Capítulo 3. Sistemas orientados a la mejora de la calidad del código

En este capítulo tratamos de analizar el panorama actual de los principales sistemas existentes que ayudan a mejorar la calidad del código. Por un lado sistemas orientados al aprendizaje de la programación y por otro, entornos y herramientas comerciales para el desarrollo de software de calidad. Hemos dividido los sistemas analizados en los siguientes apartados:

- **Sistemas de aprendizaje virtual de la programación**, de estos sistemas nos interesa el uso que realizan de la Web para facilitar el aprendizaje de la programación
- **Entornos de desarrollo para el aprendizaje de la programación**, se analizan las características de dos entornos locales orientados al aprendizaje.
- **Entornos de desarrollo comerciales**, consideramos importante también el análisis de entornos comerciales lo que aportan sus características avanzadas y las carencias que siguen teniendo en cuanto a la ayuda al desarrollo.
- **Entornos de colaboración para el aprendizaje y desarrollo de la programación**, analizamos aquí varios entornos colaborativos: unos orientados al desarrollo de aplicaciones de forma conjunta por parte de varios desarrolladores y otros más orientados al aprendizaje colaborativo de la programación.
- **Entornos de gestión de proyectos software**, consisten en entornos comerciales para gestionar proyectos software, la faceta que más nos interesa es que permiten compartir el proyecto entre varios desarrolladores y realizar un seguimiento de la evolución del proyecto.
- **Gestores de prácticas avanzados**, son entornos que han evolucionado desde la gestión académica pura: recopilación automática de trabajos a sistemas más complejos con una orientación colaborativa donde los usuarios pueden enviar, revisar y sobre estas revisiones corregir los trabajos.
- **Otros planteamientos en el aprendizaje de la programación**, se recogen en este apartado otros enfoques en la enseñanza de la programación y se analiza el impacto que pueden tener en la mejora de la calidad del software.
- **Técnicas de detección de errores**, se analizan en este apartado distintas técnicas de detección de defectos, centrándonos en las herramientas automáticas de análisis estático de código.

Se concluye haciendo una revisión de las características destacables y carencias de estos sistemas. Que posteriormente servirán como base para analizar las aportaciones de nuestro sistema.

3.1 Sistemas de aprendizaje virtual de la programación

3.1.1 El aprendizaje virtual a través de la Web

La Web abre nuevas vías de aprendizaje. Los estudiantes pueden utilizar contenidos y herramientas con soporte Web independientemente del sitio donde se encuentren. Esta independencia del lugar se refleja tanto a nivel local, es decir, en clase, en instalaciones comunes del centro de estudios, en su casa o en otro tipo de centro de estudios, como a nivel global, se pueden compartir materiales entre distintas universidades del mundo. Sin embargo, muchas aplicaciones de aprendizaje virtual en la Web se limitan a volcar contenidos estáticos con el uso de hiperenlaces, el estudiante se limita a leer texto y ver los gráficos de forma pasiva, con la única ventaja de una mayor relación entre los contenidos (si la navegación está bien diseñada); pero sin permitir realizar un aprendizaje activo con una verdadera interacción con el sistema y experimentar cuestiones teóricas en la práctica.

3.1.2 Carencias de las plataformas de enseñanza virtual estándares para el aprendizaje de la programación

Actualmente no podemos dejar de lado la enseñanza virtual, bien como complemento a la parte presencial o bien como alternativa al aprendizaje que se puede llevar en el aula.

Se han realizado varios estudios sobre los campus virtuales y la enseñanza virtual de las universidades españolas a un nivel general [MEC, 2003]. Dos de las conclusiones que se pueden extraer de estos estudios es que, en estos momentos, los campus virtuales adolecen de:

- Atención personalizada y continuada. Según un estudio realizado para el MEC en el 2003 [MEC, 2003] sobre los campus virtuales de las universidades españolas los servicios que ofrecen se centran en la organización del material docente para todos los alumnos y en herramientas de comunicación síncrona y asíncrona (conversación, mensajería y foros).
- Soporte on-line para el desarrollo de programas. Otro aspecto que aparece en el estudio citado anteriormente [MEC, 2003] es que la plataforma utilizada mayoritariamente es WebCT [WebCT, 2003]. Esta plataforma de e-learning no proporciona, de forma estándar, soporte para la realización on-line de ejercicios prácticos de programación, con lo cual el planteamiento de asignaturas de programación virtuales es básicamente teórico con un complemento de trabajos que realiza el alumno fuera de la plataforma y el profesor sólo corrige en sus resultados finales.

Estos dos aspectos inciden directamente, y quizás con mayor relevancia que en otros campos, sobre el aprendizaje de la programación y hace de la enseñanza virtual un campo difícil para incluir ejemplos dotados de la interactividad pero sobre todo se hace casi imposible la experimentación sobre programas reales y la creación de nuevos programas por parte del alumno como se suele realizar en los laboratorios de programación.

3.1.3 Libros electrónicos para como medio para el aprendizaje

Un libro electrónico es un sistema de información capaz de proporcionar a los usuarios un conjunto de páginas organizadas conceptualmente como un libro y que permite una interacción [Díaz et al, 1996]. El panorama de los libros electrónicos es muy amplio y hay grandes diferencias tanto en la interacción con el usuario como en el soporte en el que se proporciona.

Las principales características de los libros electrónicos son:

- Proporcionan contenidos en forma mayoritariamente de texto e imágenes que puede ir acompañado de algún material multimedia.
- Proporcionan mecanismos de navegación a través de todo el material, que como dijimos anteriormente está estructurado como un libro tradicional, es decir, existen una serie de temas, más o menos independientes, divididos en apartados de forma jerárquica. El usuario dispone de mecanismos para ir desde una unidad a otra.
- Algunos proporcionan ejercicios interactivos, mayoritariamente a partir de preguntas de respuesta múltiple.
- Algunos permiten realizar una auto-evaluación y evaluación del aprendizaje del alumno, normalmente esto se lleva a cabo mediante la asignación de una puntuación a los ejercicios que realiza el estudiante y el almacenamiento de esta.

3.1.4 La interacción en los libros electrónicos como base para facilitar el aprendizaje virtual de la programación

Los libros electrónicos son un soporte interesante para el aprendizaje virtual sobre todo en disciplinas técnicas donde pueden proporcionar una forma de experimentar con soluciones ya creadas, modificándolas o creando otras nuevas. Esto es lo que según Martínez-Unanue [Martínez-Unanue 2002] constituye un "laboratorio virtual". Si nos centramos en el campo del aprendizaje de la programación, un "laboratorio virtual" debería disponer de todas las herramientas de cualquier entorno de programación: editor, depurador, gestor de archivos…

Boroni [Boroni 2001] basa la importancia de los libros electrónicos en que proporcionan la posibilidad de aprendizaje activo y distingue entre libros electrónicos y proyectos de animación. Los proyectos de animación son sistemas aislados pensados para permitir a los estudiantes aprender viendo modelos de conceptos en la pantalla de ordenador. Podemos distinguir varios niveles en estos sistemas: visualización estática, muestran el estado del sistema en un momento determinado; animación, el estudiante puede ver el modelo en acción, evolucionando a través del tiempo; sistemas interactivos, permiten al usuario ver al sistema en acción como las animaciones; pero además permiten introducir datos de entrada, cambiar partes del sistema, etc. Este último nivel es el que permite al estudiante realizar un aprendizaje activo. Para Boroni los libros electrónicos serían un paso más y deberían integrar animaciones que permitan interactividad junto con materiales multimedia relacionando distintos conceptos mediante hiperenlaces.

La interactividad convierte los libros electrónicos en un medio activo de aprendizaje [Boyle 1994] [Fowler 1993] [Meyerowitz 1995]. Algunos de estos sistemas proporcionan acceso a entornos de programación con un editor y un compilador o intérprete. En estos sistemas, todos los ejemplos y ejercicios permiten un aprendizaje activo, es decir, el estudiante no sólo puede ver el ejemplo sino que si proporciona herramientas de edición, compilación y ejecución puede investigar sobre distintas variantes: ejecutarlo, cambiar ciertos aspectos y volver a ejecutarlo. Algunos libros electrónicos permiten realizar ciertos análisis de

programas y proporcionar realimentación sobre ellos, por ejemplo el sistema Ceilidh [Benford 1994] permite verificar la corrección de los programas introducidos por el estudiante.

3.1.5 Factores que influyen en la utilización de libros electrónicos

Boroni [Boroni 2001] recoge varias cuestiones que pueden hacer fracasar o al menos limitar su uso a libros electrónicos basados en animaciones pese a que puedan tener una gran calidad técnica:

- Problemas relacionados con la descarga, instalación y configuración del sistema. Pese a que el producto sea de una elevada calidad, si requiere que el usuario realice operaciones de descarga, instalación y configuración el uso de este material se reducirá drásticamente. Tanto los tutores en el aprendizaje como los estudiantes que lo van a utilizar no consideran adecuado emplear tiempo en estas tareas. Para evitar esto se deben realizar sistemas de tipo *plug-and-play* que no requieran instalación y configuración o que esta sea automática y requiera un tiempo mínimo.
- Dependencia de la plataforma. La dependencia de la plataforma siempre ha sido un elemento disuasivo en el uso generalizado de un producto. Mientras que dentro de una empresa normalmente hay unas líneas que unifican los sistemas informáticos utilizados en el ámbito de los estudiantes para su trabajo personal tienen quizás la más amplia variedad de plataformas y sistemas operativos dentro de una organización y debería ser posible utilizar el software en cualquier plataforma.
- Animaciones aisladas contra recursos de aprendizaje integrados. Si sólo se pueden utilizar animaciones de distintas fuentes y para partes muy limitadas de la materia que se está aprendiendo los estudiantes se retraen a la hora de utilizarlas. Hay que buscar el momento adecuado para utilizar cada una y además hay que familiarizarse con las características propias de cada una de ellas, lo cual vuelve a repercutir en el tiempo empleado en cuestiones que no son el objetivo central que es el aprendizaje. Se deben integrar los recursos dentro de un paquete que pueda ser utilizado en distintos momentos de aprendizaje de una materia.

La Web y en concreto los libros electrónicos en formato Web permiten evitar los inconvenientes descritos anteriormente. Evitan la tarea de realizar una instalación, permiten ser utilizados en cualquier plataforma y pueden agrupar distintos recursos sobre una materia y unificar la forma de trabajar con ellos.

3.1.6 Análisis de los libros electrónicos para la enseñanza de la programación

Los libros electrónicos en el campo de la enseñanza de la programación permitirían proporcionar al estudiante los conceptos básicos acompañados de la posibilidad ponerlos en práctica mediante la parte interactiva del libro.

Martínez-Unanue [Martínez-Unanue 2002] realiza un análisis de varios libros electrónicos orientados a la enseñanza de la programación y los resultados resumidos son los siguientes:

- Los libros electrónicos para la enseñanza de la programación siguen haciendo un amplio uso del texto, acompañado para elementos puntuales de algún recurso multimedia y muestran una estructura lineal.

- Los libros electrónicos orientados a la enseñanza de algoritmos incluyen además animaciones de algoritmos que contienen características hipermedia bastante sofisticadas.

Tabla 1. Comparación de distintos libros electrónicos para la enseñanza de la programación [Martínez-Unanue 2002]

	Principal medio	**Estructura de los temas**	**Navegación / control animación**	**Herramientas de anotación**	**Herramientas evaluación**	**Herramientas programación**
Algorithms in action	Texto y animaciones	Lecciones independientes, estructura jerárquica	Expansión / contracción. Control de animaciones	-	Preguntas	-
CAT y JCAT	Texto y animaciones	Lecciones independientes, múltiples vistas	Control de animaciones. Diferentes controles para profesor y estudiantes	-	-	-
HalVis	Texto, animaciones y audio	Lecciones independientes, 3 partes secuenciales por lección. 4 ventanas sincronizadas	Control de granularidad en las animaciones.	-	Varias clases de preguntas simples	-
Curso interactivo de programación en Pascal	Texto	Estructura lineal	Controles de navegación. Varias clases de índices.	-	Preguntas	Editor de programas independiente
ELM-ART[1]	Texto	Lecciones independientes, estructura jerárquica	Navegación adaptativa por los enlaces anotados.	Anotaciones con formularios HTML	Prueba de programas	Editor dirigido por la sintaxis. Evaluador y visualización elegante. Explicación de errores.
Exploring Computer Science Concepts with Scheme	Texto y animaciones	Estructura lineal	Control de animaciones. Facilidad de búsqueda en texto principal. Marcadores de usuario.	Libro de anotaciones	-	Intérprete del lenguaje independiente
KBS-Hyperbook Intrroduction to Java Programming	Texto e imágenes	Lecciones compuestas de secuencias de unidades de texto.	Índices. Pistas para navegación. Controles de navegación. Marcadores de usuario.	Anotaciones con formularios HTML	Carpeta de trabajos realizados	-
ProgramLive	Texto, animaciones y audio	Estructura lineal, lecciones cortas con exposiciones. Información adicional	Controles de navegación. Varias clases de índices. Facilidad de búsqueda. Marcadores de usuario.	-	Preguntas de arrastrar y soltar	-
WWW-Based C++ Course	Texto	Estructura lineal, 3 niveles	Controles de navegación. Varias clases de índices. Pistas visuales.	-	Prueba de programas	Compilación y ejecución en el servidor

[1] Peter Brusilovsky (su autor) considera que ELM-ART además de un libro electrónico es un sistema hipermedia adaptativo.

- Todos proporcionan facilidades en la navegación y el control de las animaciones.
- Desarrollan de manera más restringida otras facilidades potenciales como la posibilidad de que el alumno realice anotaciones propias y la posibilidad de que resuelva ejercicios relacionados con el contenido.
- Por último, los libros electrónicos rara vez proporcionan herramientas de programación o las proporcionan como aplicaciones independientes.

En primer lugar hacer notar que tras analizar las características los libros electrónicos desarrollan la parte teórica de la programación permitiendo "experimentar" sólo de forma limitada con los conceptos explicados. Esto se refleja en que la parte principal de estos libros es el desarrollo mediante temas teóricos; proporcionando herramientas de navegación secuenciales, mediante índices y algunos mediante herramientas de búsqueda. Fundamentalmente se basan en texto y alguna animación puntual para ilustrar algoritmos concretos; sin embargo, las animaciones son poco parametrizables y configurables con lo cual el estudiante sólo puede ver la animación preparada previamente, sin poder alterar sus parámetros para estudiar que sucede en distintos casos.

Dos sistemas (ELM-ART y KBS-Hyperbook *Introducction to Java Programming*) ofrecen servicios que permiten al estudiante realizar anotaciones y sólo uno dispone de un libro de anotaciones personal (*Exploring Computer Science Concepts with Scheme*). Sin embargo, estos sistemas más o menos sofisticados sólo permiten almacenar notas personales sin poder compartir conocimiento y experiencia con el resto de los usuarios del sistema.

La evaluación del estudiante se limita, salvo en tres casos a preguntas que el usuario tiene que responder. Es interesante el planteamiento de KBS-*Hyperbook Intrroduction to Java Programming* que permite una evaluación (aunque no automática) mediante los trabajos que van realizando los estudiantes y el de los sistemas ELM-ART y WWW-Based C++ Course que permiten la evaluación de programas reales.

3.1.7 Herramientas de programación en los libros electrónicos

Las herramientas de programación en los libros electrónicos son bastante limitadas; varios libros electrónicos (*Algorithms in action*, JCAT, HalVis, KBS-*Hyperbook Intrroduction to Java Programming*) no proporcionan ninguna en absoluto. Otras permiten utilizar un entorno independiente: *Curso interactivo de programación en Pascal*, el entorno Turbo Pascal; *Exploring Computer Science Concepts with Scheme* un intérprete de Écheme. Y sólo dos de las herramientas analizadas permiten compilar y ejecutar / interpretar programas que puede crear el usuario más o menos libremente: ELM-ART que permite crear, depurar y ejecutar programas en LISP y WWW-Based C++ Course que permite compilar y ejecutar programas en C++ sobre un servidor con la condición de que sólo utilicen la entrada – salida estándar.

WWW-Based C++ Course

En el artículo *Teaching C++ on the WWW* [Hitz 1997] Hitz y Kögeler presentan un libro electrónico en el que plantean como uno de los objetivos fundamentales de diseño el proporcionar al estudiante medios adecuados para la experimentación y plantean que el entorno no sea local debido a dos inconvenientes que esto acarrearía: las diferencias entre distintos entornos C++ y los inconvenientes de bajar los ejemplos de un repositorio e incorporarlos al entorno.

La interfaz que proponen es una interfaz Web que permite editar, compilar y ejecutar pequeños ejemplos en el servidor. La interfaz está preparada para un tipo de ejercicios que

consisten en completar fragmentos de código dentro de un programa base, por lo que muestra en una página Web el código fijo como texto de la página y un campo de texto en la parte donde el estudiante tiene que completar código. Una vez realizada esta operación un CGI compila el código fusionado en el servidor y muestra los errores emitidos por el compilador en la página Web. Permitiendo al estudiante modificar el texto del fragmento de programa introducido.

Una vez que la compilación es correcta, el entorno permite ejecutar el programa en el servidor, para lo cual se pide al estudiante los datos de entrada del programa y se muestra la salida correspondiente, por lo que los programas realizados en este entorno se tienen que limitar a la entrada / salida estándar.

ELM-ART

Schwarz, Brusilovsky y Weber [Schwarz 1996] proponen añadir inteligencia en los libros electrónicos y que realicen automática labores que normalmente realizan en clase los profesores humanos. Esto sería interesante en dos aspectos: reducir la carga de trabajo del profesor, aunque este pueda seguir supervisando el trabajo y sustituir al profesor cuando no esté disponible. Estas cuestiones se investigan en el dominio de los Sistemas Tutores Inteligentes. Existen sistemas Tutores Inteligentes que permiten apoyar el proceso de solución de problemas de un estudiante, proporcionar un análisis sobre la solución de un problema y construir un camino de aprendizaje para cada estudiante, incluyendo una selección de temas, ejemplos y problemas ajustados al aprendizaje del usuario. El planteamiento de este sistema consiste en integrar los Sistemas Tutores Inteligentes con entornos de programación, materiales on-line y las características interactivas de los libros electrónicos.

ELM-ART, el sistema donde ponen en práctica su propuesta, es un sistema basado en Web que permite el aprendizaje de la programación en lenguaje LISP. El sistema dispone de contenido teórico por el que se puede navegar, y además los ejemplos están integrados en un evaluador que permite modificarlos y ejecutarlos. Cualquier error en tiempo de ejecución se explica de forma detallada y ajustada a los principiantes y a veces también se proporcionan pistas para solucionar estos problemas.

El sistema tiene una arquitectura cliente – servidor lo cual permite optimizar el empleo de recursos ya que es necesario manejar bases de datos con gran cantidad de texto y el sistema es costoso en tiempo de ejecución; de esta forma varios alumnos se pueden conectar al mismo servidor del sistema para trabajar con él.

DSTool

DSTool [Sama 2003] [Sama 2005] es una herramienta global para el trabajo con estructuras de datos, desarrollada por el grupo HCI-RG, cuyo principal objetivo es facilitar todas las tareas que sigue el estudiante, desde el aprendizaje hasta la depuración de aplicaciones construidas por el usuario. Concretamente, DSTool proporciona soporte para las siguientes actividades:

- Aprendizaje. Por medio de su tutorial teórico formado por texto, imágenes estáticas y dando la posibilidad al usuario de interactuar con aplicaciones de prueba correspondientes a diferentes estructuras de datos. Estas aplicaciones permiten al usuario realizar distintas operaciones disponibles con la estructura de datos introduciendo los datos que el usuario decida y visualizando gráficamente su estructura además de la información de determinadas propiedades.

- Programación. Gracias a la biblioteca de clases de DSTool, biblioteca que permite programar cualquier aplicación en lenguaje Java utilizando las diferentes estructuras de datos disponibles.
- Evaluación y depuración. Proporciona un entorno gráfico de depuración y evaluación de la estructura de datos.

Las características más destacadas de esta herramienta son:

- Integración de actividades. Por un lado el tutorial sobre cada una de las estructuras de datos junto con las aplicaciones de prueba que permiten al estudiante realizar un aprendizaje activo experimentando con estas visualizaciones. Además, la biblioteca de clases permite programar y depurar aplicaciones reales
- Permite la interacción con gran número de estructuras de datos genéricas.
- Permite una evaluación y depuración gráfica, mediante el panel de depuración, el inspector de objetos y propiedades y el explorador de estructuras de datos.
- El programa y su visualización se mantienen sincronizados en tiempo real.
- Persistencia de las estructuras de datos, pueden guardarse en un archivo XML con los datos almacenados.
- Entorno inteligente. Dispone de un sistema experto capaz de aconsejar sobre la mejor estructura de datos a utilizar en cada instante. Dependiendo de las operaciones que se hayan realizado sobre la estructura actual.

Cuando se quiere visualizar el comportamiento de una estructura de datos ante determinadas operaciones no se utiliza un lenguaje de script que permita describir la animación; sino que simplemente, se realiza el programa en Java que realice las operaciones con la estructura o estructuras de datos adecuadas y el propio sistema se encarga de generar la representación gráfica y mantenerla actualizada a medida que se van realizando las operaciones. Es decir, utilizando DSTool se puede programar una aplicación real y si queremos podemos visualizar sus estructuras de datos en cualquier momento.

El soporte de este sistema se proporciona íntegramente mediante Web, el tutorial consta de distintas páginas con hiperenlaces e imágenes y a la vez permite bajarse las aplicaciones de prueba que están programadas en Java para proporcionar una mayor interactividad; para evitar problemas de incompatibilidades de versiones de Java se ha utilizado la tecnología *Java Web Start* para desplegar estas aplicaciones.

Snapshots of the Theory of Computing

El equipo de Boroni [Boroni 2001] desarrolló en la Montana State University un libro electrónico sobre Web para el aprendizaje de teoría de la computación (computabilidad) denominado *Snapshots of the Theory of Computing*, realmente este libro no trata de enseñar a programar y no se refiere a ningún lenguaje de programación, se trata de trabajar con autómatas de distintos tipos; pero se ha analizado aquí por la flexibilidad que aporta en la interacción con el usuario. La característica más notable es la inclusión de módulos de animaciones para el aprendizaje activo que están estrechamente integrados con el texto. El estudiante puede trabajar con autómatas mostrados mediante applets, esto permite realizar una interacción con ellos de varias maneras:

- Introducir una cadena arbitraria de entrada.

- Controlar la forma de ejecución: paso a paso o todo seguido hasta finalizar la ejecución.
- Se muestra el cambio de estado mediante indicadores que se mueven sobre la figura del autómata.
- El autómata finito puede ser modificado en sus estados y transiciones sin ninguna limitación.
- El mismo applet permite representar distintos ejemplos de autómatas finitos con lo cual lo podemos utilizar en distintos puntos de la materia.
- El applet está programado para validar el autómata construido por el usuario y puede realimentar al usuario automáticamente con esta validación.

En el análisis que realizan los propios autores sobre el sistema, destacan dos cuestiones importantes que son: la necesidad de emplear gran cantidad de trabajo para desarrollar cada uno de los applets que soportan las animaciones y la gran importancia de la usabilidad del sistema. Por otra parte, Moroni cita dos obstáculos como los más destacados en el desarrollo de su sistema: la carencia de un lenguaje de marcado matemático estándar y ampliamente soportado por los navegadores y la dificultad para guardar archivos desde un applet, este último punto impide que un estudiante guarde su trabajo de modificación de un autómata de una sesión de estudio para otra.

Sistema de evaluación del aprendizaje de la tecnología orientada a objetos

Seffah [Seffah 1999] describe un sistema basado en Web que permite la utilización de una red de conocimiento adaptativa para el aprendizaje de la programación orientada a objetos. Aunque este no es un libro electrónico propiamente dicho ya que se centra en la creación de un modelo de usuario simple mediante la evaluación mediante preguntas, por otra parte proporciona información textual al alumno de forma parecida a los demás libros electrónicos.

El sistema estructura el conocimiento en una serie de unidades de conocimiento relacionadas en una red en la que se establecen relaciones de precedencia. Para poder pasar de una unidad a otra se deben resolver correctamente unas preguntas que están asociadas a cada concepto; a partir de esto se crea un modelo de usuario que el acceso a nuevas unidades de conocimiento. Las preguntas tienen asociados objetos solución que explican cual es la respuesta correcta a la pregunta correspondiente y realizan una breve explicación. Sin embargo, tienen la experiencia de que la solución para una pregunta muchas veces es incompleta, ambigua o incomprensible para el estudiante. Por lo que han desarrollado lo que denominan objetos note-box que ayudan a alcanzar una mejor comprensión proporcionando mayor información por distintos medios: bibliografía, referencias a artículos, URLs de recursos en Internet, direcciones de correo de expertos.

3.1.8 Conclusiones sobre los libros electrónicos

Hemos visto en este apartado el planteamiento general de los libros electrónicos, sus características fundamentales. Uno de los puntos clave es **permitir posibilidades de interacción del estudiante con los programas** [Boroni 2001], en el sentido de poder modificarlos y comprobar los resultados de esta modificación. Si nuestro principal propósito es el aprendizaje de la programación el sistema debería proporcionar un entorno de programación que permitiera esto. Sin embargo, según Martínez-Unanue [Martínez-

Unanue 2002] **uno de los aspectos más deficientes de los libros electrónicos son las herramientas de programación**. Y considera que es necesario avanzar en dos líneas:

- Las facilidades que proporcionan los entornos para el aprendizaje de la programación en la ejecución y análisis de programas.
- Libros electrónicos orientados a otros campos permiten una mayor libertad en la experimentación del estudiante.

Por tanto, la integración de herramientas de programación en los libros electrónicos sería un avance cualitativo. No es una tarea simple debido a que la complejidad del sistema completo crece sustancialmente. Además, la interfaz de usuario debe ser rediseñada para permitir integrar de forma consistente tanto a los libros electrónicos como a las herramientas de programación. En esta línea están las propuestas *Teaching C++ on the WWW* [Hitz 1997] y ELM-ART [Schwarz 1996] vistas en el apartado anterior, las dos utilizan un sistema cliente-servidor sobre Web centralizando el procesamiento del código en el servidor. El tratamiento en el servidor además permite obviar los inconvenientes encontrados en la utilización de applets para realizar el procesamiento por parte del sistema propuesto por Boroni [Boroni 2001] que estaba limitado por las restricciones de seguridad que impone del sistema *sandbox* de los applets, como la posibilidad de guardar el trabajo realizado por el estudiante. El segundo sistema además de proporcionar información sobre los errores del programa es el único que trata de orientar al estudiante en la corrección del mismo.

Otro factor importante para no limitar el aprendizaje del estudiante, es **permitir que trabaje con código de problemas reales** (que en ocasiones pueden ser complejos) y no solamente con ejemplos simples, ya que esto puede impedir que el estudiante sepa enfrentarse a situaciones reales y por tanto, su aprendizaje no sea completo. Esto condiciona la necesidad de que el entorno sea lo suficientemente potente para poder tratar con proyectos, por ejemplo, con gran cantidad de archivos.

Como último factor de éxito para los libros electrónicos queremos resaltar la necesidad de la utilización de **la Web como medio de comunicación entre el estudiante y el sistema** y evitar los problemas señalados por Boroni [Boroni 2001]: instalaciones complejas, incompatibilidades entre sistemas y necesidad de integración con otros recursos. Mediante la integración de un sistema en un entorno Web que funcione sobre un navegador, conseguiremos evitar la instalación en el ordenador del usuario y los problemas derivados del uso de distintas plataformas con lo cual el usuario para utilizar el sistema sólo tendrá que conectarse a la URL adecuada y autenticarse.

3.2 Entornos de desarrollo para el aprendizaje de la programación

3.2.1 Requisitos de un entorno de programación

Normalmente cuando se analizan las necesidades para un estudiante de programación se da mayor importancia al lenguaje de programación que al entorno empleado. Sin embargo, actualmente los entornos de programación permiten aportar gran valor añadido al lenguaje. La experiencia cambia radicalmente al programar en un lenguaje utilizando las distintas herramientas desde la línea de comandos o desde un entorno integrado. Además, existen gran variedad de entorno para distintos lenguajes que se podrían adaptar de forma diferente a determinados requisitos. Por otra parte, a medida que el proceso de desarrollo de

software se va haciendo más complicado también son más necesarios entornos que ayuden al programador a utilizar distintas técnicas y herramientas.

Michael Kölling plantea en su tesis [Kölling 1999c] varios requisitos para un entorno de programación orientado a estudiantes de primer curso.

1. Facilidad de uso. El entorno se debe adaptar a los programadores principiantes. Un estudiante no debería tener que invertir mucho tiempo en aprender como se utiliza el entorno. Debería ser intuitivo y no sobrecargar al estudiante.
2. Integración de herramientas. Las herramientas importantes para el desarrollo de software, como el compilador, el depurador y un visor de clases deberían estar lo más integradas posible en el entorno.
3. Soporte para orientación a objetos. El entorno debería dar soporte a clases y objetos en sus construcciones básicas. Incluso debería permitir la interacción y manipulación de objetos.
4. Soporte para reutilización de código. La reutilización de código se cita muchas veces como una de las motivaciones básicas del paradigma de orientación a objetos. Para enseñar buenas prácticas de programación y con problemas realistas es necesario que el entorno permita utilizar bibliotecas de clases y desarrollar nuevas bibliotecas.
5. Soporte para el aprendizaje. Existen técnicas con gran valor para el aprendizaje de la programación: visualización, soporte a la experimentación y pruebas interactivas. El entorno debería soportar estas técnicas.
6. Soporte para el grupo. Debería ser posible que los estudiantes trabajasen en grupos sobre los proyectos asignados.
7. Disponibilidad. El entorno debe estar disponible a un coste razonable y se debe poder ejecutar en la infraestructura disponible.

3.2.2 Sistemas que plantean un entorno de desarrollo para el aprendizaje de la programación

La filosofía de estos sistemas consiste en proporcionar un entorno que facilite todas las tareas relacionadas con el desarrollo a los estudiantes. Los entornos de trabajo para desarrollo de software están pensados normalmente para profesionales con experiencia. La gran cantidad de opciones disponibles y de información proporcionada por estas herramientas hace difícil su comprensión por parte de estudiantes que todavía carecen de muchos conceptos necesarios para manejar las características avanzadas de las que disponen dichas herramientas.

AnimPascal

AnimPascal [Satratzemi 2001], es un ejemplo de entorno de aprendizaje, su característica más destacada es que permite guardar el camino seguido por cada alumno en la solución de un problema. El propósito de AnimPascal es ayudar a los estudiantes a comprender las fases de desarrollo, verificación, depuración y ejecución de un programa.

AnimPascal permite guardar las acciones de los estudiantes. Se consideran estas acciones como pasos en el proceso de solución del problema. Esto permite comprender la forma en que cada estudiante resuelve el problema. A partir de esta información, AnimPascal tiene dos objetivos: ayudar a los programadores principiantes en todo el proceso de desarrollo

hasta la ejecución de un programa y ayudar a los profesores para descubrir "lagunas" de conocimiento y problemas de aprendizaje de sus estudiantes.

Cada vez que un estudiante recompila se guarda automáticamente la última versión del código fuente y la correspondiente salida del compilador. El profesor puede recorrer las distintas compilaciones para buscar ver los errores cometidos.

BlueJ

BlueJ [Kölling 2003], es un entorno de aprendizaje enfocado a la programación orientada a objetos, que utiliza Java como lenguaje. Una vez instalado el sistema en el ordenador local, el estudiante dispone de un entorno que está dotado de todas las herramientas que necesita para desarrollar una aplicación con un lenguaje orientado a objetos. Hace énfasis en la relación entre el diagrama de clases y el código para facilitar la adquisición de conceptos de orientación a objetos.

A través del diagrama de clases como vista principal permite un estilo de interacción diferente a otros entornos. Mediante este diagrama el estudiante visualiza las clases y sus relaciones; además permite la creación de nuevas clases y nuevas relaciones gráficamente.

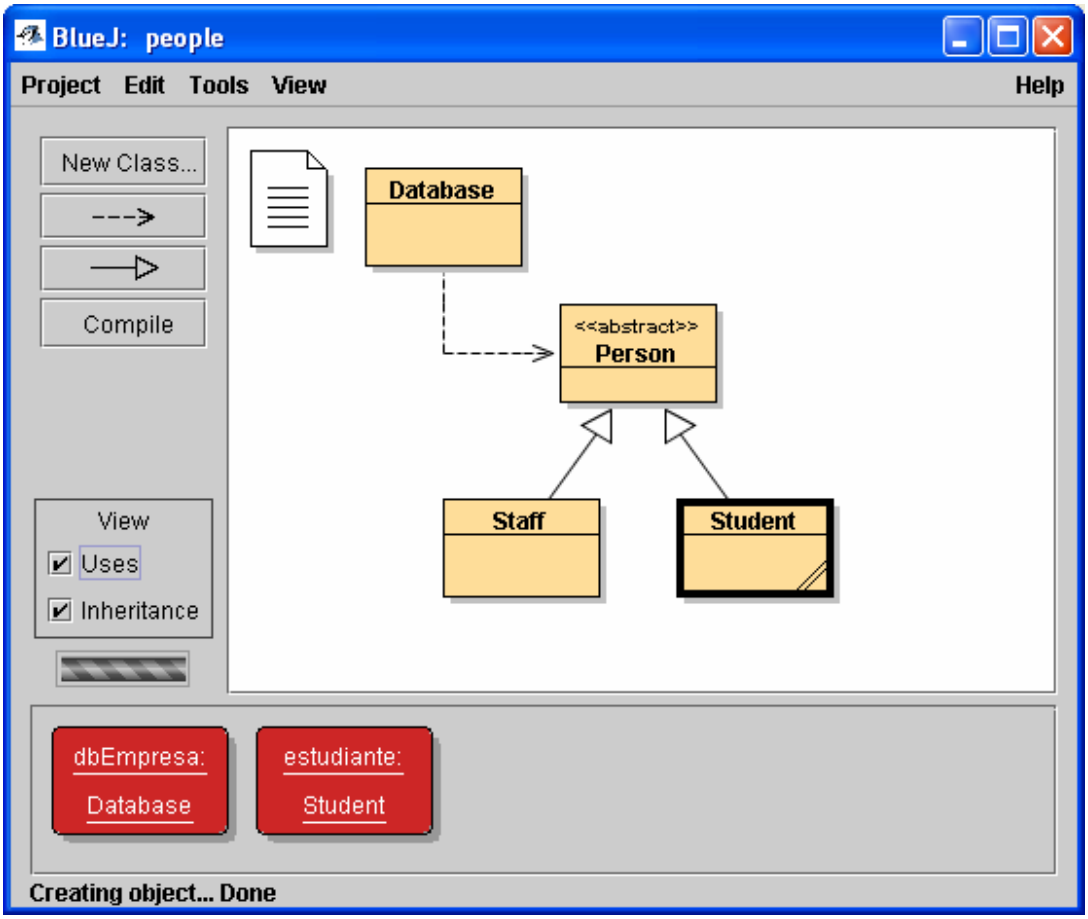

Figura 2. Panel de clases de BlueJ con varios objetos instanciados

El entorno permite la interacción con clases y objetos mediante sus menús contextuales. Con el menú contextual de las clases podemos crear instancias que aparecerán en el "banco de objetos". Sobre estas instancias podemos realizar una inspección que muestra el estado del objeto e invocar a cualquier método público.

Las diferentes zonas y operaciones que se pueden aplicar sobre los iconos de clases y objetos permiten visualizar la diferencia que muchas veces es difícil de ver para el alumno.

Por último, esta interacción gráfica permite dar al entorno una gran simplicidad. Los alumnos principiantes necesitan herramientas diferentes que los ingenieros profesionales [Kölling 1999b].

En una evaluación de esta herramienta realizada con encuestas a estudiantes después de su utilización se obtuvieron como puntos positivos la utilidad del entorno, en especial la funcionalidad del *banco de objetos* y la integración del editor de código fuente y los mensajes de error del compilador. La mayoría de los puntos negativos estaban relacionados con la estabilidad del producto y la dificultad en la instalación. A través de los exámenes y presentaciones de prácticas se constató que los estudiantes tenían un mejor entendimiento de los conceptos de orientación a objetos que anteriormente.

Los propios autores de BlueJ detectan varias áreas con problemas potenciales: BlueJ está planteado como un entorno para programadores principiantes, es interesante que los estudiantes a medida que vayan cogiendo experiencia se vayan familiarizando con otros entornos más profesionales. Por otra parte, es interesante que esta transición este monitorizada por el profesor para conseguir la correcta transferencia de BlueJ a otro entorno. Por último, según el planteamiento pedagógico de los autores de BlueJ hay peligro de que se dedique mucho tiempo a conceptos de orientación a objetos y se descuiden otros conceptos como las estructuras de datos y algoritmos; hay que tener en cuenta que estos conceptos siguen siendo importantes y dedicarles tiempo para que los estudiantes también los entiendan y puedan aplicarlos correctamente.

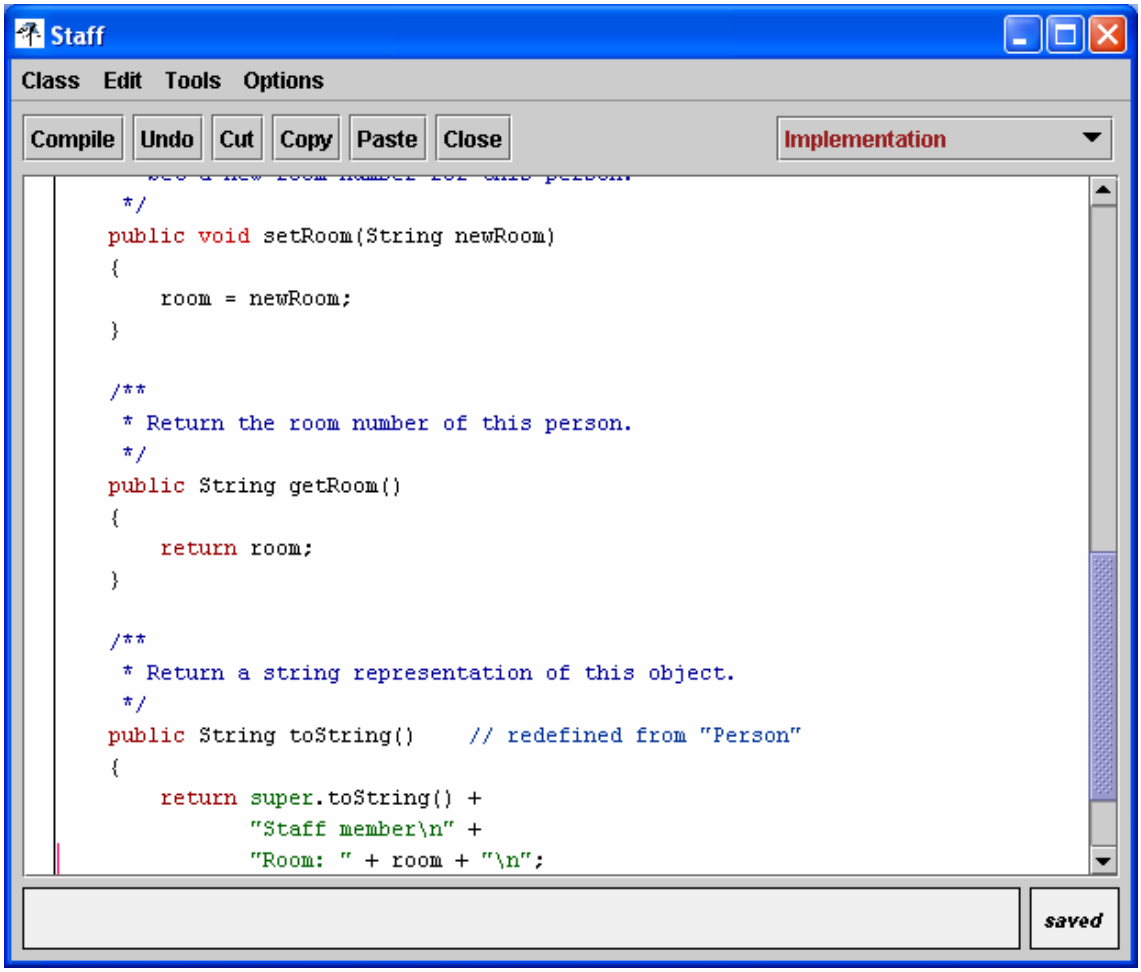

Figura 3. Panel de edición de clases en BlueJ

Las mejoras que están estudiando incorporar al entorno son soporte para trabajo en grupo y la incorporación de scripts que permitan incorporar dentro de tutoriales la interacción con el entorno.

3.2.3 Conclusiones sobre los entornos de desarrollo

Los dos sistemas analizados son **programas de escritorio mono-usuario**. AnimPascal, permite el desarrollo de programas en lenguaje Pascal, mientras que BlueJ esta diseñado para programar con el lenguaje Java. El estudiante dispone, en cualquiera de los dos sistemas, de herramientas de edición, compilación, depuración y ejecución. Los dos entornos utilizan compiladores estándar de sus respectivos lenguajes para realizar la compilación y muestran al usuario los errores encontrados por el compilador correspondiente.

El entorno BlueJ aporta la vista del diagrama de clases y el banco de objetos y la posibilidad de manipulación directa por parte del estudiante para facilitar que el estudiante se cree el modelo mental adecuado para entender la relación entre clases y objetos y el uso adecuado de cada uno de ellos.

AnimPascal, realiza un **seguimiento de las acciones que va realizando el estudiante** para solucionar los errores que encuentra en la compilación y en la ejecución del programa. Esto puede orientar al profesor al revisar el registro de cada estudiante sobre los problemas conceptuales que puede tener cada estudiante y así ayudarle a solucionarlos.

Ninguno de los dos entornos proporciona ayuda al estudiante sobre la interpretación de los errores cometidos, ni en la solución de los mismos. Sería necesaria la intervención del profesor para solucionar estos problemas.

Otro problema añadido de estos entornos es la **necesidad de instalación y configuración del programa por parte del usuario**. Hay que proporcionar al usuario las instrucciones de instalación y configuración, lo cual aunque sea un proceso sencillo ralentiza la puesta en marcha del aprendizaje del estudiante.

3.3 Entornos de desarrollo comerciales

En este apartado se estudiarán algunos de los entornos de desarrollo integrados más comúnmente utilizados por la comunidad de programadores de Java y las posibilidades que ofrecen tanto de trabajo en grupo como de prevención de errores.

3.3.1 Entornos de desarrollo integrados

Los Entornos de Desarrollo Integrados, también conocidos como IDEs (de sus siglas en inglés Integrated Developer Environment) fueron la natural evolución de los editores enfocados a la programación. Inicialmente constaban de un editor de texto que sólo se ocupaba de la edición de texto. Posteriormente se le fueron añadiendo distintas funcionalidades:

- Llamada al compilador desde el entorno de edición. El propio entorno tiene opciones para configurar e invocar al compilador y mostrar los resultados de la compilación. Además, podemos utilizar los mensajes de error para acceder a los puntos del código fuente donde se han localizado los errores para corregirlos.
- Resaltado de sintaxis de los distintos lenguajes de programación. A medida que se va escribiendo el código fuente el editor escribe con distintos colores cada palabra

del código fuente dependiendo del tipo de token de que se trate: palabras reservadas, identificadores, literales, comentarios, etc.

- Depuración. Permite realizar trazas visuales de la ejecución de los programas, en las que se puede tener en pantalla simultáneamente la línea de código en ejecución dentro del código fuente, la pila de llamadas a métodos que se han realizado, y los valores de las variables que queramos visualizar.
- Gestión de proyectos. Permite configurar la estructura de directorios para el proyecto, dejar los archivos compilados .class en un directorio determinado y de la misma forma con los demás tipos de archivos.
- Ayuda a la programación. Se puede invocar a la ayuda de forma contextual, es decir, pulsando una tecla de función sobre una clase o un método aparece toda la documentación asociada a ese elemento, sin necesidad de que el programador haga una búsqueda.
- Autocompletado de código. A medida que se va escribiendo el código, el editor va proponiendo los métodos o atributos correspondientes a la clase del objeto escrito para que el programador no tenga que acordarse de memoria de los nombres exactos de los métodos disponibles para una clase.
- Diseño visual de interfaces de usuario. El programador puede diseñar formularios simplemente arrastrando y soltando los distintos componentes que sean necesarios a partir de la paleta de componentes.

3.3.2 JBuilder

Jbuilder es un IDE para el lenguaje de programación Java desarrollado por la empresa Borland. JBuilder. Entorno de programación de lenguaje Java que dispone de todas las características comunes a los IDE descritas en el apartado anterior.

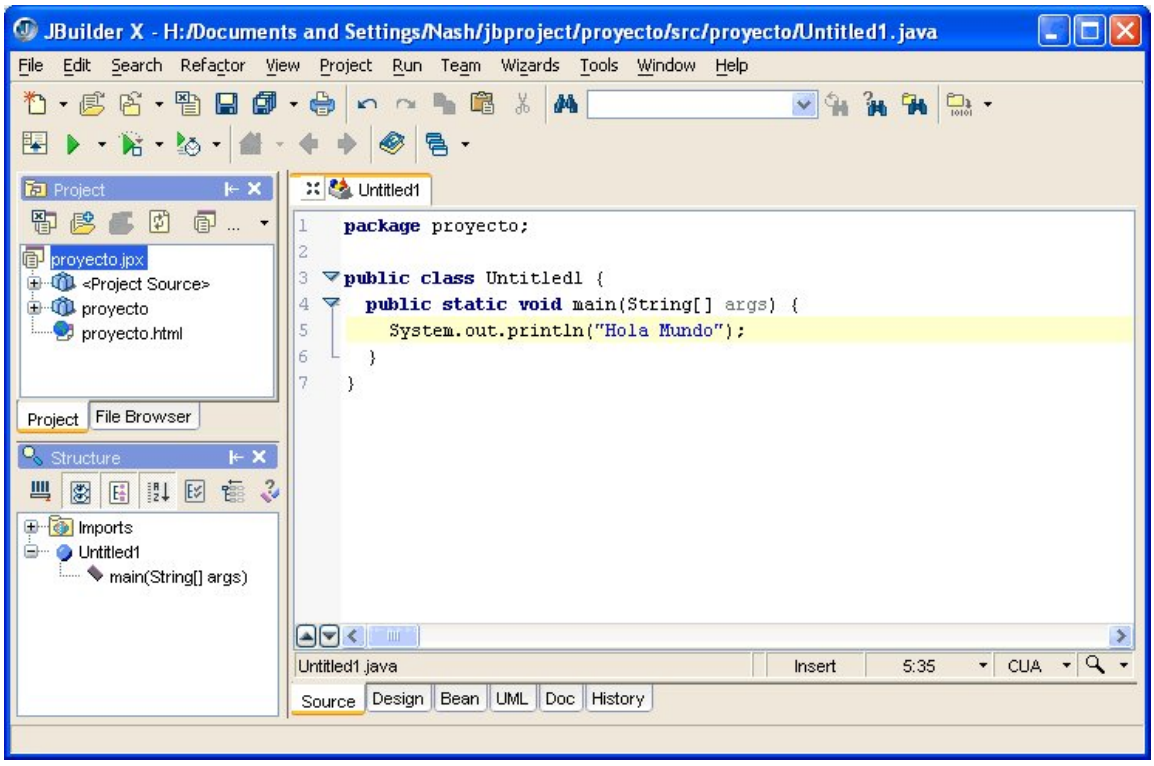

Figura 4. Entorno de desarrollo integrado JBuilder.

En la Figura 4 podemos apreciar su aspecto general: el área más amplia la ocupa el panel de edición de código; en la parte superior los menús y barras de herramientas que permiten acceder a todas las herramientas del entorno; en la parte superior izquierda el panel de gestión del proyecto donde se puede acceder a todos los archivos del proyecto.

En la zona de edición cambiando de solapa (en la parte inferior) podemos acceder a la documentación generada por javadoc, a un control de versiones propio de JBuilder y a la representación de la aplicación mediante un diagrama de clases en notación UML.

Además permite la incorporación de herramientas externas que ayuden en el proceso de desarrollo como la herramienta Ant que permite automatizar la construcción y el despliegue de un proyecto medianamente complejo.

Desde este entorno podemos utilizar determinadas herramientas para trabajo en grupo; en concreto facilita la conexión con repositorios CVS.

En cuanto al tratamiento de errores dispone de un agente que revisa el código a medida que se va tecleando lo cual permite evitar errores de sintaxis y declaración de variables. Cuando se realiza una compilación explícita dispone de un panel de errores que permite su localización de los archivos fuentes del proyecto; pero no ofrece ninguna ayuda especial para su corrección. Dispone de un modo de ejecución en depuración para buscar la causa del error.

3.3.3 NetBeans

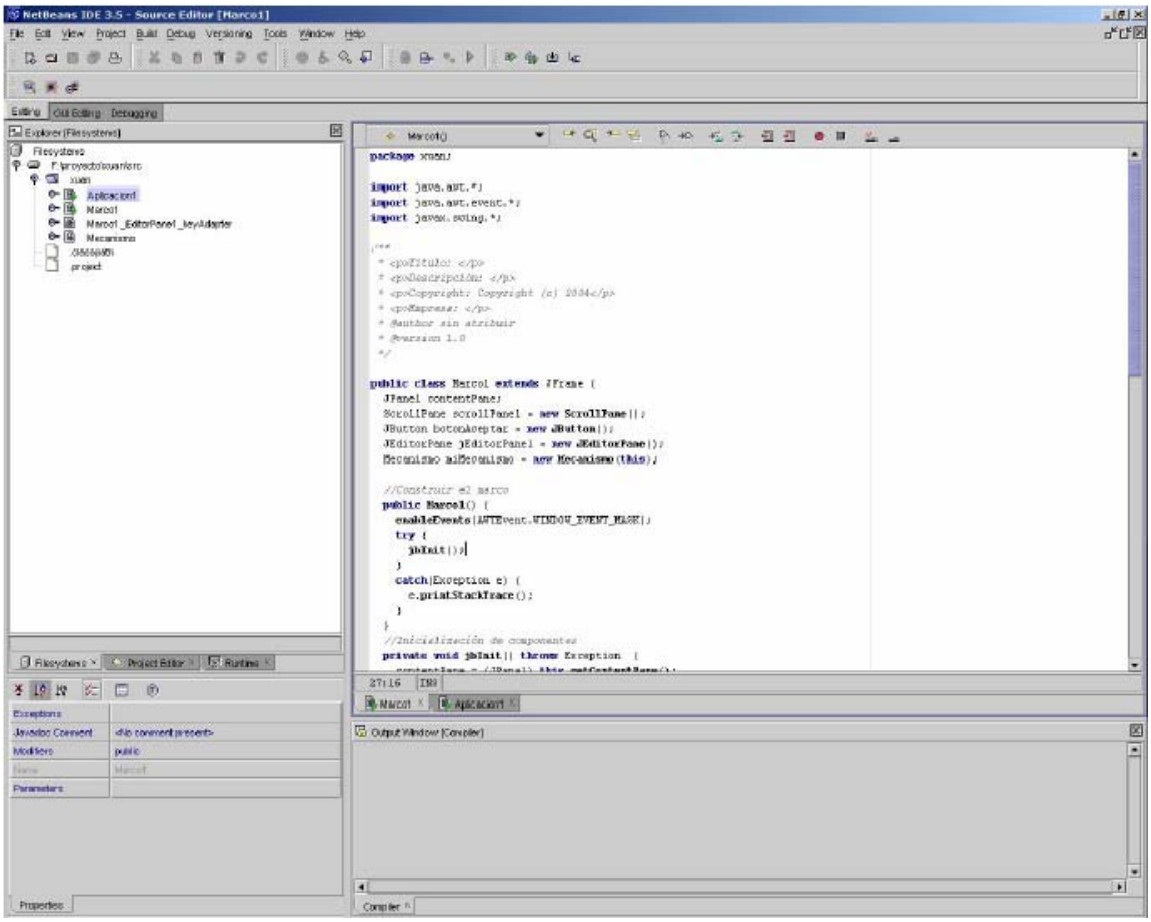

Figura 5. Entorno de desarrollo integrado NetBeans.

Entorno de programación de lenguaje Java, que dispone de todas las características comunes a todos los IDEs, está completamente escrito en Java de forma que es portable entre plataforma.

En la Figura 5 se puede ver una vista general del entorno, que por otra parte es similar a los otros dos estudiados en este apartado: gran área de edición donde el desarrollador interactúa con el código fuente, panel que permite la gestión de los archivos del proyecto, panel de visualización de los mensajes de error resultado de una compilación.

Incorpora otras herramientas externas, además de compilador, depurador, ...
Dispone de opciones de trabajo en equipo. Consisten básicamente en la conexión con repositorios CVS.

3.3.4 Eclipse

Eclipse es un IDE diseñado, en sus inicios, por la compañía IBM. Se caracteriza por la filosofía que guió desde el principio su desarrollo: "Eclipse, un IDE para todo en general, pero para nada en particular".

En la Figura 6 podemos apreciar su aspecto que encaja en líneas generales con el resto de los IDEs analizados. Barra de herramientas en la parte superior, área de edición amplia, panel de gestión de los archivos del proyecto en la parte superior izquierda.

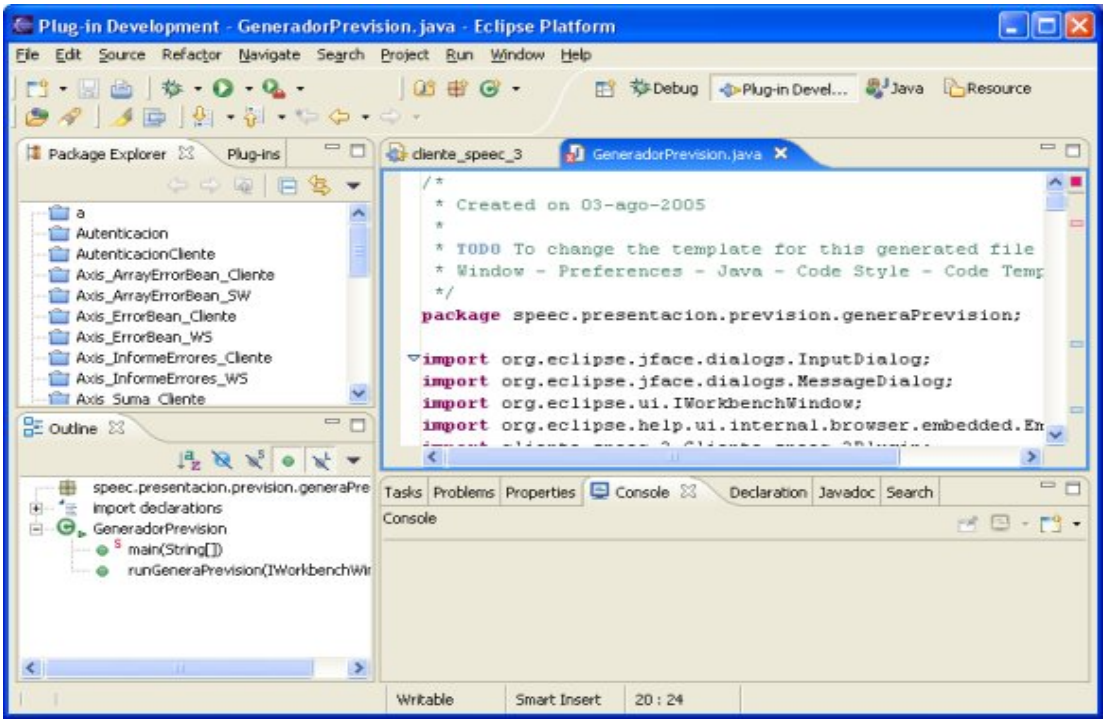

Figura 6. Entorno de desarrollo integrado de Eclipse.

La característica diferencial de Eclipse es su capacidad de ampliación mediante "plug-ins" que son pequeñas aplicaciones con una interfaz determinada para un uso específico dentro del proceso de desarrollo de una aplicación. Se integran dentro del IDE y pueden acceder a las características del entorno pudiendo, por ejemplo interactuar con el código fuente del panel de edición y con otros paneles para recoger y mostrar información al desarrollador. Todo el mundo puede contribuir mediante plug-ins. Los plug-ins creados por cualquier usuario puedan ser extensibles. Sin embargo, para determinadas operaciones la información es escasa y compleja y se hace difícil la creación de un interfaz adecuado. Es posible la

adaptación a cualquier otro lenguaje de programación e incluso a tareas que no tienen que ver estrictamente con programación.

La interfaz está organizada mediante vistas y perspectivas. Esto permite que cambie según la tarea que estemos realizando y así pueden aparecer y desaparecer vistas en perspectivas diferentes.

Como en los demás entornos las posibilidades de trabajo en equipo se limitan a la conexión con sistemas CVS que permiten mantener una copia del proyecto común para gestionar la realización de cambios de forma concurrente.

En este entorno no se realiza una compilación como tal sino que existe un agente que va compilando el código a medida que se va escribiendo y también va señalando los errores encontrados, esto permite una rápida corrección de errores relacionados con la declaración y utilización de variables, errores de sintaxis. Cuando el código fuente se guarda los errores existentes en ese momento se guardan en un panel de problemas para su posterior corrección.

3.3.5 Conclusiones respecto a los entornos de desarrollo comerciales

La **gestión de errores de los entornos de desarrollo es bastante pobre**. Todos excepto Eclipse muestran simplemente la lista de mensajes de error tras la compilación y esta lista es temporal, desaparece en cuanto pasamos a realizar otra tarea. Eclipse dispone de un panel de errores que guarda todos los errores pendientes de solución; y se puede conservar de una sesión de trabajo a otra. Ningún entorno permite mantener una "historia de compilación" o histórico de errores y realizar análisis estadísticos sobre él.

Prevención de errores. Todos los entornos analizados disponen de agentes que exploran el código mientras el programador lo escribe y señalan los errores encontrados, esto ayuda a prevenir errores léxicos o sintácticos; pero **no evita errores conceptuales**.

Ayuda para solucionar errores. La información de ayuda asociada a los mensajes de error es mínima y el programador tiene que recurrir a su experiencia para solucionar estos errores, cuando el programador es inexperto tiene que invertir mucho tiempo para encontrar la verdadera causa del error y solucionarlo.

3.4 Entornos de colaboración para el aprendizaje y desarrollo de la programación

3.4.1 Sistemas Colaborativos: CSCW y CSCL

Como afirma Enrique Rubio [Rubio 2003] dada la naturaleza social del conocimiento, la cooperación es uno de los pilares básicos de la sociedad del conocimiento.

Para permitir la cooperación entre varios usuarios a través de ordenador en la realización de un trabajo para alcanzar determinado objetivo común, los sistemas CSCW disponen de varios subsistemas: subsistema de comunicación, subsistema de colaboración, subsistema de coordinación.

Los sistemas CSCL (*Computer Supported Collaborative Learning*) son un subconjunto de los CSCW y donde la tarea a desarrollar es el aprendizaje del estudiante, basándose en un enfoque constructivista, es decir los estudiantes colaboran entre ellos para conseguir alcanzar los objetivos de aprendizaje propuestos. Estos sistemas incorporan características colaborativas que pueden ser aplicadas no sólo a entornos de aprendizaje sino también a

sistemas que permiten realizar las tareas correspondientes en un determinado ámbito de aplicación.

3.4.2 Sistemas para el aprendizaje colaborativo

Beatriz Barros define aprendizaje colaborativo como el proceso en el que los alumnos aprenden mientras proponen y comparten ideas para resolver una tarea, favoreciéndose con el diálogo la reflexión sobre las propuestas propias y las de los compañeros [Barros 2001].

Koschmann establece los siguientes principios para el paradigma CSCL [Koschmann 1996]:

- Construcción conjunta de la solución de un problema siguiendo algún método de colaboración.
- Coordinación de los miembros del grupo para organizar su trabajo.
- Semi-estructuración de los mecanismos que soportan la discusión.
- Interés tanto en el proceso de aprendizaje como en el resultado. Es fundamental proporcionar herramientas para que los estudiantes reflexionen sobre cómo llegaron al resultado final [Brown 1983]. En el enfoque colaborativo es objeto de interés tanto la solución como el proceso que permite al grupo llegar a ella. Aspectos significativos del proceso, pueden ser representados explícitamente, dando lugar a una reificación [Boder 1992].

Funcionalidades de los sistemas CSCL [Collis 1997]:

- Mediación en el intercambio de información.
- Mecanismos de ayuda a la toma de decisiones.
- Facilitar la comunicación en relación a las tareas a realizar.
- Gestionar el conocimiento compartido que se genera a lo largo de las tareas.

Para lograr estas funcionalidades los sistemas CSCL basan su funcionamiento en el concepto de *espacio de trabajo*, zona de recursos con una funcionalidad específica que permite la realización de una tarea. Un espacio tiene una *estructura*, ofrece al usuario una serie de *recursos* que le permiten interaccionar con él, una *funcionalidad* específica con privilegios asociados a roles y presenta al usuario diferentes *vistas* de los objetos que contiene.

- Espacio de tarea, asociado a una tarea, ofrece espacio y recursos para que el grupo la realice. Puede ser para tareas individuales o en grupo. Vistas del espacio de tarea:
- Subespacio de elaboración, vista del proceso. Grafo conversacional.
- Subespacio de resultado
- Subespacio de versiones
- Espacio de coordinación, asociado a una actividad, comunicación y coordinación durante la realización de las tareas. Funciones: coordinación y planificación; notificación de lo que ocurre en el sistema. Vistas:
- Tablón de mensajes
- Agenda de grupo

- Tablón de anuncios

En su tesis doctoral Beatriz Barros desarrolla un entorno genérico CSCL [Barros 1999]. Este sistema denominado DEGREE tiene las siguientes características.

Principios en el diseño del sistema DEGREE:

- Construcción en grupo de la solución de un problema.
- Realización de la tarea mediante discusión estructurada.
- Interés tanto en el proceso como en el resultado.
- Coordinación de los miembros del grupo.
- Acceso a la información.

Tareas estructuradas en subtareas. Esto permite guiar a los usuarios y representar el proceso de discusión en forma de *árbol de subtareas*. La comunicación entre los usuarios se representa en forma de grafo de tipos de contribuciones relacionadas, *grafo conversacional*.

Usuarios pueden tener *roles*. Roles definen operaciones a las que un usuario tiene acceso en los diferentes espacios de trabajo.

Información sobre una actividad se puede incluir en una *estructura organizativa de experiencias colaborativas* [Verdejo 1998].

Subsistemas de DEGREE:

- Subsistema Configurador de Experiencias.
- Subsistema Manejador de Experiencias.
- Subsistema Analizador.
- Memoria Organizativa de Experiencias.

El ámbito para el que están pensados el sistema DEGREE y la mayoría de los CSCL es el de **análisis y construcción de documentos** [Barros 2001]; que **se aleja bastante de la construcción de programas**. La colaboración se basa en discusiones sobre fragmentos de texto que luego se combinan para construir el documento final; pero esto está muy alejado de los procesos de construcción de programas en los que hay que respetar una sintaxis rígida, ni pasar un proceso de compilación y ejecución.

3.4.3 Programación colaborativa

En el ámbito profesional el desarrollo de proyectos medianos y grandes se realiza en equipo donde varios desarrolladores colaboran para realizar todas las tareas necesarias en el proyecto, desde el análisis y el diseño preliminar, la división en módulos, la implementación de estos módulos y la integración y puesta en marcha de la aplicación.

En el ámbito exclusivo de la programación una técnica cada vez más utilizada, donde colaboran dos programadores de forma síncrona es la programación en parejas. La programación en parejas es una técnica que se utiliza en Programación Extrema y que consisten en que dos personas codifican con un teclado, un ratón y un ordenador [Beck 2000] [Williams 2000]. Existen experiencias de las mejoras que aporta esta técnica a la hora de desarrollar software. Es una técnica que tiene aplicación práctica en la industria. Se emplea en proyectos de software reales y permite producir software de alta calidad. La programación en parejas está pensada para realizarse *cara a cara* como indica la definición;

pero existen artículos que estudian la posibilidad de realizar programación en parejas de forma distribuida [Scotts 2003].

Hay estudios [Nosek 1998] [Williams 2001] que avalan no sólo la viabilidad sino las ventajas de la programación colaborativa (los estudios se centran en la programación por parejas) que puede acelerar el proceso de programación y prueba; pero sobre todo permite desarrollar productos software con una más alta calidad que la programación individual. Estos estudios se basan en que la programación en parejas disminuye el número de defectos de los programas [Williams 2001] ya que se someten a un proceso de revisión continua.

Además, como efecto lateral, según Johnson [Johnson 1998], el proceso de analizar y criticar artefactos de software producidos por otras personas es un potente método para aprender o mejorar las habilidades para trabajar con lenguajes de programación, técnicas de diseño y dominios de aplicación. En la misma línea están las conclusiones de Zeller [Zeller 2000] con su gestor donde los estudiantes realizan revisiones de prácticas de sus compañeros de forma asíncrona.

3.4.4 Sistemas colaborativos aplicados al desarrollo de software

RECIPE

Dentro de los sistemas colaborativos de programación cabe destacar el sistema RECIPE (*Real-time Collabortive Interactive Programming Environment*) [Shen 2000] es un sistema que permite a programadores distribuidos geográficamente participar concurrentemente en el diseño, codificación, prueba, depuración y documentación de un mismo programa.

Razones para utilizar programación colaborativa distribuida:

- Programadores dispersos geográficamente que están trabajando en el mismo proyecto. Siempre es más efectivo trabajar colaborativamente que por separado.
- Desarrollar productos software de alta calidad en poco tiempo. Dos programadores trabajando colaborativamente pueden superar a programadores individuales trabajando individualmente [Nosek 1998].
- Permiten entrenar a los clientes o diagnosticar y resolver problemas del software ya implantado. Para proporcionar soporte a los clientes finales con el producto ya implantado resulta muy eficiente para los ingenieros de software y los administradores poder trabajar remota y colaborativamente con los clientes en los sistemas donde ya está funcionando el sistema final.

Shen propone un sistema para la programación colorativa en tiempo real basado en Internet. En realidad el sistema es un middleware que permite enviar datos de entrada desde varios puestos a un software en modo texto convencional y distribuir la salida a todos esos puestos para que todos los usuarios puedan ver las acciones de todos.

La arquitectura del sistema está centralizada en una máquina que contendrá los archivos del proyecto que estamos desarrollando y las herramientas de compilación (por ejemplo javac), depuración (gdb) y un shell de UNIX para ejecución y prueba del software desarrollado; además el sistema dispone de un editor colaborativo denominado REDUCE que permite crear y modificar archivos de código fuente entre varios desarrolladores. El servidor de RECIPE permite la gestión de sesiones en las que un desarrollador comienza una nueva tarea con una de las herramientas disponibles en el sistema y a la que se pueden unir otros usuarios bajo determinados permisos. Por otro lado los "sitios" RECIPE envían las entradas del usuario que está utilizando ese "sitio" al servidor y reciben la salida que es

distribuida, cuando se produce cualquier cambio, a todos los "sitios" RECIPE que están en esa sesión.

Las características destacadas del sistema son:

- Transparencia en la colaboración. Las herramientas que utiliza el sistema son las estándar pensadas sólo para un usuario; es el sistema el que posibilita la interacción colaborativa.
- Control de acceso flexible. A cada conexión que se realiza se le asignan unos determinados permisos: ver y participar, sólo ver o conexión rechazada.
- Extensibilidad. El sistema se podría utilizar con otros compiladores o depuradores.
- Colaboración jerárquica. Existen diferentes tipos de sesiones: sesión de shell de UNIX (UXSH), sesión de edición (EDIT), sesión de compilación (COMP) y sesión de depuración (DEBG).

Pero tiene algunas carencias, que son:

- No registra el trabajo de los programadores para obtener conclusiones.
- No ofrece herramientas especializadas de comunicación y *awareness*.

College

Uno de los grupos de investigación que está trabajando en sistemas de aprendizaje específicos para la programación es el grupo CHICO (Universidad de Castilla la Mancha). Este grupo ha desarrollado la herramienta COLLEGE [Bravo 2004] que permite trabajo colaborativo en problemas de programación complejos. Esta herramienta permite aplicar el ciclo de codificación-revisión, compilación y ejecución.

Este sistema hace un modelado del proceso de programación colaborativa basándose en Protocolos de Colaboración [Wessner 1999] para estructurar el proceso de aprendizaje. Existen una serie de espacios de trabajo en el protocolo de colaboración que son utilizados en el proceso síncrono de resolución de un problema. Estos espacios de trabajo son los siguientes:

- Edición / Revisión, en este espacio un único usuario del grupo edita, y el resto de los usuarios revisan síncronamente el código del que escribe, haciendole sugerencias de mejora. Además, el sistema proporciona, estando en este espacio, un mecanismo para cambiar el usuario que escribe el código.
- Compilación, el sistema compila el programa resultante de la edición y muestra los errores de compilación a todos los usuarios del grupo que se ponen de acuerdo en como solucionar estos errores.
- Ejecución. Permite a los usuarios ponerse de acuerdo para introducir los datos de prueba del programa.

Proporciona un mecanismo de navegación mediante *proposición* para que el grupo se ponga de acuerdo en cambiar entre espacios de trabajo.

La comunicación se establece mediante un chat estructurado flexible, proporciona tipos de mensajes predefinidos para determinadas tareas del dominio de la programación, como por ejemplo: "Creo que hay un error en la línea núm: …".

El sistema pone especial atención en las herramientas de awareness proporcionando: panel de sesión, puntero de edición; listas de interacción, área de propuesta de cambio de espacio de trabajo, autor de cada mensaje.

Sin embargo, este sistema a pesar de ser una gran herramienta colaborativa y conseguir los objetivos que se propone, adolece de una serie de puntos débiles que a continuación se detallan:

- Sólo permite realizar colaboración síncrona, lo que obliga a que los usuarios estén presentes simultáneamente en la misma sesión de trabajo, lo cual obliga a una coincidencia de horarios que no siempre es posible. Esta situación se puede ver agravada si los usuarios pertenecen a países con husos horarios distantes.
- No se desarrolla utilización en empresas, Se plantea como "un sistema de Programación Colaborativa que podría utilizarse tanto como sistema de CSCW en las empresas, como sistema de CSCL para el aprendizaje en centros de enseñanza" pero no se desarrolla el uso en empresas.
- Guía en el proceso. Se plantea que haya una guía en el proceso de programación basándose en "ideas de los Sistemas Tutores Inteligentes" pero no se desarrolla.
- Estadísticas de errores, los autores consideran adecuado incorporar una función que permita consultar la estadística de errores para que los propios alumnos puedan comprobar cuáles suelen cometer, para prestar más atención a los aspectos relacionados con esos errores y poder discutir al respecto. Esta idea es correcta; pero se pueden complementar con métricas individuales de errores que informen al usuario de sus errores más frecuentes para que así el usuario pueda evitarlos.
- Plantean un análisis del trabajo efectuado por los alumnos, se pretende registrar: el desplazamiento a través de los espacios de trabajo, la comunicación y el mecanismo de proposición. Además, de estas propuestas sería interesante un mecanismo que guardase automáticamente en una base de conocimientos los pasos para la corrección de un error.

En general, uno de los problemas que presenta la mayoría de los sistemas CSCL es que adolecen de una adaptación al ámbito de la programación ya que los subsistemas de comunicación son genéricos y no están orientados a simplificar el trabajo de los desarrolladores. Se proporcionan herramientas genéricas: correo electrónico y chat para la comunicación entre usuarios en vez de proporcionar herramientas más específicas que proporcionen una mayor eficiencia en el trabajo para este dominio. Por ejemplo, se podría partir de los errores para plantear como se puede solucionar cada uno de ellos.

PlanEdit

PlanEdit [Redondo 2002] se puede utilizar para construir una solución abstracta a cualquier problema mediante el diseño colaborativo.

Para manejar problemas de programación cada instrucción se considera como un objeto y un programa como un conjunto de instrucciones relacionadas de acuerdo al flujo de control. De esta forma utilizamos manipulación directa para construir un programa mediante objetos instrucción.

PlanEdit dispone de varios espacios de trabajo: editor de planes, espacio individual en el que cada usuario desarrolla su diseño del programa; discusión y argumentación, organiza y presenta todas las contribuciones de los usuarios, ya sean propuestas de diseño,

comentarios, preguntas, etc.; resultados, presenta la solución final consensuada por el grupo.

3.4.5 Conclusiones sobre la colaboración en el aprendizaje de la programación

Existen sistemas que dan soporte al trabajo colaborativo de dos programadores, están pensados para **realizar el trabajo de forma síncrona**, los dos usuarios deben estar en sesión al mismo tiempo. Esta forma de trabajo está orientada a la **revisión continua** de código, mientras que uno de los dos programadores escribe, el otro revisa que lo que está haciendo es correcto y es coherente con el diseño global de la aplicación, esto si se realiza correctamente puede ayudar a mejorar la calidad del código realizado. El planteamiento también permite solucionar errores de compilación o ejecución sobre la marcha, ya que la colaboración de varios programadores permite una formulación más completa de las hipótesis del problema a través de los indicios de los errores de compilación y de las pruebas y trabajando sobre esto se podrá solucionar el problema causante del error. Sin embargo, esta forma de trabajo síncrona **no permite que el conocimiento aportado por un usuario en un momento dado sirva de información para otros equipos** o para el mismo equipo en otro momento del desarrollo.

3.5 Entornos profesionales de gestión de proyectos software

3.5.1 Sistemas de gestión de proyectos software

En este apartado nos salimos del ámbito docente para estudiar varios sistemas pensados para el mundo profesional. Estos sistemas como se dijo en un anterior (1.2 Entornos de desarrollo para el aprendizaje de la programación) apartado tienen una utilizan conceptos que pueden ser difíciles de manejar para programadores principiantes y por ello no es conveniente incorporarlos directamente; pero poseen algunas características que una vez adaptadas puede ser interesante incorporarlas en un sistema de aprendizaje que no sólo necesite manejar ejemplos de "juguete" sino que necesite también manejar proyectos con muchos archivos.

La necesidad de entornos a nivel profesional para la gestión de proyectos software es evidente debido a la cantidad y complejidad de las tareas que es necesario llevar a cabo y por ello existen varios entornos que tratan de solucionar la problemática que surge al enfrentarse a un proyecto.

Hemos realizado un estudio de los cuatro entornos profesionales para el desarrollo de software más utilizados actualmente.

PHPProjekt

Es una aplicación Web con licencia GPL de trabajo en grupo implementada en PHP para facilitar su uso. Sus características más destacadas son:

- Sistema modular para facilitar su ampliación. Esto permite el desarrollo o mejora de forma independiente de herramientas para incorporarlas luego al sistema.
- Diferentes niveles de privilegios. Grupos opcionales. Con esta aplicación se pueden gestionar varios proyectos y subproyectos a la vez y se pueden llevar a cabo proyectos en grupos grandes, por tanto es necesario una autenticación del usuario y una asignación de permisos a cada usuario.

- Soporte de múltiples idiomas. Lo cual permite una utilización más fácil por parte de usuarios de distintos idiomas.
- Calendario para usuarios y grupos, eventos, reserva de recursos, etc. Permite notificar y coordinar los hitos y las reuniones en un proyecto.
- Gestión de contactos. Facilita la tarea de comunicación entre diferentes miembros de un proyecto.
- Time card. Es una característica bastante original de este producto, se usa para registrar el trabajo de cada miembro en el proyecto y permite la realización de estadísticas sobre este tiempo. Para ello, el usuario va introduciendo las horas de trabajo que dedica cada día, pudiendo incluir observaciones sobre lo que hizo en cada intervalo de tiempo.
- Gestión básica de proyectos. Definición de tareas y subtareas con profundidad ilimitada y visualización del estado actual del proyecto.
- Chat con miembros online, para la comunicación síncrona. Sistema de foros, para la comunicación asíncrona off-line. Cliente de email integrado, que permite la comunicación dentro y fuera del proyecto.
- Gestión de archivos. Los usuarios pueden subir y bajar archivos de un espacio compartido, con posibilidad de restringir el acceso al archivo Además permite clasificar los archivos en diferentes categorías.
- Sistema de encuestas / votaciones. Para ayudar a la toma de decisiones.

PHPProyekt, está pensado para albergar todos los proyectos que se realicen en una organización en un servidor común dentro de una Intranet o incluso entre organizaciones a través de Internet. Está planteado para gestionar proyectos en general y no sólo proyectos software por lo que carece de herramientas específicas para el tratamiento del código fuente o de productos software

SourceForge

Es un Portal Web para la creación de proyectos software de código abierto. Actualmente funciona como un portal centralizado especializado sobre Internet. Sin embargo, es posible hacer una instalación propia dentro de una Intranet para gestionar los proyectos software de una organización.

Describimos a continuación las características de este portal:

- Permite la autenticación de usuarios y la asignación de permisos.
- Dispone de servidores de descarga, que permiten dar servicio a los usuarios finales que van a utilizar el producto.
- Herramientas avanzadas para la búsqueda de determinados proyectos o tipos de proyectos (por categoría, descripción, nombre....). Al ser un portal que alberga infinidad de proyectos, es útil poder ver que proyectos se están desarrollando sobre un determinado tema. Es decir, utilizarlo como un repositorio de software libre.
- Foros de discusión.
- Monitorización del proyecto. Permite tener una vista completa de cualquier cosa que ocurra en el proyecto. Si se produce una respuesta, una entrada en el foro....se sabrá acerca de ello por medio de e-mails enviados por el sistema.

- "Granja" de compiladores, que permite compilar los códigos fuente del proyecto con distintos compiladores, muy útil para desarrollar proyectos multiplataforma.
- Herramientas de comunicación y coordinación. Grupos de noticias, repositorio de archivos, seguimiento de errores, tareas pendientes, listas de correo.

SourceForge está pensado para organizar grupos muy grandes de desarrolladores que carecen de una cohesión previa por tanto hay muchas herramientas de portal para dar servicio a desarrolladores voluntarios que quieren integrarse en un proyecto de un determinado tipo para participar en su desarrollo, desarrolladores que quieren seguir el desarrollo de determinados proyecto o usuarios de aplicaciones que simplemente quieren buscar, descargar e instalar el producto que se adapte mejor a sus posibilidades. Es un sistema que está especializado en el desarrollo de proyectos software y por tanto dispone de herramientas especializadas facilitando el trabajo a los desarrolladores

Software Libre.org

Es un Portal Web para el desarrollo de software de código abierto. La filosofía es la misma que SourceForge, un portal que recoja gran cantidad de proyectos software realizados por una gran comunidad de desarrolladores.

Resumimos sus características más destacadas:

- Una página para acceso a la documentación del proyecto. Permite almacenar y facilitar la consulta de los distintos manuales tanto de desarrollo como de usuario del producto software.
- Herramientas de comunicación. Foros de discusión y listas de correo.
- Gestor de tareas pendientes. Permite crear determinadas tareas que se asignan a desarrolladores y en cualquier momento podemos ver una lista de las tareas que hay pendientes en el sistema. El desarrollador debe ir introduciendo información acerca de cómo evoluciona la tarea (%), esto facilita el seguimiento del proyecto.
- Repositorio de archivos controlado por CVS.
- Espacio ftp, permite distribuir versiones del producto terminado.
- "Recortes de código". La función de esta característica es proporcionar la forma de compartir fragmentos de código para reutilizarlos. Se puede crear un nuevo "recorte", y luego publicar versiones adicionales del recorte de forma fácil y rápida. Una vez que se tengan recortes publicados, se puede publicar un "paquete" de recortes. Este paquete puede contener múltiples y específicas versiones de otros recortes. También se proporcionan herramientas que facilitan la búsqueda de "recortes".

CollabNet

CollabNet Enterprise Edition es una aplicación de escritorio para el desarrollo colaborativo de software.

Las características más destacadas de este sistema son:

- El espacio de trabajo del proyecto se divide en dos partes: administración del proyecto e informes del proyecto.
- Sistema de control de versiones por medio de CVS e incluyendo en cada nueva versión las diferencias con la versión anterior.

- Foros de discusión. Sistema de mensajes de correo electrónico. Sistema de anuncios.
- Administración de documentos y archivos
- Permisos basados en roles(no todos los usuarios pueden hacer lo mismo)

3.5.2 Conclusiones sobre los entornos profesionales de gestión de proyectos software

La gran mayoría de los sistemas están orientados a la gestión de alto nivel en el proyecto: división de tareas, planificación de tiempos, control sobre el grado cumplimiento de la planificación, distribución de las diferentes versiones del proyecto acabado, foros y listas de discusión a alto nivel sobre el proyecto, realmente esto es aplicable no sólo para proyectos sino a cualquier proyecto de ingeniería.

Los que están especializados en gestión de proyectos software incluyen un espacio compartido para almacenamiento del código y de la documentación del proyecto y foros de discusión más especializados sobre requisitos del software y defectos del producto.

Tabla 2. Tabla en la que comparamos las características de las herramientas colaborativas para el desarrollo de software

	PhpProjeckt	Source Forge	Software Libre	CollabNet
Foros / Votaciones	X	X	X	X
Repositorio	-	X	X	X
Espacio compartido	X	X	X	X
Comunicación síncrona	X	-	-	-
Comunicación asíncrona	X	X	X	X
Edición de código on-line	-	-	-	-
Revisiones de código	-	X	-	-
Recortes de código	-	-	X	-
Compilador integrado	-	X	-	-
Awareness	-	X	-	-
Diferentes perfiles de usuarios	X	X	X	X
Tareas pendientes	X	X	X	-
Monitorización de proyectos	X	X	-	-
Time Card	X	-	-	-
Calendario	X	-	-	X
Multilingüe	X	-	-	-
Búsqueda de proyectos o grupos	X	X	X	-
Interfaz de usuario intuitiva	X	-	X	-

Pocas herramientas se fijan en el proceso de escritura y revisión del código de las distintas partes del proyecto, proceso que comentábamos anteriormente fundamental para el éxito del proyecto y en concreto para el programador, que tendrá que utilizar sus conocimientos, habilidades y buen hacer con poca o ninguna ayuda de las herramientas. Las herramientas analizadas disponen de sistemas de control de versiones que permiten almacenar una versión común del proyecto para que varios desarrolladores puedan trabajar en él. Sin embargo, la escritura de código siempre es local y asíncrona, no hay herramientas de ayuda directa mientras el desarrollador está escribiendo la parte que se le ha asignado de la aplicación.

3.6 Gestores de prácticas avanzados

3.6.1 Sistemas de gestión de prácticas de programación

Un planteamiento que está adquiriendo gran relevancia en los últimos años son los "gestores de prácticas" que han evolucionado desde el campo de la gestión pura para implementar técnicas de enseñanza de la programación. Son sistemas que en un principio tenían como misión automatizar la recogida y gestión de las prácticas de una asignatura y que posteriormente fueron adquiriendo mayor funcionalidad para ayudar al profesor en la corrección-revisión de las prácticas. Estos sistemas se han mostrado útiles [Dawson-Howe, 1996] [Zeller, 2000] en la rápida mejora de la calidad del código escrito por los estudiantes y en la detección por parte del profesor de problemas conceptuales de entendimiento de estos [Huizinga 2001].

Sistema de entrega y comprobación automática de prácticas

Dawson-Howe [Dawson-Howe 1996] presenta un sistema para la evaluación automática de programas realizados por estudiantes mediante archivos de prueba, que reduce el trabajo de los profesores, sin tener que pedir a los estudiantes que ejecuten ellos mismos la prueba y envíen el resultado al profesor, lo cual podría dar lugar a falsificaciones.

Los estudiantes utilizan un programa que se encarga del control de la entrega del estudiante, realizando el proceso por pasos:

- En primer lugar, comprueba que el estudiante cumple los plazos establecidos.
- Se compilan los programas y se capturan los resultados de la compilación.
- Se ejecuta el programa del estudiante y registra los resultados de salida, es el propio estudiante el que introduce los datos de entrada que el programa va pidiendo.
- Por último, el estudiante confirma si está de acuerdo con la ejecución ya que puede ver los resultados y ratifica el envío de este resultado, junto con el código fuente y la documentación, si existe, al sistema central.
- El sistema central envía una confirmación de recepción al estudiante y registra la recepción en la base de datos.

Las ampliaciones que se planteaban para el sistema eran: mejorar la indentación automáticamente para facilitar la corrección al profesor, e introducir una realimentación al estudiante por una parte en la información en la confirmación de la recepción de cada envío y por otra realizando una evaluación simple pero automática del código.

Sistema de entrega y evaluación de objetivos de aprendizaje

Huizinga [Huizinga 2001] propone un sistema que se centra en la evaluación de los estudiantes para identificar las deficiencias en el aprendizaje y que el profesor pueda subsanarlas. Para realizar esto, Huizinga propone definir una serie de objetivos de cada proyecto relacionados jerárquicamente con los objetivos de la materia a enseñar, estos objetivos se dividen en conceptos teóricos y habilidades técnicas. Para realizar la evaluación del cumplimiento de estos objetivos utiliza una herramienta de entrega automática de prácticas. La herramienta compila y ejecuta una serie de test predefinidos y genera un informe de registro recogiendo la información sobre los errores de compilación y ejecución.

El programa elabora una serie de datos estadísticos sobre el comportamiento del conjunto de los alumnos al presentar las prácticas. El análisis de estos datos ayuda a evaluar que objetivos de concepto o habilidad necesitan más atención. Ejemplos de estos datos son y el problema de aprendizaje reflejado son: número medio de reenvíos por estudiante puede significar una especificación confusa del proyecto, objetivos conceptuales del proyecto poco dominados; porcentaje de envíos finales con errores de compilación refleja deficiencias en la sintaxis del lenguaje; porcentaje de envíos finales con errores de ejecución puede reflejar una incapacidad para manipular las estructuras de datos.

La evaluación de la herramienta ha sido positiva, sin embargo uno de los aspectos más criticados del sistema es la **generación de informes tras la presentación de una práctica**; esta autoevaluación combinada con la posibilidad de volver a enviar la práctica debería ayudar a alcanzar los objetivos de aprendizaje. Sin embargo, la mayoría de los estudiantes opinan que no es una ayuda. Uno de los problemas detectados es que gran número de **estudiantes no leen el informe**, otro gran número consideran que la información proporcionada **no es muy reveladora de los verdaderos problemas** que puede tener su programa.

Praktomat

Praktomat [Zeller 2000], es un sistema que permite a los estudiantes leer, revisar y evaluar programas de otros compañeros para mejorar la calidad y el estilo. Los alumnos envían sus prácticas y pueden recuperar prácticas enviadas por algún compañero para revisarlas y corregirlas (esta asignación se realiza por el sistema automáticamente). Una vez que una práctica esté revisada el autor pueden obtener las revisiones y volver a enviar el programa mejorado.

Sobre cada programa el sistema realiza una primera prueba automática en la que el sistema compila y realiza pruebas sobre cada programa. Parte de los casos de prueba son públicos para los estudiantes; pero hay otro conjunto de casos de prueba que son confidenciales. Por otra parte el sistema facilita las revisiones cruzadas entre estudiantes estableciendo un sistema automático de revisión ciega en la que un alumno revisor pide al sistema una práctica para revisar; pero no conoce el autor de esta.

El artículo justifica que este proceso de revisión fomente el aprendizaje y la mejora del estilo y por tanto la calidad del código de los revisores y revisados mejora en sucesivas prácticas.

3.6.2 Conclusiones sobre los gestores de prácticas avanzados

Hemos revisado algunas herramientas que permiten recoger prácticas y hacer revisiones y devolverlas a los alumnos. Estas herramientas, estaban pensadas en un principio,

simplemente para automatizar la recogida de prácticas por parte del profesor y el envío de comentarios a los estudiantes.

Posteriormente la mayoría de los sistemas y así lo hacen los dos primeros que se han estudiado, tratan de descargar de trabajo al profesor realizando una compilación automática para verificar que no hay errores de compilación y una ejecución con unos datos de prueba bien introducidos por el propio alumno o creados a priori por el profesor para realizar las pruebas. De esta forma el profesor ya tiene algunos datos en los que basar la evaluación que debe realizar sobre el programa.

Un elemento importante para lograr el aprendizaje del alumno y la mejora de los programas que construye es la realimentación mediante el informe que genera el sistema a partir de la compilación y pruebas que realiza. Sin embargo, este informe es muchas veces bastante pobre y sólo se refiere a si hay errores de compilación y cuales son y si las pruebas proporcionaron resultados correctos o no. Realmente esta información la puede obtener el alumno compilando su práctica y haciendo unas pruebas medianamente sistemáticas. No se aporta información extra sobre errores de compilación y el problema real que los está causando, tampoco se aporta información sobre los problemas en las pruebas y en definitiva el alumno dispone de poca información para solucionar los defectos del programa que ha realizado, si este es medianamente grande el problema si no puede consultar a un profesor puede convertirse en difícil de resolver.

Para conseguir descargar de trabajo al profesor e intentar mejorar la riqueza de los informes emitidos el sistema Praktomat [Zeller 2000] utiliza una revisión de pares entre compañeros. Para que este sistema tenga éxito conlleva que se cumplan dos requisitos: se debe entrenar a los alumnos en la evaluación de prácticas y la realización de comentarios y cada alumno debe realizar un trabajo personalizado diferente al de los demás para evitar el problema de las copias.

Todos estos sistemas utilizan un **protocolo petición-respuesta**: el alumno envía su práctica, el profesor (o su compañero) evalúa la práctica y devuelve la corrección (nota, comentarios, etc.). Algunos sistemas si permiten varias entregas; pero no tienen previsto correcciones intermedias, correcciones de un supervisor antes de que el estudiante/desarrollador haya finalizado la tarea asignada. Esto puede provocar que **el sistema sólo ayude a corregir errores**; pero **no a evitarlos** y peor todavía, **no permite un seguimiento del proceso de desarrollo que ha seguido el alumno** y por tanto descubrir problemas en este proceso que pueden derivar en un código de baja calidad.

3.7 Otros planteamientos en el aprendizaje de la programación

Estudiamos en este apartado, otros planteamientos para abordar el aprendizaje de la programación [Gomez 2003].

3.7.1 Visualizaciones gráficas de programas

Existen muchas propuestas en las que se utilizan representaciones gráficas, muchas veces simplificadas, para que los estudiantes puedan comprender determinados conceptos relacionados con la programación, más concretamente con la ejecución del algoritmos y estructuras de datos relacionadas.

Hay varios tipos de visualizaciones gráficas dependiendo de las posibilidades de configuración y personalización y de la interacción con el usuario.

Existen visualizaciones predefinidas de algoritmos en las que simplemente se puede ver un video o animación sobre una determinada operación [Stasko 1998].

Sistemas de visualización y animación, que permiten definir el algoritmo o estructura de datos que queremos simular. Existen gran cantidad de sistemas de este tipo entre los que destacamos:

- Animal (A New Interactive Modeler for Animations in Lectures) [Rößling 2000c] [Rößling 2001] [Rößling 2000b]
- LEONARDO animación de programas en C [LEO] [Crescenzi 2000] [Demetrescu 2000]
- XTANGO y POLKA [XTA] [Stasko 1990]
- JHAVÉ (Java-Hosted Algorithm Visualization Environment), [JHA] visualización de algoritmos en tres lenguajes de script [Naps 2000].

Sistemas que muestran el efecto que tiene la ejecución de sentencias básicas de un lenguaje sobre los elementos del ordenador. Actúan como intérpretes de las sentencias visualizando el estado de la memoria, el procesador y otros recursos.

Hypertextbooks, proyectos en los que se recogen tanto texto como visualizaciones. Un ejemplo de esto es el trabajo de Boroni [Boroni 2001].

DSTool

Si nos fijamos en lenguaje Java, todos los entornos de desarrollo actuales incluyen capacidades de "visualización de la ejecución", en forma textual, para facilitar el seguimiento y la depuración del programa en ejecución. Esto es relativamente sencillo de hacer ya que Java incorpora introspección como parte inherente del lenguaje, lo cual permite identificar las instancias de las clases en tiempo de ejecución, sus propiedades y los valores de estas. Más aún, muchos de los entornos también permiten modificar los valores de las instancias visualizadas. Un paso más allá, la herramienta DSTool [Sama, 2003] (citada ya en el apartado de 1.1 Sistemas de aprendizaje virtual de la programación), que permite visualizar de forma gráfica el estado de estructuras de datos básicas. No se trata de un entorno; sino de un *framework* que implementa el "modelo" de varias estructuras de datos básicas y que proporciona una "vista" para visualizarlas. Las estructuras son genéricas y pueden contener instancias de cualquier clase que el usuario desee crear en Java. La visualización se produce cuando desde el programa se invoca a un método predefinido. A partir de este momento el usuario puede, no sólo ver la estructura existente en memoria; sino también interactuar con ella, introduciendo nuevos elementos, eliminando otros o modificando los que quiera. Además, a partir de este *framework* se ha desarrollado una especie de tutorial en la línea de los *Hypertextbooks* citados anteriormente que combina una descripción textual y gráfica de varias estructuras de datos con la posibilidad de interactuar y experimentar con ellas a través de un programa pre-hecho que utiliza este *framework*.

Análisis de aspectos pedagógicos

Los aspectos pedagógicos de estos entornos se han analizado en varios trabajos. Algunos analizan experiencias en grupos de alumnos [Rößling 2000a]. Y otros proponen una serie de requisitos pedagógicos que deberían cumplir estos entornos [Hundhausen 2002] [Rößling 2002]:

- Ser sistemas de propósito general para que puedan integrase en diferentes aspectos.
- Permitir introducción de datos.

- Disponer de capacidad de volver hacia atrás en la animación.
- Disponer de predicción interactiva, para preguntar al alumno lo que va a ocurrir a continuación.
- Recoger la interacción de los alumnos en una base de datos, de manera que pueda ser utilizada por los profesores para detectar puntos en los que los alumnos tienen dificultad e incluso como parte de la evaluación que se haga de los alumnos.

3.7.2 Representación de mundos virtuales

En la representación de mundos virtuales, los alumnos observan un "mundo imaginario" en el que habitan seres cuyo comportamiento viene dictado por la ejecución de las instrucciones del programa.

Karel, The Robot [Pattis 1981] representa un robot habita en un mundo bidimensional muy simple. Se desarrollaron versiones para la enseñanza orientada a objetos [Bergin 97]. Está disponible una versión todavía más reciente con sintaxis Java pura [Bergin 03] [Bergin 00]. Otro autor ha realizado una extensión llamada JKarelRobot y ha realizado un estudio de su aplicación pedagógica a la enseñanza de la programación orientada a objetos [Buck 2001].

Alice [Stage3 2005], permite crear mundos virtuales en 3D que reflejan el estado del programa y van cambiando a medida que cambia dicho estado [Dann 2000]. Se utiliza para abordar una estrategia *objects-first* en asignaturas de introducción a la programación [Cooper 2003].

Con Robocode [ROB] tenemos que programar, en Java, unos robots que se enfrentan en un BattleGround. Cada Robot tendrá sensores y actuará en consecuencia. Los sensores detectan robots enemigos con lo que según los eventos podemos cambiar el comportamiento general del robot etc....

Una vez que se vaya controlando el tema, construimos equipos. Los robots de un equipo pueden mandarse mensajes para colaborar todos juntos en cierta tarea, etc.

3.7.3 Entornos que utilizan ejemplos

En estos entornos se plantea disponer de una biblioteca de soluciones a determinados problemas y utilizarla para buscar soluciones a nuevos problemas, ya que tanto programadores con experiencia como los noveles a menudo se sirven de ejemplos de programas para aprender nuevos planteamientos de código o hacer evoluciones que a partir de código antiguo cumpla con nuevo requerimientos.

Algunos trabajos giran en torno a la construcción de bases de ejemplos de programación y al desarrollo de mecanismos de acceso o selección (más o menos sofisticados) [Brusilovsky 1996], [Guzdial 1995]. El problema crucial es que los alumnos noveles carecen del conocimiento y la experiencia necesarios para encontrar ejemplos relevantes mediante herramientas como la selección a través de un menú [Neal 1989] o la búsqueda mediante palabras clave [Faries 1988]. Brusilovsky plantea varios enfoques de selección de ejemplos [Brusilovsky 1996]:

- Selección conducida por el estudiante, aplicando tecnología hipermedia para inspeccionar ejemplos. La dificultad radica en cómo estructurar los ejemplos en un hiperespacio y qué tipo de enlaces proporcionar para la navegación.
- Selección conducida por el sistema. Sistema dispone de conocimiento sobre los ejemplos y analizando el problema que se quiere abordar se sugieran ejemplos

relacionados. Una alternativa es que se disponga también de conocimiento sobre el alumno.

- Ambos agentes (sistema y alumno) intervienen. El sistema sugiere ejemplos apropiados y el alumno realiza la selección definitiva.

WebEx [Brusilovsky 2001], herramienta basada en Web que permite explorar de manera interactiva ejemplos de programas autoexplicativos.

Características:

- Ejemplos autoexplicativos, cada línea de código va acompañada de un comentario que aclara su significado en el programa.
- Hace frente al problema de la heterogeneidad de los alumnos: alumnos con diferente nivel inicial de conocimiento y distintas capacidades de adquisición. Diferentes velocidades, ejemplos y nivel de detalle.
- Alumnos eligen estrategia de exploración, la cantidad de comentarios asociados a los programas que están visibles.

Para facilitar la navegación y la localización de ejemplos planean estructurar la base de ejemplos siguiendo un enfoque conceptual. Se trata de identificar los conceptos clave de los contenidos ejemplificados en los programas y mantener enlaces bidireccionales entre los conceptos y los ejemplos. El profesor puede "monitorizar" la actividad de los alumnos y extraer conclusiones acerca de la forma en la que éstos trabajan con los ejemplos.

3.8 Técnicas de detección de errores

Analizaremos en este apartado varias técnicas para detectar errores en el código y poder corregirlos para obtener un código de mayor calidad.

3.8.1 Formas de encontrar y corregir defectos

Cuando se quieren buscar y eliminar los errores del código de una aplicación, se deben de seguir una serie de pasos que son los siguientes:

- Identificar los síntomas del defecto. Lógicamente lo primero es darse cuenta de que algo falla.
- Deducir de estos síntomas la localización del defecto. Hay que intentar aislar el código que está causando el error y esto lo tenemos que hacer a partir de sus síntomas.
- Entender lo que es erróneo en el programa. Una vez que tenemos aislado el error hay que comprender sus causas y situarlas en el modelo general del programa.
- Decidir cómo corregir el defecto. Hay que decidir cual es la mejor forma de eliminar el error, sin que se vean afectadas otras partes del programa.
- Hacer la corrección. Debemos hacer las modificaciones decididas.
- Verificar que el arreglo ha resuelto el problema. Debemos comprobar que se ha eliminado el error.

Herramientas y técnicas que ayudan al programador a detectar los defectos:

- Compiladores, su trabajo es generar código; por tanto, explorará el código fuente para ver si puede generar código. Normalmente, un compilador si puede generar código no mostrará errores.
- Análisis estático. Estas herramientas están especializadas en la búsqueda de errores sobre el código fuente. Hacen un análisis con mayor profundidad que los compiladores.
- Pruebas. Es necesario crear los casos de prueba. La calidad de las pruebas viene dada por el grado en el que los casos cubren todas las funciones importantes. Una vez realizadas, hay que comprobar los resultados.
- Prueba por parte de usuarios finales de versiones beta. Entregar programas a los usuarios finales para que ellos identifiquen e informen de los defectos.
- Inspecciones de código. La inspección es una técnica eficiente, ya que cuando se aplica se ven directamente los problemas y no los síntomas: cuando se revisa se piensa en lo que el programa debe hacer.

3.8.2 Inspección de código

Listas de comprobación para la inspección de código

La clave para realizar una inspección de código efectiva es tener un procedimiento de revisión eficiente. Cuando es esencial encontrar y corregir cada defecto en un programa, se debe seguir un procedimiento preciso y esto es lo que proporciona una lista de comprobación.

Características de la lista de comprobación:

- Esta lista debe adaptarse para buscar los defectos encontrados en programas anteriores.
- Comparar la lista de comprobación con la de otros ingenieros, puede sugerir aproximaciones útiles para la revisión.
- Encapsula la experiencia personal. Y debe ir mejorándose y adaptándose con el tiempo.

El principal peligro de las listas de comprobación es que generalmente se encuentra lo que se busca. Sin embargo, pueden aparecer problemas inesperados, por eso es buena idea hacer una revisión general del programa para buscar lo "inesperado".

Realización de una lista personal

Para que el proceso de revisión sea más efectivo la lista de revisión debe estar diseñada específicamente para cada programador basándose en los tipos de defectos que se encuentran. Para llevar a cabo esto, se puede hacer una lista ordenada por el número de defectos encontrados en cada fase en programas anteriores, descubrir los errores específicos para los tipos con mayor número de defectos, para los errores más importantes incluir los pasos para revisar que no se producen, agrupar las comprobaciones parecidas, realizar las revisiones y comprobar cuales son efectivas y cuales no; mantener las revisiones efectivas y cambiar las no efectivas. De esta forma, la lista de comprobación se convierte en un resumen de la experiencia personal. Cada cierto tiempo hay que reducir la lista de comprobación, eliminando las comprobaciones menos relevantes; ya que el poder de una lista de comprobación es que centre la atención.

Contabilización de defectos del código fuente

Humphrey propone en su libro "Introducción al Proceso de Software Personal" [Humphrey 1997] una contabilización de defectos en el software mediante un Cuaderno de Registro de Defectos. Esta contabilización tendría las siguientes características:

- Registrar los defectos corregidos sin tener en cuenta que cada uno pueda provocar varios mensajes de error de compilación
- Diferenciar entre defectos de diseño, de requisitos o de codificación.
- No considera defectos los que se corrigen sobre la marcha. Es decir, se contabilizarían los defectos al terminar una fase de una parte de un producto.
- En un principio es importante concentrase en los defectos que se encuentren durante la compilación y las pruebas.

Esta contabilización permite conocer los errores cometidos y su tipo. Se trata de estudiar los resultados para ver que errores son los que se cometen más frecuentemente y ver como evitarlos.

3.8.3 Revisión automática de código

Se han desarrollado muchas técnicas para encontrar errores de software automáticamente. Se basan en métodos formales que tienen mucho valor; pero son difíciles de aplicar, ya que no siempre encuentran errores reales.

Existen muchas investigaciones sobre técnicas para la detección automática de errores [Flanagan 2002] [Xie 2002] [Foster 2002] la mayoría basadas en técnicas formales pero no encuentran una forma de aplicación amplia. Existen otras técnicas simples de análisis estático para encontrar errores basándose en la noción de patrón de error. Un patrón de error es una expresión de código que podría ser un error. Las ocurrencias de los patrones de error son lugares donde el código no sigue la práctica correcta habitual en el uso del lenguaje o una biblioteca del mismo. Utilizando técnicas de análisis relativamente simples pueden implementarse detectores para muchos patrones de error.

3.8.4 Herramientas análisis estático de código

En este apartado se analizan las herramientas para el análisis de código fuente más extendidas [Rutar, 2004]. Haciendo una descripción inicial de su disponibilidad y el tipo de errores que detectan. El estudio de las herramientas de detección de errores se centrará en las herramientas que trabajen con el lenguaje Java.

Tipos de herramientas

- Un Bug checker, usa el análisis estático para encontrar código que viola una propiedad específica de corrección, y que puede causar un comportamiento incorrecto del programa en ejecución.
- Un Style checker, examina el código para determinar si contiene violaciones de reglas de estilo concretas.

El valor principal de hacer cumplir las reglas de estilo de código es que cuando se usa un estilo consistente a lo largo de todo el proyecto, es más fácil para los desarrolladores trabajar en el proyecto ya que pueden comprender el código de los demás. Algunas reglas de estilo como "no usar sentencias de asignación en la condición de un if" pueden ayudar a

prevenir ciertas clases de errores. Sin embargo, las violaciones de estas guías de estilo es poco probable que sean errores.

Otra forma de ver esto es que las violaciones de guías de estilo sólo causan problemas a los desarrolladores mientras que los avisos producidos por un bug checker pueden representar errores que causen problemas a los usuarios del software.

Los Bug checkers son más utilizados, causas de esto:

- Es necesario más trabajo para utilizar la salida de los Bug checkers. Normalmente los Style checkers son más precisos y sus avisos proporcionan información directa sobre lo que hay que cambiar para cumplir la guía de estilo. Los Bug checkers producen avisos más imprecisos o difíciles de interpretar. Además, para arreglar los errores proporcionados por un Bug checker se requiere comprender la causa del error.
- Los Bug checkers tienen un porcentaje de falsos avisos que tiende a incrementarse con el tiempo, a medida que los errores reales se corrigen.

Las herramientas basadas en patrones de error representan un buen punto en el diseño de Bug checkers por varias razones:

- Son fáciles de implementar
- Tienden a producir una salida que es fácil de entender para los programadores.
- Suelen ser bastante efectivos porque fijan el objetivo en encontrar desviaciones sobre las prácticas aceptadas.

Jlint

Jlint [Knizhnik 2003] es una herramienta que examina el código intermedio de Java (bytecode) buscando posibles errores, inconsistencias y problemas de sincronización de hilos haciendo un análisis del flujo y un análisis sintáctico mediante un *grafo sintáctico cerrado*. También busca interbloqueos cuando se utilizan varios hilos en un programa, construyendo un grafo de bloqueos y asegurando que nunca se forman ciclos en el grafo.

En realidad, la herramienta está compuesta por dos programas AntiC que realiza la verificación sintáctica y Jlint que efectúa la verificación semántica. AntiC es un programa que realiza un análisis léxico y sintáctico de archivos fuente de C, C++ o Java. Debido a la similitud de la sintaxis en estos tres lenguajes los autores decidieron plantear la herramienta de forma que pudiera procesar archivos de cualquiera de los lenguajes de la familia del C. Los errores que puede detectar son los siguientes:

- Errores en los tokens.
- Errores de prioridad de operadores.
- Errores en bloques de sentencias.

Jlint detecta errores semánticos analizando archivos .class de Java. Se utiliza en conjunción con AntiC. Es una herramienta implementada en C++ por su mayor rapidez frente a otros lenguajes como Java, pero se implemento para conservar su portabilidad entre las plataformas Windows y Linux. Puede detectar errores de los siguientes tipos:

- Errores de sincronización de hilos [Artho, 2001].
- Problemas que pueden darse en la jerarquía de herencia.

- Errores detectados en el análisis del flujo de datos, para ello calcula los posibles valores o rangos de variables locales o expresiones.

Ejemplo de salida JLint:

```
D:\edi4mod2\vallina\Datos\src\datos\datos\BaseDeConocimiento.java:276:      Local
variable 'nombre' shadows component of class 'datos/BaseDeConocimiento'.
D:\edi4mod2\vallina\Datos\src\datos\datos\BaseDeConocimiento.java:298:      Local
variable 'nombre' shadows component of class 'datos/BaseDeConocimiento'.
D:\edi4mod2\vallina\Datos\src\datos\datos\BaseDeConocimiento.java:429:      Local
variable 'nombre' shadows component of class 'datos/BaseDeConocimiento'.
D:\edi4mod2\vallina\Datos\src\datos\datos\BaseDeConocimiento.java:488:      Local
variable 'join' shadows component of class 'datos/BaseDeConocimiento'.
Verification completed: 4 reported messages.
Ejemplo salida Antic:
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:79:7: Possible miss of
BREAK before CASE/DEFAULT
D:\edi4mod2\vallina\Datos\src\datos\BaseDeConocimiento.java:458:3:  May  be  wrong
assumption about loop body (semicolon missed)
D:\edi4mod2\vallina\Datos\src\datos\Poblacion.java:21:33: ')' expected
D:\edi4mod2\vallina\Datos\src\datos\Poblacion.java:21:38:   Unbalanced   ')'   in
statement
D:\edi4mod2\vallina\Datos\src\datos\Tabla.java:43:27: ')' expected
D:\edi4mod2\vallina\Datos\src\datos\Tabla.java:43:40: Unbalanced ')' in statement
Verification completed: 6 reported messages
```

FindBugs

FindBugs [Hovemeyer 2004a] [Hovemeyer 2004b] es una herramienta de detección de errores (bug checker) que se basa en *patrones de error* para detectar errores en programas escritos en lenguaje Java. FindBugs analiza los archivos bytecode de Java, es decir, archivos .class. Para ello hace uso de la librería BCEL [BCEL 2004], que sirve para manipular este tipo de código. Se utiliza esta librería como base para implementar los detectores para encontrar *patrones de error* en las clases. Los patrones de error son expresiones de código que no siguen la práctica correcta habitual, se pueden definir como anti-patrones, es decir patrones negativos de codificación, que indican que en ese punto puede haber un error. Estos patrones de error suelen provenir de malos hábitos a la hora de programar y pueden ser errores difíciles de encontrar, puesto que un programa puede compilar sin problemas, sin que el error provocado por el patrón de error se manifieste.

La herramienta permite la integración con varios IDEs, interfaces gráficos y de texto. También se dispone de una tarea Ant que lo ejecuta. Emite informes de errores en varios formatos: xml, emacs, etc. Además, FindBugs puede ser ampliado [Grindstaff 2004] construyendo nuevos detectores de patrones de error en Java. Su principal problema es la cantidad de memoria (está configurado para utilizar 256 MB) y que los análisis requieren un tiempo bastante elevado.

Ejemplo de salida:

```
datos/AnalizadorSemantico.java:291:291
Dm:   datos.AnalizadorSemantico.invocarMetodo(String)   invokes   dubious   new
String(String) constructor; just use the argument (M)
SS: Unread field: datos.AnalizadorLexico.ASIGNACION; should this field be static?
(M)
SS: Unread field: datos.AnalizadorLexico.CADENA; should this field be static? (M)
[…]
UrF: Unread field: datos.AnalizadorSintactico.token (M)
[…]
UrF: Unread field: datos.ImprimirFichero.objeto (M)
SIC: Should datos.Lista$CabLista be a _static_ inner class? (M)
[…]
```

Tabla 3. Categorías de errores detectadas según la clasificación propia de FindBugs

Categorías de errores	Descripción	Ejemplo
Correctness	Problemas que afectan a la corrección del código.	Comparación de objetos de la clase String utilizando == o !=.
Malicious code vulnerability	El código es susceptible de ser utilizado de una forma no esperada por el desarrollador.	Un método finalize() de una clase debería ser de acceso protegido y no público.
Multithreaded correctness	Problemas con la concurrencia.	Método no libera el bloqueo en todos los posibles caminos.
Performance	Problemas que afectan al rendimiento de la aplicación.	Método instancia un objeto sólo para poder acceder al objeto de la clase.
Style	Problemas que no afectan a la corrección pero que constituyen malas prácticas de programación.	Método utiliza el mismo código en dos ramas.

PMD

PMD [PMD] es una herramienta cuyo objetivo es el análisis de archivos fuente Java, en los que intenta detectar posibles errores mediante la comprobación de reglas. Las reglas están agrupadas en conjuntos estándar; además el usuario puede crear sus propias reglas.

Las clases de errores que detecta PMD son los siguientes:

- Bloques vacíos en sentencias try/catch/finally/switch
- Variables, parámetros o métodos privados no utilizados.
- Sentencias if / while vacías.
- Expresiones complicadas sin necesidad o innecesarias.
- Clases con complejidad ciclomática alta.

Tabla 4. Conjuntos de errores detectados según la clasificación propia de PMD

Conjuntos de errores	Descripción	Ejemplo
Finalizer Rules	Problemas con los métodos finalize()	If the finalize() method is empty, then it does not need to exist.
Unused Code Rules	Búsqueda de código no utilizado.	Avoid unused local variables such as <var_name>
Controversial Rules	Conjunto de reglas que no son aceptadas universalmente como buenas prácticas.	Assigning a "null" to a variable (outside of its declaration) is usually bad form.
Coupling Rules	Detectan un acoplamiento alto o inadecuado entre objetos o paquetes.	A high number of imports can indicate a high degree of coupling within an object.
Optimization Rules	Avisa de que pueden ser aplicadas mejoras en el código fuente respecto a su rendimiento.	A local variable assigned only once can be declared final.
Basic Rules	Verifican que se sigan un conjunto de buenas prácticas.	Empty Catch Block finds instances where an exception is caught, but nothing is done.
Design Rules	Busca malas prácticas en el diseño del código	Final field could be made static

Conjuntos de errores	Descripción	Ejemplo
	fuente.	
Security Code Guidelines	Verifica la guia de estilo de seguridad de Sun[2].	Exposing internal arrays directly allows the user to modify some code that could be critical.
Strict Exception Rules	Comprueba el cumplimiento de una guía de estilo estricta sobre el lanzamiento y captura de excepciones.	Avoid catching throwable.
JavaBean Rules	Avisa de clases JavaBeans que no siguen las reglas de estos componentes.	If a class is a bean, or is referenced by a bean, directly or indirectly it needs to be serializable.
java.lang.String Rules	Problemas en la manipulación de la clase String o StringBuffer.	Avoid instantiating String objects; this is usually unnecessary.
Code Size Rules	Comprobación de convenciones sobre tamaños de código de métodos y clases.	Excessive Method Length.
Import Statement Rules	Problemas con sentencias import.	Avoid duplicate import statements.
Clone Implementation Rules	Búsqueda de usos cuestionables del clone.	Object clone() should be implemented with super.clone().
Jakarta Commons Logging Rules	Malos usos del framework de Jakarta para realizar logs.	To make sure the full stacktrace is printed out, use the logging statement with 2 arguments.
Naming Rules	Reglas de estilo sobre convenciones de nombres.	Variables that are not final should not contain underscores
JUnit Rules	Buscan problemas en las clases de prueba realizadas con JUnit	The suite() method in a JUnit test needs to be both public and static.
Java Logging Rules	Buscan malos usos de las clases para hacer un log de ejecución.	In most cases, the Logger can be declared static and final.
Braces Rules	Reglas de comprobación de utilización de llaves en sentencias de control.	Avoid using 'if...else' statements without curly braces

PMD dispone de interfaz Ant, así como conexión para diferentes IDE's y su configuración es muy flexible.

En el proyecto han desarrollado una herramienta llamada CPD que es capaz de encontrar fragmentos de código repetido en un archivo fuente Java, C, C++ o PHP. El que haya en el código fragmentos repetidos no suele ser una buena técnica de programación, puesto que es mucho más conveniente crear un método con el código repetido y llamarlo donde haga falta. Nuestras aplicaciones serán más flexibles y ahorrarán memoria.

Ejemplo de salida:

```
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:26: Variables that are
not final should not contain underscores.
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:31:  This  final  field
could be made static
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:66:     Avoid     using
'if...else' statements without curly braces
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:75:     Avoid     using
'if...else' statements without curly braces
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorLexico.java:87:   Avoid   using   if
statements without curly braces
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorSemantico.java:241:          Switch
statements should have a default label
D:\edi4mod2\vallina\Datos\src\datos\AnalizadorSemantico.java:291:           Avoid
instantiating String objects; this is usually unnecessary.
D:\edi4mod2\vallina\Datos\src\datos\Principal.java:17:    Avoid    unused    local
variables such as 'principal1'
```

[2] Sun ha publicado una guía de estilo para impedir que el código fuente se utilice de forma diferente a como fue diseñado por el desarrollador. http://java.sun.com/security/seccodeguide.html#gcg

```
D:\edi4mod2\vallina\Datos\src\datos\Principal.java:20: All methods are static.
Consider using Singleton instead.
 […]
```

Checkstyle

Checkstyle [Checkstyle] analiza si archivos fuente de java cumplen una serie de normas, de hecho parte de estas normas están tomadas de PMD y de Findbugs; la diferencia principal entre ellas es que Checkstyle se centra más en detectar problemas de estilo en el archivo, por lo que se potencia este tipo de chequeos en comparación a PMD. Al igual que PMD usa conjuntos de reglas y tiene la capacidad de ser ampliadas y personalizadas. Las reglas estándar se dividen en:

- Comentarios Javadoc.
- Convenciones en nombres y cabecera de los archivos.
- Sentencias "import".
- Tamaño del código.
- Espacios en blanco y bloques.
- Diseño de clases.
- Código duplicado.
- Chequeo de métricas.

Dispone de interfaz Ant y también se puede configurar para que de la salida en un tipo de formato determinado o para indicar un conjunto concreto de normas que serán chequeadas.

JCSC

JCSC [JCSC] es una herramienta que al igual que Checkstyle se centra proporcionar reglas para asegurar un buen estilo en el archivo Java. Pueden crearse reglas personalizadas y dispone de varios conjuntos estándar, los siguientes:

- Chequeos generales.
- Chequeo de nombres usando expresiones regulares.
- Orden de los campos y los métodos en una clase o interfaz.
- Chequeo de comentarios Javadoc.
- Obtención de métricas sobre el código.

Al igual que ellas dispone de un interfaz Ant y se puede configurar fácilmente.

DoctorJ

DoctorJ [DoctorJ] es una herramienta cuyo objetivo es la comparación entre el código y la documentación para verificar que esta está completa y correcta. Sus principales campos de actuación son:

- Verificación de la documentación. Se lleva a cabo por una comparación entre lo que dicen los comentarios del Javadoc y el propio código fuente. Mediante este análisis se puede encontrar errores como la omisión de parámetros en la documentación, incorrecta escritura de los nombres de métodos y variables, etc.

- Generador de métricas sobre el código fuente.
  - Número de líneas de código del proyecto, de las clases, ...
  - Número de parámetros de los métodos.
  - Número de variables empleadas en un método.
- Análisis de Sintaxis. Esta parte del proyecto está aun en desarrollo y no es estable. Su objetivo sería detectar errores parecidos a los que detecta Jlint.

Una de las limitaciones que tiene esta herramienta es que esta implementada en C++, y se ideo para ser usada en sistemas Linux, a pesar de ello la plataforma usada para la implementación es POSIX, por lo que teóricamente podría ser compilada para Windows.

Jwiz

Jwiz [Geis 1999] es una herramienta cuyo objetivo es el análisis de archivos de código fuente Java. Su entrada son archivos fuente de Java sintácticamente correctos por lo que deberían de ser compilados previamente. Puede usarse para detectar errores de tipo funcional y errores producidos por el mal estilo de codificación.

Comparativa de las herramientas

Tabla 5. Resumen de las características de las herramientas de búsqueda de errores

Nombre	Versión	Entrada	Interfaces	Técnica de análisis	Comprobaciones
JLint	3.0	Bytecode	LC	Sintaxis, Flujo de datos	Incorrecciones, problemas de concurrencia
FindBugs	0.7.3 (2004)	Bytecode	LC, GUI, IDE, Ant	Sintaxis (patrones de error), Flujo de datos	Incorrecciones, problemas de concurrencia, mal rendimiento
PMD	1.6	Fuente	LC, GUI, IDE, Ant	Sintaxis (patrones de error)	Incorrecciones, normas de estilo
Checkstyle		Fuente	LC, Ant	Sintaxis	Normas de estilo
JCSC			LC, Ant	Sintaxis	Normas de estilo
DoctorJ			LC	Sintaxis	Coherencia entre documentación y código
Jwiz		Fuente	LC	Sintaxis	Incorrecciones

3.8.5 Herramientas de análisis dinámico: JUnit

JUnit es un conjunto de clases (framework) para automatizar las pruebas de los programas Java. Escrito por Erich Gamma y Kent Beck. Y distribuido bajo licencia de código abierto. Está disponible en: www.junit.org.

Consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba para los métodos de una clase (prueba unitaria) y ejecutarlos automáticamente. Los casos de prueba son realmente programas Java. Quedan archivados y pueden volver a ejecutarse tantas veces como sea necesario y además de forma conjunta, pruebas de regresión.

La idea consiste en evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado. En una clase que hereda de este framework creamos métodos en los que se instancia el objeto esperado y otro objeto al que vamos a someter al método que queremos probar. Mediante assertEquals comprobamos que sean iguales sino JUnit indica que hubo un fallo.

JUnit permite agrupar casos (métodos) de prueba mediante la clase del framewokr: TestSuite.

El propio framework incluye formas de ver los resultados (runners) que pueden ser en modo texto, gráfico (AWT o Swing) o como tarea en Ant.

En la actualidad las herramientas de desarrollo como NetBeans, JBuilder y Eclipse cuentan con plug-ins que permiten que la generación de las platillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, facilitando al programador enfocarse en la prueba y el resultado esperado, y dejando a la herramienta la creación de las clases que permiten coordinar las pruebas.

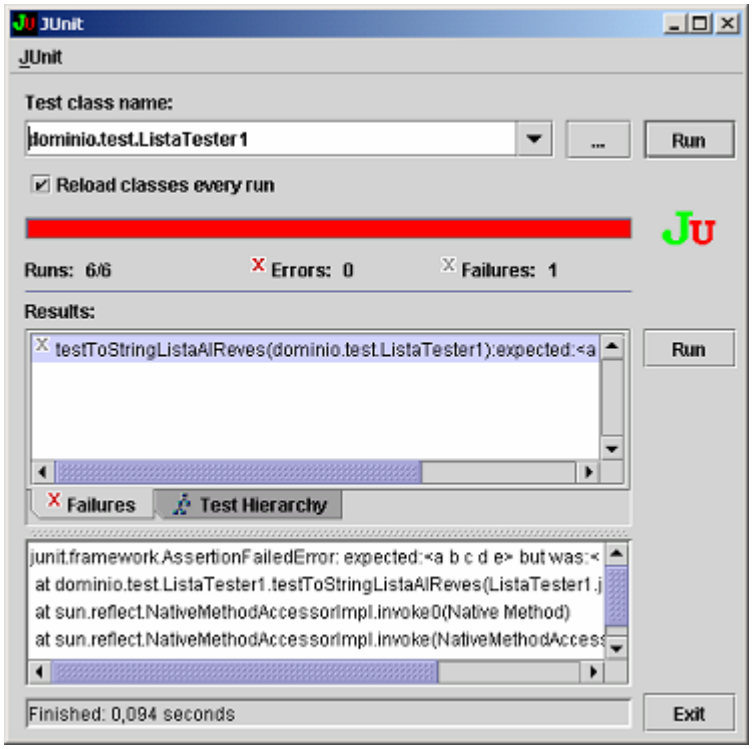

Figura 7. Ejecución de un caso de prueba con JUnit provocando un fallo

3.8.6 Conclusiones sobre las técnicas detección de errores

Resulta interesante trabajar en la **comprensión y solución de errores de compilación y ejecución dentro de un programa y esto puede ayudar a mejorar la calidad del código**.

Tabla 6. Comparación de técnicas de detección de errores

Técnica	Ventajas	Desventajas
Inspección manual	Es una técnica muy eficiente, con experiencia se pueden encontrar del 75% al 80% de los defectos. Al encontrar problemas y no solamente síntomas se ahorra tiempo en la corrección.	Consumen tiempo (se necesitan al menos 30 minutos para revisar 100 LOC) Es difícil hacerlas correctamente (debido a que hay que evitar subjetividad)
Técnicas dinámicas (pruebas y asertos)	Es una técnica imprescindible para comprobar que se cumplen los requisitos establecidos en la aplicación.	Es difícil construir un conjunto de prueba en un programa relativamente complejo para que tenga una buena cobertura. Además, la ejecución de los casos de prueba puede llevar bastante tiempo. La mayor cantidad de tiempo para encontrar y corregir un defecto se invierte en ir desde el síntoma al problema.

Técnica	Ventajas	Desventajas
Técnicas estáticas	Pueden explorar abstracciones de todos los posibles comportamientos del programa de forma genérica (con lo que ahorran tiempo). No están limitados a la calidad de los casos de prueba para ser efectivos.	La mayor cantidad de tiempo para encontrar y corregir un defecto se invierte en ir desde el síntoma al problema. El análisis no siempre es suficientemente preciso y genera falsos errores.

Es importante proporcionar al usuario información más detallada sobre su código que la proporcionada por los compiladores estándar. La utilización herramientas de análisis estático para la detección de errores no permite determinar de forma segura si existen determinados defectos en el código; pero si permite orientar a un revisor, ya sea el propio programador o una tercera persona en la búsqueda de estos defectos.

Cómo se puede aprender de los errores

La detección de errores en el código fuente puede ser una fuente importante de aprendizaje. De hecho, hay muchas áreas donde se potencia el contemplar los errores, tanto los propios como los cometidos por terceros, como una fuente de aprendizaje.

Por una parte, es necesario que un desarrollador conozca el significado a las implicaciones que tienen todos los mensajes de error que puede producir el compilador. Esto no es una tarea trivial, por una parte los mensajes de los compiladores están pensados para programadores con experiencia y por otra parte, el problema que causa el error puede ser distinto dependiendo del código fuente causante.

Además, es fundamental aprender a arreglar distintos tipos de error una vez comprendidos.

3.9 Conclusiones

En este capítulo hemos visto soluciones desde el mundo profesional, encaminados a la detección de errores y mejora de la calidad del código, bien como herramientas de detección específicas, o como entornos más generales: entornos de desarrollo y de gestión de proyectos software. Por otra parte también hemos visto distintas aportaciones desde el mundo académico: aprendizaje virtual de la programación, entornos de desarrollo pensados para el aprendizaje, gestores de prácticas y otros tipos de sistemas. Además, destacamos la importancia de la componente de colaboración que está presente tanto en el mundo profesional como en el académico.

Si comenzamos el análisis por los **sistemas del mundo académico**, una característica común a la mayoría de los sistemas es que están orientados al aprendizaje de los conceptos fundamentales e iniciales de la programación, primer curso universitario y **hay muy pocos orientados a enfrentarse con proyectos grandes, trabajando en equipo, en los que la localización de errores no es tan trivial**.

Hemos visto la **importancia fundamental de utilizar la Web como interfaz** entre usuario y sistema permitiendo su uso desde cualquier lugar y en cualquier momento, independientemente de la plataforma y sin necesidad de realizar una instalación y configuración para el sistema. También hemos visto que los sistemas actuales de aprendizaje de la programación para la Web **permiten muy poca interacción entre estudiante y sistema**, y dejan muy poco campo para que el estudiante experimente salvo raras excepciones, y **se limitan a exponer una serie de temas** sin proporcionar herramientas de programación que permitan experimentar al usuario, lo cual se ve como una gran carencia que debe de ser solucionada.

Se han estudiado distintos **entornos de desarrollo**, tanto comerciales como específicamente orientados al aprendizaje y en todos ellos la **gestión de errores es bastante pobre**. Se limitan a mostrar los errores resultado de una compilación pero **carecen de una *historia de compilación*** (registro de todas las compilaciones que realiza el desarrollador) y **proporcionan poca ayuda para comprender cada mensaje de error**, buscar cual es su causa y ver como se puede solucionar. Sólo AnimPascal (apartado 3.2.2) permite realizar un seguimiento de las acciones que realiza el estudiante en la solución de errores.

Por otra parte, en todos los entornos se aprecia la importancia de la integración de herramientas que permitan al usuario seguir todo el proceso de escritura, compilación y eliminación de errores de una aplicación software.

Un factor básico que se extrae de varios tipos de sistemas analizados y que ayudaría a la eficiencia del proceso de desarrollo es que los usuarios puedan trabajar en colaboración con otros sobre los proyectos. Hay bastantes sistemas colaborativos que se han aplicado con éxito tanto a situaciones de desarrollo de software [Shen, 2000] (CSCW, Computer Supported Cooperative Work) como en situaciones de aprendizaje [Bravo 2004] (CSCL, Computer Supported Cooperative Learning); además se ha detectado la tendencia hacia la colaboración en otros tipo de sistemas: gestores de prácticas, entornos que anteriormente sólo se pensaban para usuarios individuales y gestores de proyectos software. Por tanto, la **colaboración es importante** tanto en el mismo **proceso de desarrollo** como en el intercambio de conocimiento entre los desarrolladores para realizar un **aprendizaje más completo** y lograr una mejora en la calidad del código más rápida y efectiva.

Los **gestores de prácticas** permiten realizar un proceso de escritura por parte del estudiante y revisión por parte del profesor (en algunos ejemplos también intervienen otros estudiantes) que permite al alumno ver los defectos de su código e ir corrigiéndolos. Este proceso aunque interesante tiene al menos dos carencias: el estudiante tiene que realizar un proceso de **envío manual de su trabajo** y sólo se pueden hacer **revisiones al final del trabajo** sin poder revisar el proceso de realización.

Por último, se han revisado distintas **técnicas para la detección de errores**. Las técnicas de análisis estático permiten automatizar el proceso de búsqueda de errores y lo hacen de una forma eficiente. Parece necesario que las herramientas que **no sólo detecten errores sino que los registren, hagan un seguimiento y permitan un análisis y clasificación** de estos errores. Sobre esta clasificación se debería añadir la suficiente información semántica para que permita al usuario la interpretación del mensaje de error en su contexto, la relación con sus causas y las posibles soluciones que permitan eliminar el error.

Después de haber realizado este análisis se hace patente la **necesidad de nuevos entornos** que subsanen las carencias extraídas de los sistemas existentes actualmente. En el siguiente apartado plantearemos un modelo de las características que debería tener un sistema para la mejora de la calidad del código fuente y en los siguientes capítulos desarrollaremos prototipos que prueban la viabilidad de este modelo.