Diego
velasco@ia.uc3m.es
Departamento de Informática
Universidad Carlos III de Madrid
Butarque 15, 28911 Leganés

La asignatura "Metodología de la Programación" es una asignatura cuatrimestral obligatoria de segundo curso en Ingeniería Técnica en Informática de Gestión. Su contenido se centra principalmente en el estudio de la fase de pruebas en el ciclo de vida software. Esta fase no recibe la importancia y el estudio que se merece pese a que se ha estimado que ocupa aproximadamente el 50% del esfuerzo total de todo el proyecto software. En la mayoría de los casos, la falta de formación del personal que debe llevarla a cabo impide su correcto desarrollo. Esta asignatura trata de paliar este problema entre los titulados técnicos, introduciéndoles una serie de conceptos, en su mayoría básicos, relacionados con el entorno de desarrollo de la fase de pruebas. A la vez, se ha intentado ir más allá, profundizando en la mejora y definición de un conjunto de técnicas de prueba existentes.

1. Objetivos de la asignatura

La asignatura "Metodología de la Programación" es una asignatura cuatrimestral obligatoria de 5 créditos que se imparte en segundo curso en Ingeniería Técnica en Informática de Gestión. Su contenido se centra principalmente en el estudio de la fase de pruebas del ciclo de vida software. Para el estudio del resto de las fases del ciclo de vida software existen en esta titulación otras asignaturas relacionadas como son: Lógica de la Programación, Ingeniería de Software I y II, Diseño Avanzado de Software y Metodología de Desarrollo de Software.

El temario de la asignatura no se circunscribe únicamente a la fase de pruebas pero se ha considerado ésta de mayor interés, por su novedad, para la presentación en estas jornadas.

La fase de pruebas [1], [3], [4] no ha recibido la importancia que se merece ni desde el lado académico ni desde la empresa. Normalmente la fase de pruebas se limita a la instalación de la aplicación y a un mínimo periodo de vigilancia de su funcionamiento, pero sin buscar las condiciones extremas de trabajo que son las que verdaderamente encierran los problemas. Todo suele funcionar en condiciones óptimas, mientras que la mayoría de errores sólo aparecen en los límites de funcionamiento, límites en los valores de entrada/salida, límites en los valores de repetición de una iteración, etc. De esta forma, la mayoría de aplicaciones funcionan cuando son instaladas porque no son sometidas normalmente a este tipo de condiciones adversas, pero no por ello deben ser olvidadas.

Los programadores y analistas que desempeñan su trabajo en las empresas provienen en gran parte de la universidad y no han sido educados como probadores del código que generan. No han sido educados tampoco desde el punto de vista psicológico. Para ellos la actividad de prueba siempre ha sido vista como una actividad destructiva [1] y no creativa. El hecho de probar un programa quedaba para el profesor que les examinaba. Normalmente los alumnos están mal acostumbrados a terminar sus prácticas o trabajos el último día, dejando el mínimo tiempo posible para realizar la prueba de sus programas, o en el caso de que sí se dedique un esfuerzo significativo, realizando la prueba de forma ineficiente. Cualquier profesor que corrija una práctica sabe que sólo es cuestión de tiempo el encontrar errores a la aplicación, siendo en gran parte de los casos bastante sencillo, incluyendo casos extremos de código que no compila o ejecutables que, sin dar ninguna opción, dejan "colgada" la máquina.

A partir de todo esto se adivina la dificultad ante la que cualquier grupo de trabajo que desarrolle un proyecto se encuentra al realizar las pruebas que deben validar correctamente el producto obtenido. No existe personal especializado capaz de realizar un plan de pruebas y sólo en algunos casos existe un leve conocimiento de alguna de las técnicas a emplear, técnicas que como se recordará a continuación sólo son válidas para un conjunto determinado de casos.

Por ello y para intentar hacer cambiar la mentalidad del alumnado en el comienzo de su vida en el mundo de la informática se han planteado los siguientes objetivos con el estudio de esta materia:

a) Concienciar al alumno de la importancia que tiene la fase de pruebas intentando que no sea vista como una tarea destructiva sino como la culminación del trabajo bien realizado. De esta forma se conseguirá obtener un grupo de programadores responsables. Perdiendo el miedo a realizar las pruebas de sus propios programas se completará la labor del programador. Todo programador debe ser el primero que pruebe su código.

b) Conseguir que tengan un conocimiento extenso de las distintas técnicas de prueba, conociendo los distintos fundamentos matemáticos que las sustentan.

c) Desarrollar en profundidad las técnicas existentes. En un gran número de casos las técnicas propuestas sólo funciona en unas condiciones muy determinadas y su funcionamiento es muy limitado. Pueden obtenerse fácilmente mejoras para ampliar el espectro de trabajo de cada técnica y añadirles funcionalidad. De esta forma, ante un aspecto inesperado, siempre podrá intentarse una solución mediante una modificación inteligente de la técnica.

d) Intentar definir perfectamente el ámbito de aplicación de cada técnica para que, ante un problema dado, el alumno sea capaz de seleccionar correctamente la técnica más apropiada, estableciendo unas reglas generales de selección. Es importante no solamente saber qué hay que hacer sino además en qué casos.

2. Contenidos

El contenido del curso se divide principalmente en cuatro bloques [2], que se presentan a continuación.

En el primero ("Introducción a la prueba de programas") se presentan las primeras nociones de pruebas, intentando hacer constar la importancia de éstas mediante una selección de ejemplos sobre fallos notorios y famosos en el software, la mayoría debidos a la incorrecta realización de las pruebas. Se establece una división de las pruebas en función de los elementos necesarios para su realización: requisitos, código fuente y código ejecutable. Es importante que el alumno perciba claramente cuáles son los elementos de trabajo de los que va a disponer, para poder seleccionar en cada caso la técnica correspondiente. Una clasificación [1], atendiendo a los elementos de trabajo, de los distintos tipos de pruebas se presenta en la figura 1.

	Objetos necesarios			Ejemplos de Técnicas
Clases	Especificaciones	Código Fuente	Código Objeto	
1				Ciertas pruebas aleatorias.
2				Análisis estático del flujo de datos, Revisiones de código, Cálculo de complejidad
3				Análisis dinámico del flujo de datos.
4				Análisis de coherencia de especificaciones formales.
5				Pruebas en los límites, Grafos causa-efecto, Análisis particional Análisis sintáctico, ...
6				Ejecución simbólica, Prueba formal
7				Métodos de cobertura, Pruebas mutacionales

Figura 1. Clasificación de los distintos tipos de pruebas

En el segundo bloque ("Técnicas de prueba funcional") se trabaja con el primer grupo de técnicas, las funcionales, que no necesitan el código fuente y cuya comprensión por parte del alumno es mayor. Su componente matemática es menor que en las pruebas estructurales y siempre es más sencillo el análisis de los requisitos antes que la observación del código, en muchos casos ilegible. Se trabaja con todas las técnicas: Análisis particional, Prueba de los límites, Grafos causa-efecto, Análisis sintáctico, Pruebas aleatorias, Pruebas adaptativas y Análisis transaccional.

La tercera parte ("Técnicas de prueba estructural") incluye aquellas técnicas que sí precisan del código fuente para llevarse a cabo. A estas alturas del curso el alumno ha obtenido la suficiente base matemática como para comprender los aspectos de las mismas. Se introduce un primer capítulo sobre elementos de teoría de grafos, capítulo básico para poder comprender la representación de programas y la medición de la complejidad. Se introducen posteriormente las revisiones de código e inspecciones para terminar con una nueva división en las técnicas; las que no ejecutan el programa, llamadas estáticas: Estimación de la complejidad, Ejecución simbólica, Análisis del flujo de datos y Análisis de los campos finitos; y las dinámicas, que sí precisan de la ejecución del programa, que son: Técnicas de cobertura del grafo de control y Pruebas mutacionales.

Por último, existe un bloque que trata sobre los estándares existentes sobre pruebas y planes de pruebas: unitarias, de integración y de sistema, centrándose fundamentalmente en el Plan de Verificación y Validación de Software [12]. Es conveniente que el alumno perciba que todo esto está convenientemente certificado por una serie de estándares, que van desde cómo realizar una prueba unitaria [11] hasta cómo debe documentarse un plan de pruebas [10]. Se estudian los estándares

ISO/IEEE, estableciendo una pequeña comparativa con los del Departamento de Defensa [13] de EE. UU.

La figura 2 presenta el temario resumido.

1 INTRODUCCIÓN A LA PRUEBA DE PROGRAMAS	1.1 Definición de la prueba 1.2 Dificultad de la prueba 1.3 Nociones de estrategia de prueba
2 TÉCNICAS DE PRUEBA FUNCIONAL	2.1 Análisis particional 2.2 Prueba de los límites 2.3 Grafos causa-efecto 2.4 Análisis sintáctico 2.5 Pruebas aleatorias 2.6 Análisis transaccional
3 TÉCNICAS DE PRUEBA ESTRUCTURAL	3.1 Elementos de la teoría de grafos 3.2 Revisiones de código 3.3 Análisis estático 3.3.1 Estimación de la complejidad 3.3.2 Ejecución simbólica 3.3.3 Análisis del flujo de datos 3.3.4 Análisis de los campos finitos 3.4 Análisis dinámico 3.4.1 Técnicas de cobertura del grafo de control 3.4.2 Pruebas mutacionales
4 VERIFICACIÓN Y VALIDACIÓN DE SOFTWARE	4.1 Plan de verificación y validación 4.2 Estándares de V&V

Figura 2. Temario resumido de la asignatura Metodología de la Programación

3. Método de enseñanza

Existe una combinación de clases teóricas y prácticas cuya disposición en el tiempo no está fijada de antemano sino en función de las necesidades del grupo. Claramente es una asignatura eminentemente práctica. No pueden explicarse los fundamentos de las pruebas sin realizarlas, ya sea en ordenador o en el papel.

Tras la presentación del fundamento teórico de cada técnica se fomenta la participación del alumno para la discusión general de la técnica. En este caso interesa que el alumno no sepa sólo cómo se realiza una técnica de prueba determinada sino también por qué se hace así. Los distintos libros y artículos que comentan técnicas específicas de prueba dejan la puerta abierta en todos los casos a la aparición de excepciones a las reglas y de requisitos iniciales no contemplados. Hay múltiples ejemplos, como pueden ser la no definición del tipo de gramática necesario para que pueda aplicarse el Análisis sintáctico en la que unos tipos determinados de gramáticas hacen muy difícil la generación de casos de prueba; o también el paso de parámetros a funciones y el tratamiento de vectores y punteros en el análisis del flujo de datos.

Por lo tanto existe la posibilidad de proponer nuevas reglas, siempre con cautela por el riesgo que conlleva, pero que hacen ver al alumno que también es capaz de diseñar sus propias técnicas o sus aproximaciones a técnicas ya existentes.

Se realiza posteriormente un conjunto suficientemente numeroso de ejemplos (de menor a mayor dificultad) hasta llegar a la comprensión global de la técnica.

4. Programación y herramientas

La inexistencia de herramientas específicas que realicen técnicas determinadas de prueba hace imposible que el alumno pueda seleccionar datos de prueba para un programa de forma automática.

Las únicas herramientas existentes en el mercado son herramientas llamadas de traza o seguimiento, las cuales simplemente se limitan a hacer un "debugging" del código en ejecución. Entre estas herramientas se encuentran por ejemplo Microsoft Test, SQLTest y algunos trazadores para C. Estos programas simplemente facilitan la gestión del entorno de traza para la observación del estado de una serie de variables. Podría considerarse en parte similar al flujo de datos dinámico (sin generar las dr-cadenas) para la observación de variables, tanto estáticas como dinámicas. Ni mucho menos existen herramientas que a partir de requisitos generen datos de prueba para ningún tipo de prueba funcional. Sería relativamente sencillo el diseñarlas pero quizá sería complejo el disponer de personal formado para poder ejecutarlas, y como ya se comentó anteriormente la fase de pruebas está un poco olvidada.

Para paliar esta carencia se van generando curso a curso aplicaciones por medio de proyectos fin de carrera. Estas aplicaciones sirven para que los alumnos puedan realizar la comprobación de los ejercicios planteados. De esta forma pueden observarse los fallos cometidos y las propias carencias de cada técnica. Para ello, y en una primera fase se ha trabajado con la técnica de Analisis del flujo de datos y con Estimación de complejidad.

Pese a todo siempre puede realizarse una prueba manual de los programas. Las técnicas de inspección tampoco necesitan, en su mayoría, más que una observación manual del código. De esta forma se combina el aspecto programatorio por parte de los alumnos (generación de código) con el de la realización de pruebas, bien manuales (todas) bien automáticas (flujo de datos y complejidad).

5. Lenguajes de programación utilizados

Existen técnicas de prueba independientes del lenguaje de programación, como es el caso de las funcionales, pero existen otras, la mayoría de las estructurales, en las que es necesaria la observación del código fuente. Las reglas de análisis del código, como es el caso del Análisis del flujo de datos, pueden ser perfectamente independientes del lenguaje, pero es necesario conocer éste para obtener, en este caso, las cadenas correspondientes a cada variable. Además existen entre los lenguajes de programación diferencias notables a la hora de tratar distintos aspectos de programación como puede ser el caso de la utilización de memoria dinámica.

De esta forma, y debido a la formación de la que parten los alumnos, se utiliza para la realización de ejemplos y programas el lenguaje Pascal, el cual no es el más conveniente. Sería preferible la utilización de otro tipo de lenguajes de mayor riqueza como puede ser el lenguaje C. El problema es que no es conocido por los alumnos lo que restringe las posibilidades. De esta forma cualquier entorno de trabajo para lenguaje Pascal es válido. De todos modos, en ningún caso se trabaja con lenguaje Pascal en trabajos más avanzados como los proyectos fin de carrera donde tanto el código que se analiza como la propia codificación de la aplicación se realizan en C.

Incluso, debido a la aparición de nuevos entornos de desarrollo (cuarta generación) sería también adecuada la utilización de 4GL's o a lo sumo de lenguajes orientados a objetos, como el C++ [5]. Se han desarrollado proyectos fin de carrera que aplican la técnica de Análisis del flujo de datos a programas escritos en C++ [7], [8]. Para estos tipos de lenguajes orientados a objetos es necesario

definir nuevas reglas de aplicación de las distintas técnicas estructurales. Basta con pensar en el tratamiento del flujo de datos en variables pertenecientes a objetos instancias de clases en C++ [8].

6. Trabajos prácticos

Como mínimo se emplea un tercio de las horas lectivas para la realización de prácticas y ejercicios en clase. Se proponen enunciados sobre las distintas técnicas, de forma que abarquen todas las posibilidades de cada una. Un conjunto de estos ejercicios (aproximadamente 5) se entrega al profesor una semana después de su exposición y se corrige en clase.

También se realizan controles periódicos en las horas de prácticas para comprobar el grado de asimilación de la asignatura.

Existen otras modalidades que se realizan o no dependiendo del interés del alumnado. Se han realizado programas en grupo que después se han probado mediante varias técnicas. En algunos casos la prueba de estos programas ha correspondido a grupos diferentes del que los realizó, de tal forma que, psicológicamente, la prueba ha sido más completa.

Una experiencia interesante, desarrollada únicamente durante el último curso, ha sido la exposición por medio de los alumnos de técnicas no incluidas en el temario, o de técnicas sí incluidas de las que se ha profundizado en sus reglas de trabajo. A partir de estos trabajos, el enriquecimiento ha sido mutuo, llegando en los casos más interesantes a la proposición de proyectos fin de carrera.

7. Material didáctico

El material didáctico se compone de los elementos normales:

- Presentación en clase por medio de transparencias y pizarra,
- Utilización de herramientas de seguimiento,
- Programas realizados por los propios alumnos,
- Libros relacionados (por desgracia escasos y demasiado generales)
- Lectura de artículos relacionados,
- Libro de apuntes de la asignatura con una colección numerosa de casos prácticos

8. Actividades relacionadas

El nuevo sistema de créditos prevé la obtención de créditos por parte de los alumnos mediante la realización de trabajos dirigidos en departamento. En este caso, cada año, hay varios alumnos que desarrollan una serie de trabajos, fundamentalmente relacionados con la mejora de las distintas técnicas. Estos trabajos son considerados posteriormente como anteproyectos para la realización de proyectos fin de carrera

Ya se han desarrollado con éxito 4 proyectos fin de carrera, 3 sobre Análisis del flujo de datos [6], [7]. [8] y 1 sobre Análisis sintáctico [9], este último en la E.U. de informática de la Universidad Politécnica de Madrid. En la actualidad se encuentran en fase de desarrollo otros 5 proyectos.

Aunque no se han llevado a cabo todavía, para el curso próximo está proyectado el desarrollo de seminarios monográficos sobre las distintas técnicas por parte de los alumnos que estén realizando el proyecto fin de carrera, con vistas a informar e interesar a nuevos alumnos del trabajo realizado a fin de diseminar el conocimiento específico que estos alumnos están adquiriendo en la realización de sus proyectos fin de carrera.

9. Conclusiones

Como conclusiones fundamentales podrían indicarse las siguientes:

- La asignatura se ha desarrollado con un éxito creciente durante los tres últimos años. El cambio del plan de estudios de la titulación impulsó la aparición de un temario específico de pruebas, aunque englobado en una asignatura con otros contenidos.

- Al comienzo hubo que salvar las reticencias de los alumnos ante lo que suponía una novedad para ellos. En algunos casos desconocían por completo lo que eran las pruebas de software.

- El interés demostrado por ambas partes ha creado un ambiente general favorable en el que la participación del alumno ha sido considerable. Los estudiantes han visto la utilidad del contenido de la asignatura, porque les ayuda a formarse como informáticos desde un punto de vista que no contemplaban al comienzo de sus estudios.

- La escasa bibliografía práctica existente sobre el tema induce al desencanto entre aquellos que buscan mejorar sus conocimientos. No existe ningún libro, al margen de esta universidad [2] y que yo conozca, al estilo de los que se publican sobre otras asignaturas, que contenga un número suficiente de ejercicios. Los libros conocidos [1], [3], [4] presentan fundamentalmente una vertiente teórica dejando de lado los casos prácticos.

- La aparente indefinición de algunas técnicas en cuanto al tratamiento de algunas excepciones (punteros en el análisis del flujo de datos, gramáticas recursivas en el análisis sintáctico, etc.) han inducido a la mejora de estas técnicas con resultados satisfactorios. En este caso parece observarse un estancamiento entre quienes trabajan en el mundo de las pruebas.

10. Bibliografía

[1] Test & Controle des Logiciels: Methodes, Techniques & Outils. Spyros Xanthakis, Michel Maurice, Antonio de Amescua, Olivier Houri, Luc Griffet, EC2, 1993.

[2] Cuaderno de apuntes de la asignatura Metodología de la Programación. Antonio de Amescua y Manuel Velasco, Universidad Carlos III de Madrid, 1996.

[3] Software testing : a craftsman's approach. Paul C. Jorgensen. CRC Press, 1995

[4] Testing object-oriented software. D. Firesmith. Prentice Hall.

[5] Object-oriented software techniques combining testing and metrics. Chu-Ming, Wong, Shih & Lee. Information and Management Sciences, 1995

[6] Prueba de programas: Análisis del flujo de datos. Belén Criado. Universidad Carlos III de Madrid, 1996. Proyecto fin de carrera dirigido por Manuel Velasco.

[7] Descomposición en unidades mínimas de prueba mediante la aplicación de técnicas del nivel de herencia en programación orientada a objetos. Pedro Pérez. Universidad Carlos III de Madrid, 1997. Proyecto fin de carrera dirigido por Manuel Velasco.

[8] Prueba de programas: Análisis del flujo de datos en programación orientada a objetos. Juan Antonio Martín. Universidad Carlos III de Madrid, 1997. Proyecto fin de carrera dirigido por Manuel Velasco.

[9] Generación de casos de prueba mediante la técnica de análisis sintáctico. Isabel Carrascoso. Universidad Politécnica de Madrid, 1997. Proyecto fin de carrera dirigido por Manuel Velasco.

[10] IEEE Standard for Software Test Documentation (Std 829-1983), IEEE Software Engineering Standards Collection, 1994

[11] IEEE Standard Software Unit Testing (Std 1008-1987), IEEE Software Engineering Standards Collection, 1994

[12] IEEE Standard Software Verification and Validation Plans (Std 1012-1986), IEEE Software Engineering Standards Collection, 1994

[13] Military Standard. Software Product Standards. DOD-Std-1703 (NS), Departamento de Defensa de EE. UU., 1987