Departamento de Informática

Universidad de Oviedo

TESIS DOCTORAL

CLASIFICACIÓN DE USUARIOS BASADA EN LA DETECCIÓN DE ERRORES USANDO TÉCNICAS DE PROCESADORES DE LENGUAJE

Presentada por:
Juan Ramón Pérez Pérez
Para la obtención del título de Doctor por la Universidad de Oviedo

Dirigida por el
Profesor Doctor D. Juan Manuel Cueva Lovelle

Oviedo, enero de 2006

Resumen

Esta tesis lleva a cabo la definición de un modelo para un entorno de desarrollo de software, que facilita a los usuarios la construcción de aplicaciones con una mejor calidad del código que los entornos actuales.

El modelo que, está implementado en el sistema denominado SICODE, se basa en técnicas de procesadores de lenguaje que permiten realizar un análisis estático del código fuente, para descubrir los errores de programación presentes en el código. Este modelo pretende incidir sobre los estilos en la forma de programar y no sólo sobre errores puntuales. Para ello se ha establecido lo que se denomina una *historia de compilación*, sobre la que se generan métricas de evolución y frecuencia de errores a lo largo del tiempo. Se utiliza un modelo activo en la prevención de errores mediante el envío de avisos. Estos avisos permiten asociar información semántica a cada error en una base de conocimientos. Esta base de conocimientos es dinámica y se realimenta con la experiencia de todos los desarrolladores, lo que permite aprender de la experiencia de los demás.

En esta tesis se ha realizado un estudio que permite llevar a cabo la clasificación de usuarios basándose en la detección de errores en el código. SICODE nos ha permitido analizar los errores de un amplio número de proyectos realizados por desarrolladores. Así, hemos caracterizado distintos tipos de errores dependiendo de la experiencia de los programadores. Este estudio proporciona datos que permitirán construir las bases para modelizar al usuario y de esta forma hacer que el modelo sea adaptable.

La tesis ofrece nuevos enfoques sobre los sistemas de desarrollo existentes en la actualidad. La construcción de la *historia de compilación* permite un análisis más profundo que el que puede realizar un simple compilador, y este análisis está al servicio de la mejora del código. Además, el sistema está centrado en el entorno de desarrollo y fusiona el propio desarrollo de software con el aprendizaje para hacer un código de calidad: el usuario simplemente programa y SICODE le proporciona medios para mejorar el estilo de programación. Por último, la colaboración se utiliza, no sólo para facilitar el desarrollo, sino también en la construcción de una base de conocimientos, lo cual permite construir un sistema que se realimente con el intercambio de experiencias entre los usuarios.

Palabras clave

Entorno de desarrollo integrado, análisis estático de errores, base de conocimientos, historia de trabajo, historia de compilación, CSCL (Computer Supported Cooperative Learning), calidad del código fuente, unidad de compilación, aprendizaje basado en errores.

Abstract

This thesis defines a model for a development environment of software that makes easier the construction of the applications with an improvement of the code quality that the current environments.

The model, we have implemented in the system called SICODE, is based on language processor techniques which make it possible to carry out a static analysis of the source code. With this model we want to see the styles of programming and not just isolated errors. For this purpose the *compilation history* is used that lets us produce statistics of error evolution and frequency during a period of time. An active model is used in the prevention of errors by means of sending of warnings. These warnings allows to link semantic data to each error in the knowledge base. This knowledge base is dynamic and it is feedback with the experience shared by all developers, this permits to learn of the experience of other.

A research that allows classifies the users based in code errors, has been carried out in this thesis. The SICODE system has been use to analyse a wide number of projects wrote to different developers. This way we have characterized different types of errors according to the experience of the developer. This investigation provides data which will make it possible to build up the foundation of a user model and so the system will be adapted to each user.

This thesis makes new approaches to the nowadays existing development systems. The building of the compilation history allows a deeper analysis than that of a simple compiler and it helps to improve the code. Besides, the system focuses on the development environment and merges development of software with learning for developing an improvement code: the user only has to make a program and SICODE provides means of improvement the programming style. Finally, the collaboration is used not just in the development, but also in the construction of a knowledge base providing the exchange of the experiences between the users.

Keywords

Integrated development environment, static analysis of errors, knowledge base, work history, compilation history, CSCL (Computer Supported Cooperative Learning), source code quality, compilation unit, learning based on errors.

Agradecimientos

Quisiera reflejar en estas líneas mi agradecimiento a todos mis compañeros, muy especialmente a los integrantes del grupo del Laboratorio de Tecnologías Orientadas a Objetos (OOTLab), por todo el apoyo tanto en la parte científica como en la parte personal que me han proporcionado durante la elaboración de esta tesis doctoral, y que de una forma u otra han colaborado a que pudiera llegar hasta aquí.

A Juan Manuel Cueva, mi director de tesis, que con sus sabios consejos me ha guiado en la escritura de esta tesis haciendo posible que este documento tuviera forma y porque estuvo siempre ahí, apoyándome y caminando a mi lado cuando más lo necesité.

A mis compañeros del café, Macamen, Marián, Fernando y Lourdes, por permitirme disfrutar junto a vosotros de esos pequeños momentos de esparcimiento y porque las ideas muchas veces surgen a partir de conversaciones de café.

A todos los estudiantes que han tenido que ver con SICODE, especialmente a Ramón, Daniel, Xuan y Cristina, por las ideas que han aportado y porque en las conversaciones con ellos el sistema ha crecido y se ha perfeccionado.

A mis padres porque gracias a ellos he llegado hasta aquí. A mis hermanos, Joaquín y Pablo, porque siempre confiaron en mí y son un ejemplo de cómo hacer bien el trabajo.

A mi mujer, Puerto, por tener paciencia conmigo, por darme fuerzas cuando las mías flaqueaban, por escuchar cuando le planteaba cosas sobre la investigación y aportar un montón de cosas, en definitiva por darme tanto amor. A mi hija Laura porque me ha cambiado la forma de ver la vida y eso me ha proporcionado un nuevo espíritu para afrontar retos como este.

Juan Ramón, enero de 2006

Tabla de Contenidos

Capítulo 1. Introducción

En este capítulo hacemos un planteamiento general del problema que abordamos en la tesis y los objetivos globales que pretendemos conseguir. En el último apartado, describimos la organización de todo el documento.

1.1 Planteamiento del problema

La aparición de errores es algo común en el proceso de desarrollo del software. Los errores siempre conllevan pérdidas de tiempo y dinero en el proceso de desarrollo, en ocasiones, los errores llegan a la fase en la que producto software debe ser utilizado por los usuarios finales, lo que implica productos de baja calidad que causan defectos tanto funcionales como no funcionales. Esta baja calidad del software supone un grave problema. Algunos informes dan datos como que *el 60% de los desarrolladores de software en Estados Unidos están involucrados en resolver errores que se podrían haber evitado* [Jones, 1998].

Estos problemas pueden reducirse utilizando distintas técnicas de ingeniería del software que dan buenos resultados y se hacen imprescindibles para obtener software de calidad [Pressman, 1997]. Sin embargo, dentro del proceso general de construcción de software, es básico fijar la atención en el subproceso de escritura de código de la aplicación. En este subproceso, la interacción entre el programador y el entorno de desarrollo es fundamental para lograr reducir la cantidad de defectos introducidos. Actualmente, los compiladores y otras herramientas de análisis de código permiten la detección automática de distintos tipos de errores. Utilizando esta información, el desarrollador puede eliminar estos errores a través de un proceso de depuración. Esto es fundamental para obtener un producto software de alta calidad; según Allen [Allen, 2002] *existe una interdependencia crucial entre la depuración efectiva y el desarrollo efectivo.*

Para conseguir una alta calidad del software, es muy importante tener programadores con experiencia ya que la calidad y seguridad del software producido depende inevitablemente de la destreza y experiencia de los programadores involucrados. Además, según Allen: *hay pocos programadores experimentados para la demanda que existe. Para formar a más programadores es insuficiente la enseñanza tradicional de conocimiento teórico. Necesitamos transmitir habilidades prácticas en el desarrollo de sistemas robustos, esto lleva años de experiencia* [Allen, 2002]. Parte de este conocimiento práctico consiste en la habilidad de diagnosticar eficientemente y subsanar los errores del sistema software. La depuración efectiva está lejos de ser una habilidad trivial. Buscar y eliminar errores ocupa una porción significativa del tiempo de desarrollo en un proyecto software. Si esta tarea se pudiera realizar de forma más eficiente, el software resultante podría ser más fiable y el proceso de desarrollo sería más rápido. Por tanto, es fundamental el aprendizaje de **técnicas de detección y corrección de errores** que sirvan para mejorar el software ya creado y se puedan utilizar en la mejora del proceso general de codificación. El aprendizaje de estas técnicas no es sencillo y para alcanzar un dominio

adecuado es necesario complementar los conocimientos teóricos con la experiencia práctica que ayude a la asimilación de estos conceptos. Además, es importante que lo aprendido no sólo sirva para la depuración de errores sino poder aplicarlo también a la codificación y así evitar que se produzcan nuevos errores y poder hacer más eficiente todo el ciclo de desarrollo.

La adquisición de habilidades y experiencia en el ciclo de desarrollo que debe realizar un desarrollador no se limita a una etapa de formación en el inicio de la vida. Los desarrolladores deben de estar aprendiendo continuamente porque todos los días surgen nuevas tecnologías y tendencias mientras que otras se quedan obsoletas. Según Ivar Jacobson [Jacobson 2002]: *El desarrollo de software nunca ha sido tan complejo como lo es ahora. Los desarrolladores de software trabajan intensivamente con el conocimiento. No sólo deben comprender las nuevas tendencias y tecnologías, necesitan saber como aplicarlas de forma rápida y productiva.*

Pese a la gran cantidad de trabajos y los múltiples enfoques existentes en el campo de la calidad en el desarrollo de software, hay muy pocas propuesta relacionadas con la detección, análisis y corrección de errores en el código del programa [Luján-Mora, 2003]. En la práctica, esta es una faceta que cada desarrollador va aprendiendo, en la mayoría de los casos de forma autodidacta, por medio de la experiencia que va adquiriendo utilizando el ensayo y error en el desarrollo de proyectos. Los libros de lenguajes de programación prácticamente no tratan el tema y los manuales se limitan a proporcionar una serie de instrucciones de manejo de las herramientas; pero no profundizan en la aplicación de las técnicas. Los enfoques científicos plantean modelos de difícil aplicación práctica [Hovemeyer, 2004b]. Las únicas propuestas a las que se pueden acoger los desarrolladores son libros que describen una serie de reglas prácticas para mejorar el desarrollo de aplicaciones.

Los compiladores automatizan la comprobación de alguna de estas reglas prácticas y tras un proceso de compilación emiten mensajes indicando su violación. Sin embargo, los mensajes de error de los compiladores no facilitan la labor de la corrección en la aplicación de las reglas, ya que muchas veces son crípticos y ambiguos. Esto causa que sean difíciles de entender para un desarrollador con poca experiencia. Además de ser difíciles de comprender, *los compiladores solamente proporcionan síntomas de defectos y es necesario entender dónde y cual es el problema* [Humphrey, 1997]. Muchas veces, pese a disponer de la información que proporciona el compilador no es sencillo relacionar los síntomas indicados en los mensajes de error con los problemas reales para eliminarlos y tratar de evitarlos en el futuro. Según Humphrey [Humphrey, 1997]: *Es importante separar la cuestión de encontrar o identificar los defectos de la determinación de sus causas. El primer paso para gestionar los defectos es entenderlos. Para hacer eso se deben reunir los datos de los defectos. Entonces, se podrán entender esos errores y comprender cómo evitarlos. También se podrá comprender como encontrarlos mejor, corregirlos o prevenir los defectos que todavía se introducen.*

Por lo tanto, es necesario la creación y uso de **herramientas que no sólo detecten errores sino que contabilicen, hagan un seguimiento y permitan un análisis y clasificación** de estos errores. Además, se debe añadir la suficiente **información semántica a los errores, de forma que permita al desarollador la interpretación del mensaje de error en su contexto, la relación con sus causas y las posibles soluciones que permitan eliminar el error**. Por último, los entornos no deben de ser pasivos y esperar a que el usuario pida información sino que deben **indicar al usuario los errores en el momento que se produzcan mediante avisos activos**.

Un punto básico que ayudaría a que todo el proceso se desarrollase de forma eficiente es que los desarrolladores trabajasen en forma colaborativa. Hay muchos 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 et al, 2004] (CSCL, Computer Supported Cooperative Learning) Consideramos que es fundamental que cualquier entorno de desarrollo proporcione facilidades para desarrollar el trabajo en grupos de forma adecuada, y proporcione herramientas tanto para compartir archivos como para coordinarse en todas las tareas involucradas en el desarrollo. Otro ámbito en el que la colaboración va a aportar un valor añadido es la construcción de una base de conocimientos que, a través de la experiencia de los desarrolladores permitirá añadir información semántica a los distintos mensajes de error recogidos por el sistema.

Por último, no queremos que la utilidad del sistema se limite a la corrección de errores puntuales, debería permitir elaborar una guía adaptada al perfil de cada desarrollador extraído del análisis de errores. De esta forma se proporcionaría una herramienta para mejorar el estilo de programación y evitar futuros errores.

En la tesis se plantean dos contextos de aplicación para la herramienta descrita: un contexto académico donde los estudiantes utilizan el sistema en grupos para aprender técnicas de programación, construcción de software y lenguajes de programación y un contexto profesional donde los desarrolladores tienen que adquirir el dominio de nuevas técnicas y producir software de alta calidad.

1.2 Objetivos de la tesis

En este apartado se enuncian los objetivos que se pretenden desarrollar en esta tesis visto el planteamiento previo. Existen dos grandes ejes que marcan la línea de investigación: el modelado del sistema y la utilización de éste para el análisis de datos orientado a la clasificación de los desarrolladores; cada uno de estos ejes conlleva la consecución de varios sub-objetivos. A continuación, se describen dichos ejes.

1.2.1 Modelado de un sistema que permita la mejora de la calidad del código fuente

El primer objetivo de esta tesis es el modelado de un sistema que permita superar los inconvenientes de los sistemas actuales en la construcción de software de calidad descritos, muy brevemente, en el apartado anterior.

El sistema esta fundamentado en las siguientes bases:

1. **Sistema orientado a la mejora de la calidad del código fuente.** Permite detectar, eliminar y prevenir errores de forma más eficiente a través técnicas de procesadores de lenguaje. Para ello dispone de las siguientes características:
    - Captura y almacenamiento on-line de errores y *warnings* mediante técnicas de procesadores de lenguajes. Se recogen en tiempo real los problemas existentes en el código mientras los desarrolladores están escribiendo.
    - Mantenimiento de una *historia de errores*, facilitando el seguimiento y la monitorización de usuarios individuales y de grupos de trabajo en relación con los errores de programación.
    - Análisis de los errores de programación cometidos por los desarrolladores. Para ello se estudian distintas métricas para aplicarlas sobre los errores individuales y de grupo recogidos en la base de datos, con el fin de definir indicadores personalizados para cada desarrollador.

2. **Entorno que proporciona soporte para equipos virtuales de desarrollo y permita aprender de los errores propios**. El sistema debe proporcionar distintos servicios a los desarrolladores a través de un entorno de desarrollo.
   - Los desarrolladores deben disponer de un Entorno Integrado de Desarrollo sobre Internet.
   - Los desarrolladores pueden aprender y adquirir experiencia en las nuevas técnicas de construcción de aplicaciones mediante el desarrollo de proyectos software aprendiendo así de los errores propios.
   - Los desarrolladores pueden colaborar en el desarrollo y revisión de programas. Se pretende facilitar el funcionamiento de equipos virtuales de desarrollo mediante herramientas de comunicación y colaboración. Todos los desarrolladores tendrán acceso al código de los demás, permitiendo la realización de revisiones y mejoras sobre el código fuente.
   - Los desarrolladores disponen de información activa y adaptada que les ayuda a realizar correctamente la escritura del código. Se pretende que el entorno, mediante la emisión de avisos, proporcione información relevante y adaptada a los desarrolladores mientras escriben el código.
3. **Base de conocimientos colaborativa** que asocia información semántica a cada uno de los tipos de errores. A través de esta base de conocimientos, los usuarios pueden consultar la información relacionada con los errores surgidos en el análisis de su código.
   - El sistema permite una asociación automática de los distintos mensajes de error con la información disponible en la base de conocimientos.
   - La base de conocimientos dispondrá de información contextualizada y derivada de la experiencia de los desarrolladores lo que permite que cualquier desarrollador pueda aprender de la experiencia de los demás.
   - La base de conocimientos es dinámica. Permite que la información contenida crezca y el sistema se realimente con la información que los desarrolladores van generando.

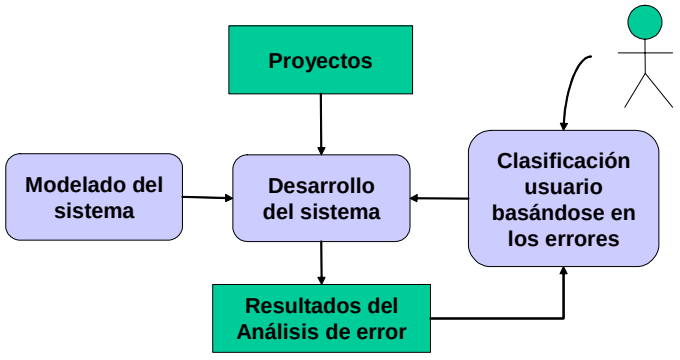


Figura 1. Representación esquemática de los objetivos de la tesis

1.2.2 Clasificación de usuarios basada en la detección de errores

El segundo gran objetivo es la utilización del sistema modelado en el objetivo anterior para recoger y analizar datos que permitan clasificar a los usuarios del sistema (desarrolladores) utilizando las estadísticas de errores. La consecución de este objetivo conlleva:

1. **Caracterizar los errores** que cometen los desarrolladores en distintos niveles de aprendizaje basándose en el sistema con los objetivos descritos en el apartado anterior. Para ello se analizarán los errores de una amplia muestra de proyectos de distintos niveles académicos.
2. **Obtener un perfil de desarrollador** dependiendo de su experiencia, utilizando los datos del análisis anterior.
3. **Desarrollar una guía adaptada al perfil de cada usuario** que permitirá al propio desarrollador conocer mejor los errores cometidos a la hora de programar para evitarlos y así, construir aplicaciones de mayor calidad.

La información extraída en este objetivo se podrá utilizar para realimentar la base de conocimientos del sistema y así proporcionará permitirá una adaptación al perfil del usuario.

1.3 Organización del documento

A continuación describimos la organización de este documento por capítulos:

Introducción y problema a resolver. En el capítulo 1 se hace una introducción general planteando el problema abordado y los objetivos generales que queremos alcanzar con esta Tesis. En el capítulo 2 se describen con mayor profundidad los retos de un sistema que pretenda hacer más fácil la utilización de lenguajes de programación, obteniendo un código fuente de calidad: sin errores y fácilmente mantenible.

Estado del arte. En el capítulo 3 se estudian de forma crítica otros sistemas que desde distintas perspectivas intentan abordar el problema planteado y se identifican sus puntos fuertes y sus carencias.

Modelo del sistema. En el capítulo 4 se plantean los requisitos del modelo para un sistema que permita dar solución al problema planteado.

Implementación del sistema. En los capítulos siguientes se describen y analizan diferentes prototipos desarrollados del sistema definido en el capítulo 4:

- Sistema SICODE. Capítulo 5. Descripción global del sistema y explicación de su división en subsistemas y de las decisiones comunes que se han tomado en su implementación.
- Compilador Web SICODE. Capítulo 6. Describe el primer prototipo desarrollado en el que se diseñan las diferentes características del sistema para someterlas a una primera evaluación.
- Sistema de análisis de errores de programas: PBA. Capítulo 7. Se plantea un prototipo para el subsistema de captura, almacenamiento, análisis de errores y generación de informes sobre los mismos.
- Sistema para la colaboración en el desarrollo de aplicaciones: COLLDEV. Capítulo 8. Se explica el prototipo que desarrolla el subsistema que permite la comunicación entre los distintos desarrolladores de un proyecto, la colaboración en el desarrollo

de los distintos usuarios utilizando un almacén común para los proyectos software y la coordinación en las tareas de desarrollo y corrección de errores.

- Entorno de desarrollo integrado en Web y base de conocimientos colaborativa: IDEWeb. Capítulo 9. Se describe el subsistema de entorno de Web de desarrollo y la base de conocimientos que permite obtener a los usuarios información sobre las causas de un error y como corregirlo.

Clasificación de usuarios. En el capítulo 10 se realiza un estudio del comportamiento de los desarrolladores en distintos niveles de aprendizaje. Recopilando un número relevante de proyectos software que se analizan con nuestro sistema. Así podemos comparar los tipos de errores que cometen los usuarios dependiendo de lo expertos que sean en las técnicas de programación.

Conclusiones y trabajo futuro. En el capítulo 11 se describen las aportaciones de este trabajo y las conclusiones extraídas y las posibles líneas de investigación que se abren a partir de ahora.

Apéndices. Se incluyen cuatro apéndices con información complementaria:

- Apéndice A. Archivos de configuración del sistema de análisis de errores en programas. Incluye los archivos de configuración XML de distintos aspectos de este subsistema.
- Apéndice B. Información sobre los errores frecuentes. Incluye información sobre los errores más frecuentes analizados en el capítulo 10.
- Apéndice C. Avisos por proyecto e informes de análisis. Se incluyen todos los tipos de errores para cada conjunto de proyectos con su frecuencia. Además incluye los informes generados directamente por el subsistema de análisis de errores en programas.
- Apéndice D. Referencias bibliográficas.

Capítulo 2. Problemática en la calidad del código fuente y en el aprendizaje de la programación

Antes de proceder a explicar los sistemas actuales orientados al aprendizaje y mejora de la calidad del código, es necesario, enmarcar en el contexto adecuado la problemática que tiene un programador principiante; pero también uno experimentado en el aprendizaje de la programación y la mejora de su código fuente. Este capítulo servirá para enfocar y mostrar las dificultades que existen en dicho aprendizaje.

2.1 Dificultades de los programadores principiantes para aprender los conceptos de programación

Satratzemi y sus colegas [Satratzemi, 2001] a partir de [Brusilovski, 1998] [Boulay, 1989] resumen las dificultades que encuentran los programadores a la hora de enfrentarse a un programa:

- Dificultades que aparecen por tener un modelo conceptual de la máquina que difiere del modelo real.
- Dificultades atribuidas a la sintaxis y semántica de los lenguajes de programación.
- La necesidad de comprender las estructuras de programación establecidas.
- La necesidad de aprender cómo diseñar, desarrollar, verificar y depurar un programa con ciertas herramientas.
- El hecho de que los editores, compiladores y depuradores se diseñan por y para programadores profesionales. De tal forma, que tienen una complejidad extra para los programadores principiantes.
- Los entornos de programación no son capaces de ofrecer visualización de la ejecución del programa. Así, los detalles de la ejecución permanecen ocultos y los programadores pueden percibir un programa como una caja negra difícil de comprender. Esta carencia de realimentación visual hace difícil la compresión de la semántica del lenguaje.

Kölling [Kölling, 2003] se plantea la causa de las dificultades en el aprendizaje de la programación y en concreto de la programación orientada a objetos y considera que la programación a objetos no es más compleja que la estructurada; pero la carencia de herramientas apropiadas la hacen más complicada. Kölling [Kölling 1999a] plantea tres

problemas clave de los entornos de desarrollo actuales para la enseñanza de la programación orientada a objetos:

- Los entornos no son orientado a objetos. Un entorno para un lenguaje orientado a objetos no tiene por que ser un entorno orientado a objetos. Sin embargo, el entorno debería reflejar el paradigma del lenguaje. En la mayoría de los entornos hay que trabajar con el sistema de archivos y la estructura de directorios. Además, la interacción del usuario es exclusivamente con líneas de código fuente, no con objetos. La interacción con objetos no está soportada por muchos entornos pese a ser uno de los conceptos fundamentales.
- Los entornos son demasiado complejos. Un ejemplo es la utilización, para programar en Java, del Java SDK de Sun en línea de comandos, lo que provoca los problemas típicos de la línea de comandos. En otros casos, se utilizan entornos que están pensados para programadores profesionales y que suponen una sobrecarga en su interfaz para los principiantes. Otros son modificaciones de entornos para lenguajes procedurales que no disponen de las herramientas apropiadas para las abstracciones de la orientación a objetos.
- Los entornos centran al usuario en la construcción de interfaces. Muchos entornos utilizan gráficos; pero para tareas no relevantes para que los estudiantes puedan adquirir los conceptos de orientación a objetos adecuados. En concreto, muchos entornos se concentran en la construcción de interfaces gráficos (GUIs) y esto puede distorsionar la visión de la programación orientada a objetos. Una de las aplicaciones más interesantes de los gráficos sería mostrar la estructura de clases y pocos entornos la utilizan.

2.2 Entornos de programación y sus limitaciones en para el aprendizaje

Los entornos actuales de desarrollo de aplicaciones software aportan de forma integrada un gran número de herramientas: editores, compiladores, depuradores, gestores de configuración, gestores de versiones, etc. Sin embargo, desde el punto de vista de un programador principiante esto, lejos de facilitar su trabajo con el entorno y que este permita un aprendizaje de la programación, crea barreras, a veces, difícilmente superables. Estos entornos no proporcionan un modelo mental coherente con los conceptos básicos de programación y tampoco ayuda a los programadores a construir nuevos conocimientos de programación a partir de conocimientos previos.

Un punto especialmente destacable por la complicación que representa para un programador principiante, es la explicación de los errores: normalmente consiste en un mensaje corto, sin más explicación, acompañado de la línea donde se detecta este tipo de error. Esto es suficiente para los programadores profesionales con experiencia, consiste en un simple enlace mental a un tipo de error ya solucionado muchas veces y para el cual se dispone de patrones de solución. Sin embargo, esto es muy críptico para los programadores sin experiencia, por lo que muchas veces tienen que emplear bastante tiempo buscando información relacionada que les indique cual es el problema real indicado por el mensaje relacionándolo, si es necesario, con los conceptos que ya tiene el programador y cuales son los posibles patrones para solucionar el problema.

Por último, los entornos tampoco proporcionan facilidades para integrarlos en la dinámica del aprendizaje. Por una parte son sistemas pasivos que analizan el código de forma superficial (lo imprescindible para poder generar código objeto) y sólo cuando el

desarrollador realiza la compilación [Jacobson 2002]; por otra parte carecen de herramientas que permitan guiar al programador para mejorar su estilo de programación, el único objetivo es corregir errores puntuales en el código.

2.3 La práctica de la programación como medio para aprender y mejorar

Los desarrolladores tienen la máxima de "a programar se aprende programando", es decir, las destrezas necesarias para ser un buen programador se adquieren sobre todo en la práctica, implementando programas, encontrando problemas y buscando soluciones a estos problemas. Sin embargo, los sistemas de aprendizaje de la programación raramente permiten ponerse a programar directamente; se trata más bien de la transmisión de conocimientos sobre programación de forma teórica. Sería necesario plantear un sistema en el que el estudiante pudiese ponerse a programar desde el primer momento incluso antes de saber instalar el compilador y que fuese el propio entorno el que le fuese guiando en su aprendizaje.

2.4 Programar en grupo, colaborando en la solución de errores y en mejora del código fuente

Los entornos de programación están pensados para trabajar individualmente con ellos; normalmente cuando se trabaja en equipo este se encuentra próximo físicamente y los desarrolladores se pueden comunicar directamente sin necesidad de soporte informático. Normalmente el soporte para el trabajo en equipo de desarrollo consiste en un sistema de control de versiones (CVS) [Cederqvist 1993] común que permita compartir sin problemas los archivos del proyecto entre todos los miembros del equipo. Sin embargo, este sistema en si mismo no ayuda a que los programadores se coordinen y colaboren en la mejora de la calidad del código en general y en la solución de errores del código fuente en particular. Es necesario un sistema que además de compartir los ficheros permita la comunicación entre desarrolladores y la toma de decisiones en el proyecto, dando soporte a lo que se denominan equipos virtuales de desarrollo [Scotts 2003].

2.5 Programar a distancia

Es útil, sobre todo a la hora de dar los primeros pasos en el aprendizaje de la programación, el poder programar sin necesidad de tener instalado un compilador, ni un entorno en la máquina local. Si disponemos de un entorno Web que permita realizar una edición de los archivos fuente, una gestión de los archivos del proyecto almacenados en el servidor y una compilación de estos utilizando un compilador del servidor el desarrollador podría ponerse a programar en cualquier ordenador con conexión a Internet sólo con tener instalado un navegador, independientemente del resto del software de que disponga.

2.6 Tutor permanente: qué estoy haciendo mal y cómo puedo solucionarlo y no volver a repetirlo

Los compiladores pueden detectar determinados problemas sintácticos, semánticos o lógicos e informar de ellos mediante un mensaje de error. Pero este mensaje de error es muchas veces demasiado corto y ambiguo, por lo que puede resultar difícil de interpretar para un desarrollador poco experimentado; otras veces aunque el mensaje del error sea fácil

de descifrar, la causa de este puede ser difícil de encontrar; tanto en un caso como en otro aunque exista un mensaje de error señalando un error el desarrollador puede estar perdido y necesita hacer trazas sobre el programa y cambios que le vayan guiando hasta llegar a la solución del problema, gastando en ello una gran cantidad de tiempo. Para evitar esto, el desarrollador necesita justo en el momento de cometer un error una indicación de lo que está haciendo mal y una indicación de cómo solucionarlo según el contexto en el que se encuentre y debería de ser el propio entorno el que detectase el problema y emitiese las indicaciones.

2.7 Aprendizaje basado en ejemplos y a partir de los errores

Un ejemplo siempre sirve para ilustrar cualquier caso teórico y muchas veces el contexto que proporciona un ejemplo permite comprender mejor el concepto que se pretende transmitir. Sería interesante disponer de ejemplos concretos para los conceptos con los que estamos trabajando, incluso varios ejemplos que correspondan a varios casos; sin embargo, el trabajo con ejemplos es costoso para el que enseña siempre lleva tiempo prepararlos; sería interesante disponer de un sistema que pudiera recoger los ejemplos cuando vayan surgiendo, clasificarlos y relacionarlos con los conceptos.

Si los ejemplos positivos ayudan a realizar un buen aprendizaje, los negativos son aún más adecuados a la hora de proporcionar información de cómo NO se deben hacer las cosas y por tanto, cómo se deberían hacer. Si fuéramos capaces de personalizar estos ejemplos y llevarlos al propio código del desarrollador conseguiríamos explicar al usuario que buenas y malas prácticas está siguiendo a la hora de programar; permitiríamos de esta forma que el usuario estuviese informado sobre sus malos hábitos y mejorase su forma de programar.

2.8 Creación de software de calidad

El trabajo de un ingeniero del software es entregar productos de *calidad* con unos costes y programaciones planificados.

Desde hace mucho tiempo el concepto de calidad de software ha sido algo que muchos autores han intentado definir, por ejemplo Pressman [Pressman, 1997] define *calidad de concordancia* como el grado de cumplimiento de las especificaciones de diseño durante su realización. Lo que parece evidente es que todas las definiciones refieren el concepto de calidad como que el programa implementado *haga lo que tiene que hacer* y además *lo haga bien*. Según Humphrey: *Los productos software deben satisfacer tanto las necesidades funcionales de los usuarios así como realizarlas de una forma segura y consistente* [Humphrey, 1997].

Aunque hay muchos aspectos relacionados con la calidad del software, el primer aspecto de la calidad está relacionado necesariamente con sus defectos. Un defecto, es cualquier cosa que reduce la capacidad de los programas para cumplir completa y efectivamente las necesidades de los usuarios. Los errores triviales de implementación pueden causar serios problemas en el sistema. Según Humphrey, la fuente de muchos defectos software son simples descuidos y errores del programador [Humphrey, 1997].

Para reducir el número de defectos que se introducen en los productos software, el ingeniero debe cambiar la forma de hacer las cosas. Sin embargo, para eliminar los defectos en los productos, sencillamente hay que encontrarlos. La eliminación de defectos es, por lo tanto, un proceso más sencillo que la prevención de defectos.

Los defectos deben de ser encontrados y corregidos para que no lleguen a la fase de producto acabado en explotación. El problema es que lleva mucho tiempo y esfuerzo encontrar y corregir defectos; además, este tiempo es difícil de predecir con lo que a menudo causa problemas de costes y ajuste de tiempos al plan de trabajo. Por tanto, sería muy beneficioso que los ingenieros utilizasen técnicas que evitaran la introducción de defectos.

2.8.1 Comprensión de los defectos

El primer paso para gestionar los defectos es entenderlos. Para ello, hay que reunir datos de los defectos cometidos. A través de estos datos, se podrá comprender como evitarlos y como encontrarlos mejor y corregirlos.

Hay que tener en cuenta los errores que localiza, el propio programador y también los que pueden encontrar otros. Estos últimos errores, tienen gran importancia ya que, si el propio programador no los encuentra pueden ser síntomas de debilidades en el proceso de escritura del software por parte del programador.

A la hora de realizar revisiones, también es fundamental conocer los tipos de errores cometidos en anteriores programas, teniendo en cuenta que si se desarrolla software de una misma forma los tipos de errores serán muy parecidos a los encontrados anteriormente. Como cada programador tiene una formación y experiencia, diferentes los tipos de defectos también lo podrían ser, por tanto, la estrategia de revisión debería basarse en el perfil de defectos personales de ese programador.

Para conseguir esta calidad en el software es casi imprescindible pasar por un proceso de revisión. Citando a Pressman [Pressman, 1997]: *las revisiones del software son un "filtro" para el proceso de ingeniería del software. Se pueden aplicar en varios momentos del desarrollo del software y sirven para detectar defectos que puedan así ser eliminados.*

De nuevo, aquí, se pone de manifiesto la importancia de la colaboración en un equipo de desarrollo; una revisión es una forma de aprovechar la diversidad de un grupo de personas para: señalar la necesidad de mejoras en una aplicación, conseguir un trabajo técnico de una calidad más uniforme, o al menos más *predecible*, que la que puede ser conseguida sin revisiones. El mismo Pressman, plantea revisiones en la fase de diseño de la aplicación como un método efectivo para detectar defectos del software en fases tempranas.

A pesar de todas estas recomendaciones, lo cierto es que muchas veces los defectos no se pueden detectar hasta fases donde se disponga de código para realizar pruebas. Es más, las metodologías ágiles de desarrollo de software retrasan, a propósito esta revisión a la fase de codificación. Programación Extrema [Beck, 2000], por ejemplo, propone la *programación en parejas* como medio de asegurar que una persona realiza la revisión del código en el mismo momento en que la otra lo escribe.

Llegados a este punto parece evidente la necesidad de la revisión del código. Este proceso de revisión no siempre cumple las funciones pretendidas por tres razones:

- El programador se conforma básicamente con evitar los errores de compilación, por esto se recomienda que sea otra persona la que revise el código. Como dicen Freedman y Weinberg [Freedman, 1990]: "aunque la gente es buena descubriendo algunos de sus propios errores, algunas clases de errores se le pasan por alto más fácilmente al que los origina que a otras personas".
- El método clásico de revisión de código es el de inspección visual por parte de revisores diferentes al programador. Este método no es seguro ya que el revisar línea a línea se convierte en un proceso tedioso cuando los archivos son grandes,

con lo que el proceso de revisión puede perder su efectividad debido a la perdida de atención que sufre el revisor cuando lleva un tiempo largo realizando la revisión.

- Existen herramientas automáticas de revisión de código, que en general se basan en la utilización de patrones de error, revisión del estilo del código, etc.; pero que sin embargo al final requieren una revisión manual para verificar la existencia de defectos en el código.

Unas buenas habilidades para la depuración incluyen un conocimiento de las causas comunes de errores y como solucionarlos. Los *patrones de error* [Allen, 2002] son relaciones recurrentes entre errores indicados y errores subyacentes en el programa. El conocimiento de esto patrones y sus síntomas ayuda al programador a identificar rápidamente nuevas apariciones de los errores, y además evitarlos sobre la marcha. Los programadores pueden tardar mucho tiempo en reconocer estos patrones si lo dejamos a su experiencia individual. Pero si identificamos estos patrones y los enseñamos explícitamente, podemos ayudar a mejorar la experiencia de los programadores. Además, una vez aprendidos, permiten una identificación y comunicación explícita y los desarrolladores pueden beneficiarse mutuamente de la experiencia de los demás en depuración, y de este modo, adquirir la destreza más rápido que de cualquier otro modo.

2.9 Creación de un modelo para la buena codificación y depuración

Como dice Allen [Allen, 2002] el diagnóstico de errores software tiene mucho en común con la realización de un experimento científico. En ambos casos, debemos seguir los siguientes pasos:

1. Observar el sistema bajo investigación.
2. Formular una hipótesis para explicar las observaciones.
3. Probar esta hipótesis con nuevos experimentos.
4. Repetir hasta que lleguemos a una hipótesis que explique todo el comportamiento observado.

La parte más importante del proceso de depuración y pruebas (y que muchas veces nos saltamos) es el desarrollo de hipótesis de por qué el sistema se comporta como lo hace. Para hacer esto correctamente, es necesario crear un modelo de cómo cada componente del sistema interactúa con otros. Algunos de estos componentes serán modelados sólo a alto nivel. Otros (los que se consideran causa directa del error) serán modelados con bastante precisión. Por ejemplo, al ver los errores de un proyecto con veinte clases planteamos la hipótesis de que clases es la responsable y nos fijamos en detalle sobre esa clase, sin mirar en profundidad el resto. Cuando se realizan más pruebas, es esencial que el programador actualice su modelo teniendo en cuenta los resultados.

La habilidad de los expertos en cualquier campo no es sólo resultado de la inteligencia pura. También es resultado de la experiencia. La experiencia proporciona muchos ejemplos específicos que se pueden generalizar en patrones. Los expertos aplican estos patrones a nuevas situaciones y así obtienen razonamientos más efectivos. En particular, ignoran detalles irrelevantes más rápidamente y se centran en lo que es importante. Muchas veces es posible identificar *patrones de error* con sus síntomas, causas y soluciones. Si se es consciente de la existencia de un patrón de error, se pueden identificar las ocurrencias de ese patrón más rápidamente y solucionarlo. Examinando estos patrones de error podemos hacer más

que identificarlos, podemos también que prácticas de codificación pueden ayudar a prevenir la aparición de este error en el futuro.

El problema reside en que la creación y el aprendizaje de los patrones de error son complicadas y requieren de un tiempo que muchas veces no se dispone. Ante esto surgen ideas de intercambio de información sobre errores en entornos colaborativos, de tal manera que cualquier programador pueda consultar información un error y si no existe introducir la información que tenga en ese momento.

2.10 Adquisición de competencias para eliminar / evitar defectos en el software

Para conseguir lo planteado en la sección anterior es imprescindible mejorar el proceso de adquisición de habilidades por parte de los futuros ingenieros de software. Sabemos que es muy importante tener programadores con experiencia ya que *la calidad y seguridad del software producido depende inevitablemente de la destreza y experiencia de los programadores involucrados*. Además, *Para formar a más programadores es insuficiente la enseñanza tradicional de conocimiento teórico. Necesitamos transmitir habilidades prácticas en el desarrollo de sistemas robustos, esto lleva años de experiencia* [Allen, 2002]. Esto enlaza directamente con el aprendizaje basado en competencias que se plantea con el Espacio Europeo de Educación Superior y que trata en profundidad el "Libro blanco del Título de grado en Ingeniería Informática" elaborado por un grupo de trabajo de la Conferencia de Directores de Ingeniería Informática y editado por la ANECA [ANECA 2005].

Dentro de la competencia específica definida como "Programación" consideramos fundamental el conocimiento práctico que consiste en la habilidad de diagnosticar eficientemente y subsanar los errores del sistema software. La depuración efectiva está lejos de ser una habilidad trivial. Buscar y eliminar errores ocupa una porción significativa del tiempo de desarrollo en un proyecto software. Si esta tarea se pudiera realizar de forma más eficiente, el software resultante podría ser más fiable y el proceso de desarrollo sería más rápido. Por tanto, es fundamental el aprendizaje no sólo de técnicas de codificación sino también de detección y corrección de errores. El aprendizaje de estas técnicas no es sencillo y para alcanzar un dominio adecuado es imprescindible complementar los conocimientos teóricos con prácticas que ayuden a la buena asimilación de estos conceptos.

2.11 Conclusión

Es necesario un entorno "plug-and-play" que pueda utilizarse desde cualquier ordenador sin necesidad de instalación de software; simplemente conectarse a Internet. Un entorno Web también facilitaría la integración dentro otros sistemas Web pensados para el aprendizaje de la programación y sería un paso decisivo que posibilitaría una mejor adquisición de buenas prácticas de programación.

Un modelo que ha tenido éxito en otros campos es el **aprendizaje basado en errores**. En el campo de la programación se ha explotado poco este enfoque; pese a que los compiladores generan determinados errores a la hora de compilar. Estos **mensajes de error** son utilizados por los programadores para corregir el error concreto de ese programa; pero **no se aprovechan como una oportunidad para aprender** y evitar que este error vuelva a ocurrir.

Por otra parte, los **mensajes de error de los compiladores son excesivamente escuetos y en ocasiones crípticos** por lo que muchas veces es necesario tener una experiencia para

poder utilizar correctamente la información proporcionada por el error y corregirlo de una forma efectiva. Sería muy útil acceder a una base de conocimientos que proporcionase información sobre el mensaje de error: una explicación más amplia, en qué contextos se puede dar el error, cuál es la causa y cómo se puede solucionar. Sería como si un programador novato tuviera un grado suplementario de experiencia.

Si el sistema puede realizar una detección automática de problemas y puede dirigir al desarrollador hacia la información adecuada para saber cual es el problema y poder solucionarlo estaremos reduciendo la necesidad de una comunicación asíncrona entre desarrolladores o entre profesor – estudiante en un entorno de aprendizaje con lo cual los usuarios podrán realizar auténtico trabajo autónomo y con el ritmo adecuado; pero a la vez respaldado por el apoyo de un tutor automático.

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.

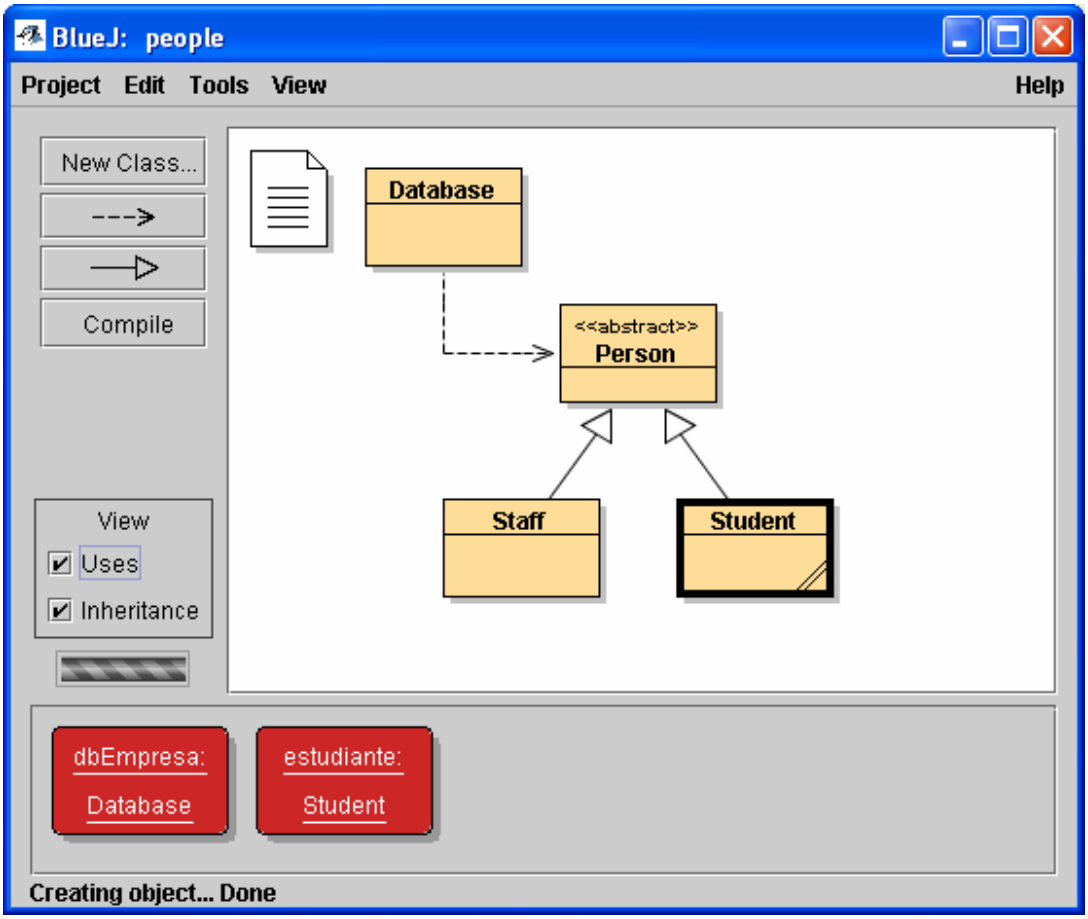


Figura 2. Panel de clases de BlueJ con varios objetos instanciados

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.

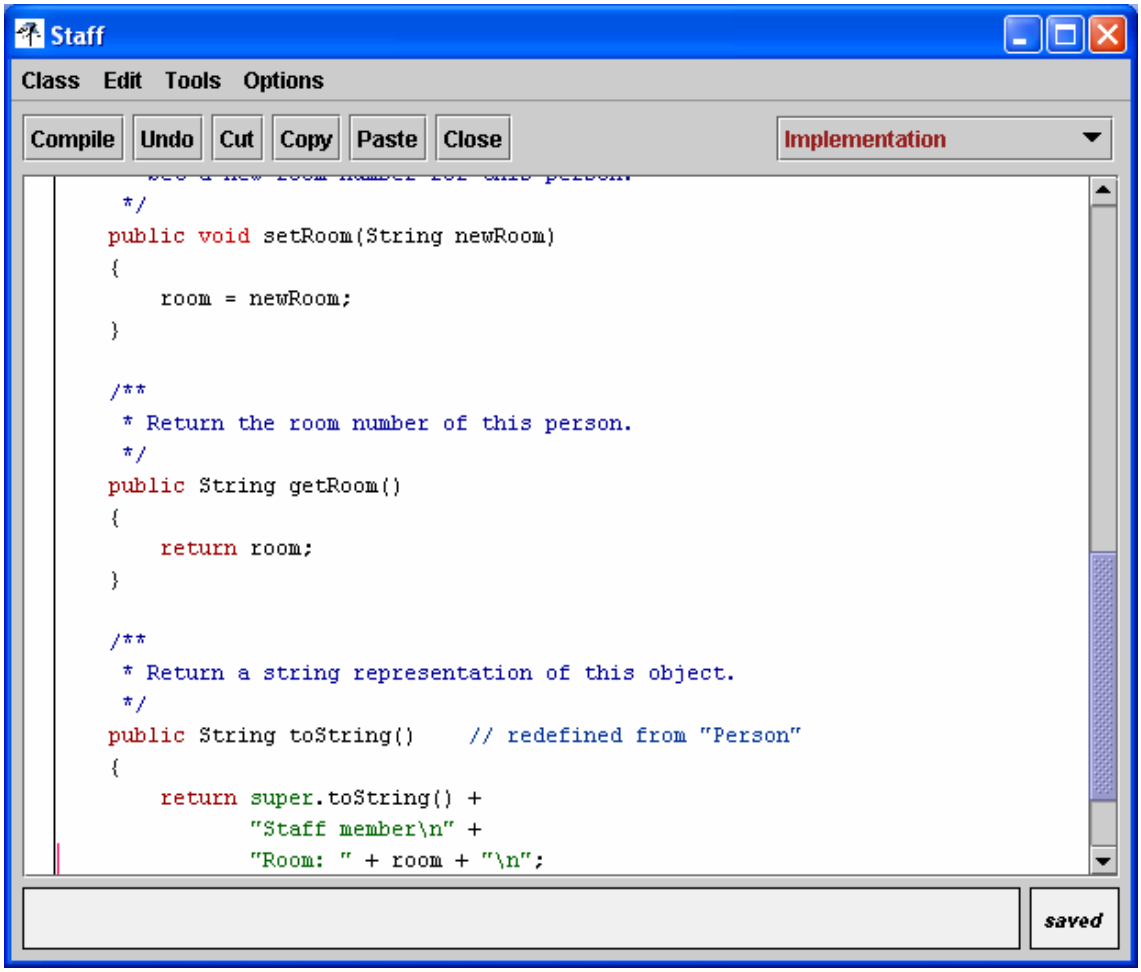


Figura 3. Panel de edición de clases en BlueJ

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.

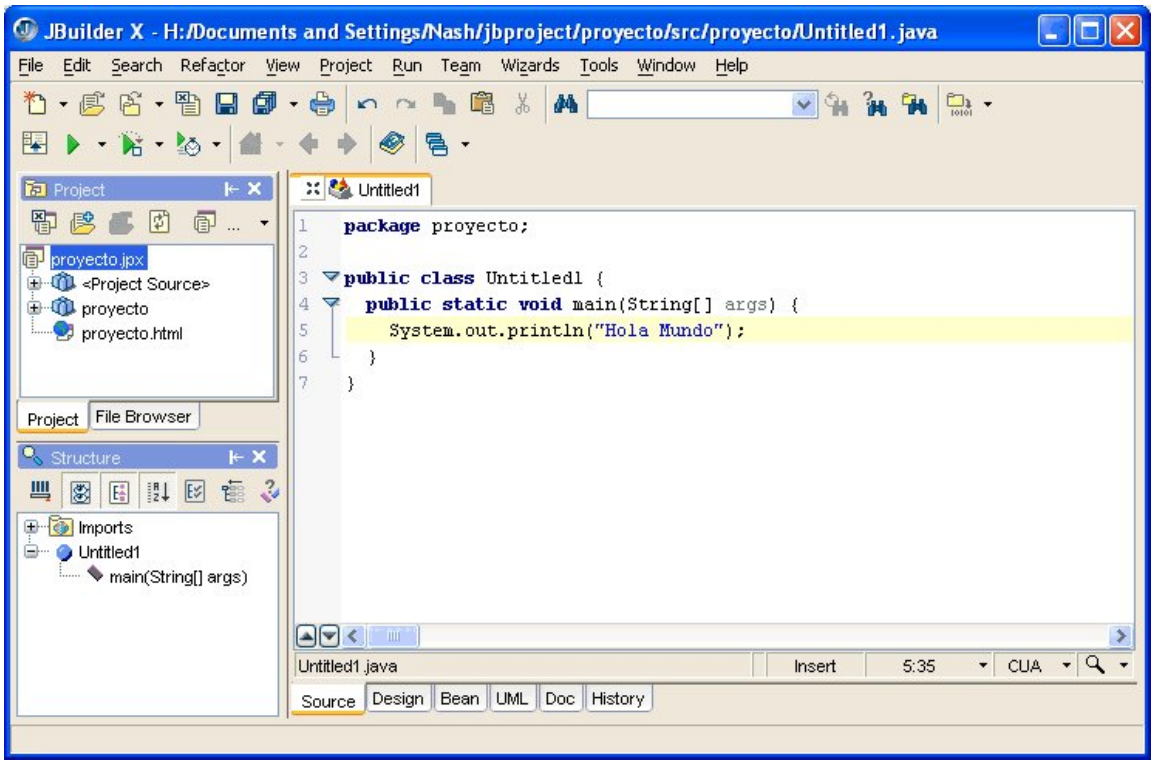


Figura 4. Entorno de desarrollo integrado JBuilder.

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

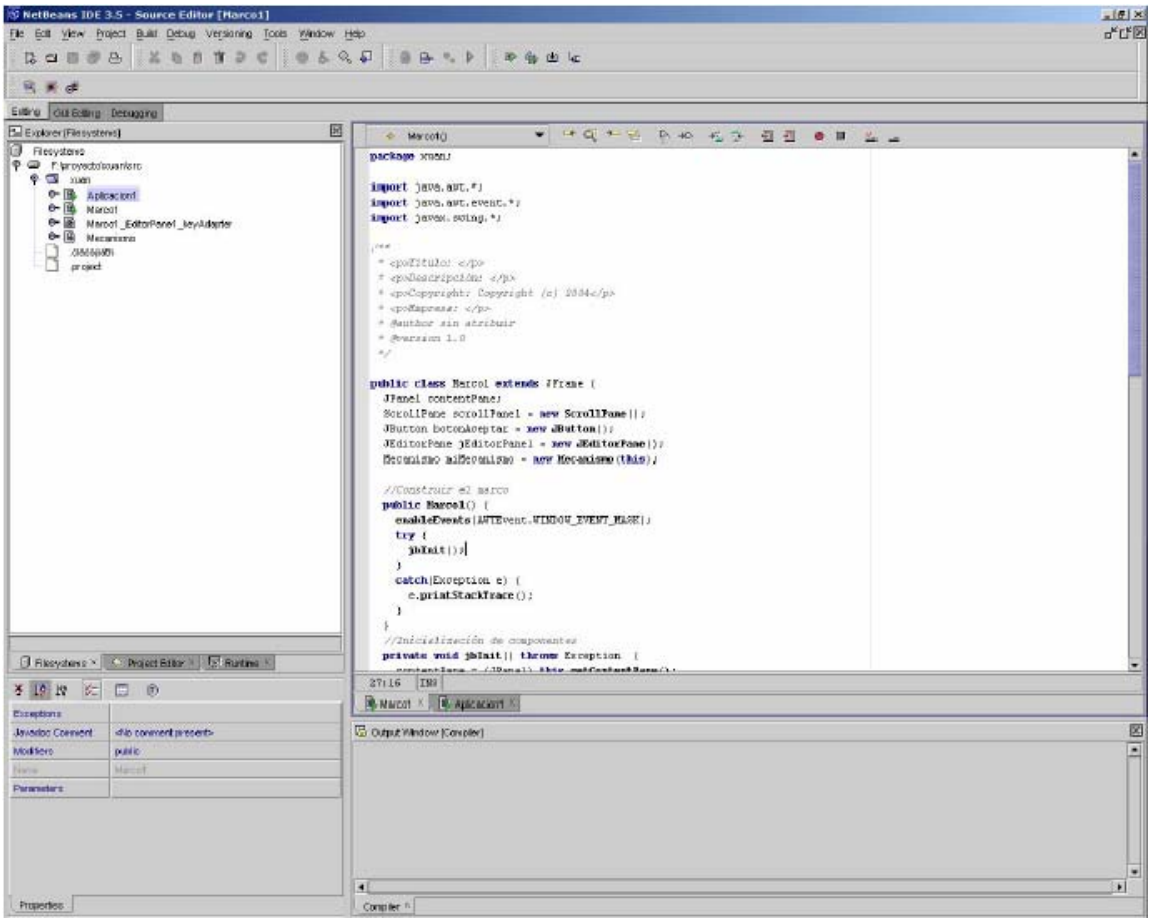


Figura 5. Entorno de desarrollo integrado NetBeans.

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.

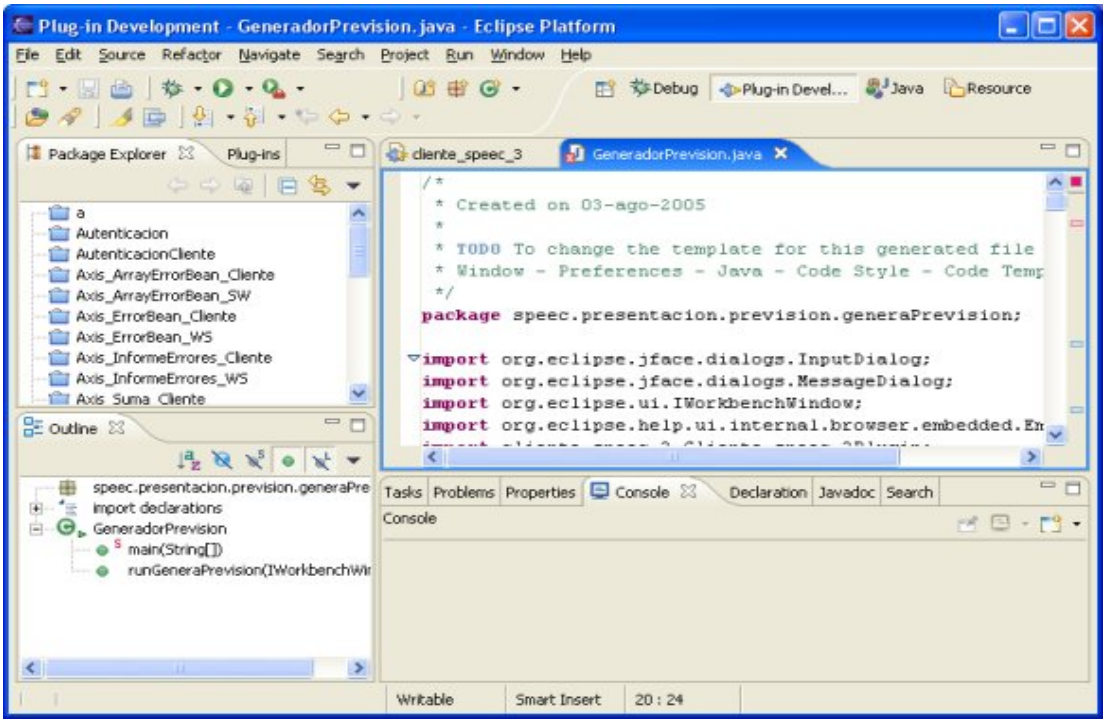


Figura 6. Entorno de desarrollo integrado de Eclipse.

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.

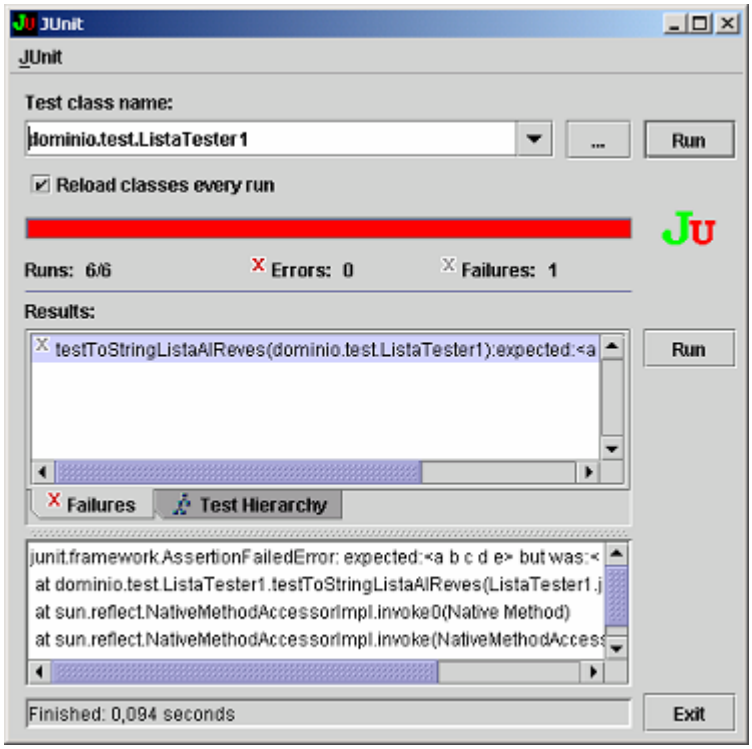


Figura 7. Ejecución de un caso de prueba con JUnit provocando un fallo

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.

Capítulo 4. Requisitos del modelo para la mejora de la calidad de código fuente

En este capítulo definimos los requisitos del modelo para un sistema centrado en la mejora de la calidad del código fuente. Nos basaremos en los objetivos planteados en el Capítulo 1 y en las conclusiones extraídas en el capítulo anterior a partir del estudio de los distintos enfoques actuales y las virtudes y carencias de estos.

Para articular el modelo nos basamos en cuatro ejes fundamentales:

- Entorno de desarrollo, que facilita el trabajo del desarrollador no sólo en condiciones de aprendizaje sino también de trabajo real.
- Utilización de técnicas de procesadores de lenguajes para detectar errores que se almacenarán para permitir un análisis de su evolución.
- Análisis de los errores actuales mediante la historia de compilación y generación de avisos que haga al modelo activo ante la solución y prevención de errores.
- Utilización de los errores como elemento en la mejora del código.

4.1 Contexto del sistema

Sistema de desarrollo de software para la mejora de la calidad del código y prevención de errores de código. Este sistema está pensado para el desarrollador ya conoce los conceptos básicos de programación y del lenguaje que utiliza; pero tiene que enfrentarse a proyectos con un tamaño medio o grande. El número de clases implicadas en estos proyectos es relativamente grande y la dificultad de localización y corrección de los errores que surgen crece ya que estos no son tan triviales como en proyectos pequeños ya que se complica la interrelación entre las clases.

En los siguientes apartados describiremos los requisitos de este sistema.

4.2 Entorno de desarrollo

El entorno de desarrollo es el centro del sistema. Consistirá la interfaz con el usuario desde donde invoca a todas las herramientas necesarias en el desarrollo, comprueba los errores de sus programas y desde donde consulta la ayuda para corregirlos.

4.2.1 Características deseables para un entorno colaborativo de desarrollo de software

En este apartado recogemos las características que un sistema para el desarrollo de software debería incluir. Estas características han sido extraídas tras el análisis de distintos sistema realizadas en el capítulo anterior y servirán para definir las características fundamentales que integrará el entorno en nuestro modelo:

- Edición de código en Web, permite evitar el tener un equipo local un espacio de trabajo y tener que descargar, modificar y volver a subir los archivos del proyecto en desarrollo.
- Posibilidad de corregir código de otras personas, todos los archivos deben poder ser compartidos para que otros componentes del equipo realicen revisiones de código, es muy habitual que uno mismo no vea sus fallos.
- Comunicación entre desarrolladores, los desarrolladores tienen que mantenerse en contacto entre si, cuando el equipo de desarrollo trabaja en el mismo lugar físico esto no tiene problemas; pero cuando trabajamos con equipos virtuales de desarrollo donde cada persona está en un lugar diferente, es necesario que el entorno integre herramientas de comunicación.
- Sistema de foros / votaciones, hay veces en las que es necesario llegar a un acuerdo en el equipo de desarrollo cuando hay varios caminos o alternativas que tomar.
- Repositorio de versiones, poder recuperar versiones anteriores de archivos por si se necesita volver a trabajar empezando desde ellas.
- Espacio compartido, para el intercambio de archivos entre desarrolladores.
- Compilador integrado (para independizar el proyecto de la plataforma)
- Monitorización del proyecto, tener una visión general y rápida de lo que ocurre en el proyecto.
- Gestor de tareas pendientes, ver la lista de tareas que aún no se han finalizado y su estado.
- Calendario, para fijar fechas de reuniones, revisiones, entrega de documentos.
- Recortes de código, de gran utilidad a la hora de tener que hacer algo que previamente ya se hizo sin necesidad de tener que analizar todo el código fuente.

4.2.2 Entorno de programación integrado

Un requisito que debe cumplir el sistema es el de integración de distintas herramientas. Si queremos gestionar proyectos reales de forma eficiente debemos disponer de un entorno que agilice el ciclo del desarrollo de una aplicación: edición, compilación, depuración y corrección de errores, documentación, prueba y también la gestión de los distintos archivos que forman el proyecto. Las herramientas deberían poder reutilizar la información de unas como entrada de otras; por ejemplo, la compilación genera errores que utiliza la herramienta de análisis y el entorno busca la ayuda correspondiente al error al usuario, que cuando está listo para corregirlo ya tiene cargado en el editor la clase donde se localiza el error.

4.2.3 Entorno de programación que permita la edición de código real

Como ya se dijo en el contexto, el entorno debe permitir desarrollar proyectos con un tamaño relativamente grande y además debe permitir incluir experiencias reales de programación. Programas que incluyan varias decenas de clases con distintas instancias y con múltiples relaciones entre ellas, y no sólo pequeños programas con un número limitado de clases. Esto es imprescindible si queremos entrenar a los desarrolladores en la escritura de un código de calidad y que establezcan unos buenos hábitos de programación.

Por otra parte, el entorno debe permitir abordar el desarrollo de programas que den solución a problemas reales que involucrarán distintos bibliotecas del lenguaje utilizado. La definición de un conjunto de paquetes cerrado sobre los que desarrollar programas pueden servir para un nivel inicial; pero una vez que los estudiantes adquieren los conceptos básicos, el limitar la solución a determinados paquetes estándar o realizados a propósito puede limitar los problemas a resolver y la aplicación de habilidades para resolverlos.

Este requisito tiene mucha relación con la que explicaremos posteriormente de soporte para trabajo en grupo del apartado 4.2.5.

4.2.4 Entorno fácil de usar y disponible en cualquier sitio

Se ha comprobado que con frecuencia sistemas que tienen unas buenas características funcionales dejan de ser utilizados por dificultades para su uso. Por esto la facilidad de uso es fundamental para permitir el acceso a la funcionalidad del sistema.

- Facilidad de uso. El entorno no debe de ser un obstáculo para el propósito principal que es la mejora de la calidad del código en el desarrollo de aplicaciones, por tanto debe de ser fácil de manejar y cualquier desarrollador debe poder familiarizarse con él en poco tiempo.
- Evitar la instalación y la configuración. El entorno debe evitar el proceso de instalación y configuración o al menos, si es imprescindible, que sea lo más automatizado posible. Lo ideal sería un sistema donde el programador se pudiese conectar y empezar a programar inmediatamente.
- Portabilidad entre distintos hardware y sistemas operativos. Las organizaciones tienen una gran variedad de sistemas, por tanto es requisito de la portabilidad es importante para cualquier tipo de sistema.

Por otra parte, sería muy interesante que el sistema estuviera disponible en los distintos lugares, si queremos que los desarrolladores puedan tener movilidad y puedan hacer modificaciones en el código no sólo desde la oficina de desarrollo sino también desde otras ubicaciones. Esta disponibilidad incluye lógicamente el requisito, antes explicado, de no tener que instalar la aplicación. Además, la disponibilidad no sólo se refiere al sistema sino que incluye la configuración propia del desarrollador y el acceso a todos los proyectos en desarrollo.

4.2.5 Soporte para trabajo en grupo

El trabajo en equipo es una característica de la programación en el mundo profesional. Hoy en día sólo se entiende el desarrollo de una aplicación de tamaño medio o grande en equipo, por tanto, este sistema también debe soportar el trabajo en grupo. El sistema debería permitir a un grupo de desarrolladores trabajar simultáneamente en diferentes partes del sistema en desarrollo y coordinarse para realizar un proceso de integración adecuado.

El soporte para el trabajo en grupo no sólo es interesante para el proceso de desarrollo de programas; sino que puede ser fundamental para el proceso de mejora de la calidad del código. Por un lado posibilita la realización de inspecciones manuales de código y por otro permite el intercambio de experiencias entre los usuarios. Para facilitar esto el sistema deberá disponer de una base de conocimientos en la que se introduce información por dos vías: el sistema automáticamente introduce información a partir de los errores de los usuarios y todos los usuarios podrán completar esta información incluyendo experiencias y ejemplos respecto a los errores de programación y las posibles soluciones.

Si el sistema proporciona la información adecuada cuando el desarrollador la necesita, esto puede suponer un gran aumento en el rendimiento que no tiene que basarse exclusivamente en su propia experiencia para resolver los problemas que le surjan, sino que puede utilizar la experiencia de otros que se han enfrentado previamente a problemas similares.

4.3 Búsqueda, almacenamiento y visualización de errores mediante técnicas de procesadores de lenguaje

El sistema debe realizar un análisis profundo del código para detectar automáticamente todos los errores posibles; además es importante guardar estos errores y las distintas versiones de los ficheros del proyecto para que se pueda hacer un seguimiento del proceso de creación y la evolución global del proyecto.

4.3.1 Utilización de técnicas de procesadores de lenguaje para la búsqueda de errores

Los compiladores controlan que se cumplan las reglas léxicas, sintácticas y semánticas del lenguaje. Sin embargo, no controlan que se sigan un conjunto de buenas prácticas de programación aceptadas por la comunidad de programadores, muchas veces el no seguimiento de estas prácticas conlleva un riesgo elevado de que haya un error. Esto es lo que se denomina "patrón de error". Utilizaremos procesadores de lenguaje que detecten estos patrones de error para proporcionarle al desarrollador una mayor información sobre todos los posibles errores en el código fuente.

4.3.2 Creación de la historia de compilación

Los compiladores actuales son "máquinas combinacionales", es decir, a partir del archivo fuente de entrada generan una salida consistente en un código objeto si el archivo es correcto o, un conjunto de mensajes de error si el archivo no es correcto. Todo esto es independiente de lo que el programador haya hecho previamente. Sin embargo, es interesante conocer los errores cometidos anteriormente por un programador. Para esto debemos convertir a los compiladores en "máquinas secuenciales", es decir, máquinas con memoria, que guarden un registro de todas las compilaciones realizadas en el proyecto, lo que llamamos historia de compilación. Así se podría estudiar este registro para proporcionar una información más precisa sobre los errores que medidas debería tomar el programador para evitar repetir errores.

4.3.3 Visualización de los errores del código fuente

El sistema como cualquier entorno integrado mostrará los errores resultado de la compilación de los ficheros del proyecto. Pero además de esto mostrará una serie de mensajes de las demás herramientas de análisis del código. También permitirá un acceso

directo a la entrada de la base de conocimientos (ver apartado 4.5.1) donde existe información relacionada con el mensaje concreto, incluyendo ejemplos y una guía para localizar el error que ha generado el mensaje y solucionarlo.

4.3.4 Seguimiento del usuario mediante la historia de trabajo

Para mejorar el proceso de aprendizaje y el proceso de desarrollo de software en si mismo, no sólo hay que revisar los resultados finales; sino que es necesario realizar un seguimiento continuo a lo largo de todo el proceso.

Para realizar este seguimiento el sistema almacena la evolución de los archivos fuente de un proyecto con sus sucesivas versiones y simultáneamente el registro de errores derivado de las compilaciones del estudiante. Este sistema de almacenamiento va acompañado con un sistema de navegación a través de las versiones una historia de trabajo que permite revisar fácilmente el proceso de construcción de la aplicación. Este sistema permite ir a una determinada versión y visualizar simultáneamente el código fuente y el resultado de la compilación sobre ese código.

4.3.5 Detección y almacenamiento de los errores en tiempo de ejecución

Para verificar la corrección del código es imprescindible realizar pruebas ejecutando los módulos del proyecto con distintos valores de entrada. El sistema debe permitir la detección de errores en tiempo de ejecución y guardar un registro de estos errores para poder hacer un seguimiento posterior.

4.4 Análisis de los errores de programación

Los errores no sólo deben poder ser almacenados sino que el sistema debe analizarlos para mostrarle al desarrollador los puntos que tiene que revisar para mejorar la calidad de su código.

4.4.1 Obtención de métricas de errores mediante el análisis de la historia de compilación

El sistema debe permitir la obtención de distintas métricas para visualizar las necesidades de mejora del código. Estas métricas se obtendrán a partir del análisis de la historia de compilación almacenada por el sistema al realizar el análisis de los archivos fuente del proyecto.

El sistema trabajará al menos con métricas de frecuencia y evolución de errores aunque debe estar abierto para incorporar más tipos de métricas. El sistema debe permitir configurar si la métrica debe ser obtenida para un usuario individual o para un grupo. Además, también debe poder seleccionarse que tipos de errores deben incluirse en la métrica, esto permitirá adaptar el análisis a las necesidades de cada usuario.

4.4.2 Generación de avisos personalizados adaptados al perfil de los desarrolladores

A parte del seguimiento manual que gracias a la historia de trabajo se puede realizar sobre el proceso de desarrollo de la aplicación, el sistema debe poder actualizar y usar las métricas en cada compilación para proporcionar al usuario información de forma eficaz.

A partir de la información de la historia de compilación, el sistema generará avisos adaptados al perfil de usuario que le aparecerán al programador mientras está escribiendo

código para prevenir los errores que estaba cometiendo de forma más frecuente hasta el momento.

Es muy importante que el sistema reaccione de forma rápida ante los cambios en las métricas para que el desarrollador pueda aprovechar los avisos y poder prevenir errores mientras está escribiendo el código. Además, normalmente habrá varios usuarios en sesión simultáneamente y un supervisor humano no los podría controlar a la vez. Por tanto, este proceso debe ser realizado por el sistema de forma automática.

4.5 Diseño centrado en el aprendizaje continuo para la mejora partiendo de los errores

El fin último del sistema es que el desarrollador escriba un código fuente de mayor calidad; para ello se dispone de una base de conocimientos asociada a cada error que permite aprender sobre las causas, soluciones y que hacer para evitar el error.

4.5.1 Base de conocimientos con información semántica sobre los errores

El primer paso para poder dar solución a los errores de programación antes de ponerse a depurar el programa, es comprender los mensajes que proporcionan el compilador y las herramientas de análisis estático, y las excepciones lanzadas por los programas en ejecución. Los mensajes de error de los compiladores no facilitan la labor ya que muchas veces son crípticos y ambiguos. Esto causa que sean difíciles de entender para un desarrollador con poca experiencia.

Además, *los compiladores solamente proporcionan síntomas de defectos y debes entender dónde y cual es el problema* [Humphrey, 1997]. Muchas veces es difícil, pese a disponer de la información que les proporciona el compilador sobre los errores, entender y relacionar los síntomas indicados en los mensajes de error con los problemas reales para eliminarlos. Y todavía más difícil es deducir lo que se está haciendo mal para tratar de evitarlos en un futuro.

En los entornos de desarrollo existentes la información que se proporciona al desarrollador no se puede ampliar. Es necesario que el sistema permita incorporar y consultar información adicional, para que el desarrollador comprenda los mensajes de error, el contexto en el que aparece cada mensaje, los problemas reales de los que puede ser un síntoma y la forma más adecuada de solucionar cada uno. Para llevar a cabo esto planteamos una base de conocimientos colaborativa integrada en el sistema de desarrollo.

4.5.2 Colaboración entre los usuarios para completar la información de la base de conocimiento

Esta base de conocimientos utiliza la información generada por las herramientas de análisis del código estático, almacenando los nuevos errores del código cuando aparecen tras la compilación de un usuario, generando nuevas páginas con la información básica de cada error.

A partir de esta información generada automáticamente el sistema se basa en la incorporación de información adicional por parte de los usuarios. La información que añadirán los usuarios serán ejemplos reales donde se haya producido el error, causas del error para un contexto determinado y la forma en la que se ha solucionado, es decir cada usuario aporta su experiencia con el error en cuestión.

4.5.3 Información orientada a la solución y prevención de errores

La información de la base de conocimientos no se debe limitar a informar sobre que significa el mensaje de error y poner ejemplos del mismo, también y sobre todo debe guiar al programador a encontrar una solución al error.

Por otra parte, debe proporcionar información que permita al usuario tomar medidas para evitar que este tipo de errores vuelva a ocurrir.

Capítulo 5. Sistema SICODE

A partir de los requisitos establecidos en el Capítulo 4 y las conclusiones obtenidas en el Capítulo 3, en este capítulo y los siguientes abordamos la descripción de los prototipos implementados. En concreto, en este capítulo establecemos los planteamientos generales del sistema SICODE (SIstema COlaborativo de DEsarrollo), las decisiones básicas que se han tomado en su desarrollo [Pérez 2004a] y una descripción del proceso de desarrollo seguido. Por último, realizaremos una panorámica general del sistema en su conjunto.

5.1 Planteamiento general

El planteamiento es implementar el sistema definido por los requisitos del capítulo anterior para poder evaluar las ventajas proporcionadas en la práctica.

SICODE debe de ser fundamentalmente una herramienta que sirva para el desarrollo de aplicaciones de tamaño medio; para ello integra un conjunto de herramientas que facilitan este trabajo a los desarrolladores. Además, SICODE es un entorno orientado a la mejora de la calidad del código de sus usuarios; para realizar esto la herramienta realizará un exhaustivo examen del código mediante procesadores de lenguaje y analizará los resultados de estas herramientas para proporcionar a los usuarios información que le permita evitar errores futuros.

5.2 Decisiones básicas de diseño

A la hora de desarrollar los distintos módulos del sistema se han tomado varias decisiones que condicionan el esquema general del sistema. En los siguientes apartados se hablará de tres decisiones que tienen gran influencia sobre el resto del sistema: el lenguaje de programación al que se dará soporte, la arquitectura general del sistema y el proceso de mejora del código centrado en los errores.

5.2.1 Lenguaje que soportará el entorno

Discutiremos en este apartado la elección del lenguaje de programación que soportará el entorno que desarrollaremos. El diseño general del sistema es que sea un entorno abierto a cualquier lenguaje, ya que esto puede conllevar muchas ventajas como reseña el Dr. Labra [Labra 2003]; sin embargo, en la construcción de los prototipos de evaluación del sistema hemos decidido decantarnos por un lenguaje; aunque siempre dejando abierta la posibilidad de una configuración que permita el uso de otros.

Requisitos del lenguaje de programación

Han sido muchos los artículos que han estudiado los requisitos necesarios de un buen lenguaje de programación en un contexto general. Sin embargo, un lenguaje no es bueno o malo en si mismo sino que depende del propósito para el que se aplique; por tanto, debemos analizar los requisitos en el contexto de nuestro sistema:

- El primer objetivo general de nuestro entorno es la mejora de la calidad del código escrito por los programadores, entendiendo por calidad de código que haga lo que tenga que hacer y sea fácilmente legible y mantenible. Por tanto, orientaremos los requisitos más hacia la calidad del código fuente que hacia el rendimiento o la optimización del código objeto generado por el entorno de compilación del lenguaje.
- El segundo propósito general es tratar de favorecer la transición de estudiantes o desarrolladores con poca experiencia desde pequeños proyectos a proyectos de software de mayor tamaño. Para ello el sistema debe disponer de herramientas avanzadas y que faciliten la colaboración por varios programadores.
- El tercer objetivo es disponer de un lenguaje que facilite la búsqueda de errores en el código fuente.

Teniendo en cuenta estos objetivos y los requisitos que plantea Kölling [Kölling 1999c] para los lenguajes para la enseñanza de la programación, planteamos los siguientes requisitos que nos guiarán en la búsqueda del lenguaje de programación más adecuado:

1. **Orientación a objetos pura**. El lenguaje que elijamos debe de ser orientado a objetos ya que todas las técnicas y métodos actuales se sustentan sobre este paradigma; por tanto, debemos crear el modelo mental correcto y entrenar a los programadores en este paradigma y es la única forma de que nuestro entorno pueda ser capaz de soportar proyectos reales.

   Además, el lenguaje debería ser orientado a objetos puro y no un lenguaje híbrido. Hay mucha gente que argumenta que los lenguajes híbridos permiten una transición más fácil desde lenguajes que siguen el paradigma estructurado como C. El problema de los lenguajes híbridos es que no fomentan el cambio de estilo de los programadores con experiencia previa en otros paradigmas. Al contrario, permiten seguir escribiendo programas creyendo que son orientados a objeto y sin embargo olvidarse de los conceptos clave. Los programadores, pueden considerar que cuando un programa compila sin errores ya está bien programado. De la misma forma, consideran que cuando el compilador de C++ no da errores ya tienen un programa orientado a objetos, lo cual puede ser totalmente falso. Esto puede ser un inconveniente a la hora de motivar a los programadores a mejorar su calidad de código. El lenguaje debe ser el instrumento que fuerce a los alumnos a escribir un buen código orientado a objetos y es la base para evitar futuros defectos en el software.

2. **Seguridad**. El principio de seguridad consiste en que se detecten el mayor número de errores posible bien por el compilador o por el entorno de ejecución. Más aún, deberían detectarse lo antes posible y deberían proporcionarse mensajes sobre la causa de estos. Por otra parte, deberían evitarse las construcciones del lenguaje que sean propensas a errores.

   En la línea de este requisito están los lenguajes fuertemente tipados. Lenguajes como C++ no son fuertemente tipados y pueden producirse errores en tiempo de ejecución difíciles de encontrar. Los lenguajes con tipos dinámicos como Smalltalk

tienen inconvenientes similares. Todo esto provoca que el punto de localización del error esté muy lejano de la fuente del problema y sea difícil asociarlas y por tanto, comprender y eliminar el error.

Otra característica que también proporciona seguridad es la comprobación de los límites de un array o la eliminación de construcciones problemáticas como los punteros explícitos, varios lenguajes permiten programar sin utilizarlos. Muchas veces se utiliza el rendimiento como argumento en contra de estas características, sin embargo, cada vez prevalece más la escritura de un código de alta calidad, libre de errores y que permita un mantenimiento simple, que el rendimiento puro en tiempo de ejecución de una aplicación.

3. **Alto nivel**. El programador debería poder despreocuparse de la estructura interna del sistema. El compilador y el entorno de ejecución pueden encargarse de esta tarea. Un claro ejemplo que va en contra de este requisito es, la gestión explícita de la memoria dinámica por parte del programador. Poner la gestión de memoria dinámica en manos de un programador principiante es una fuente segura de frustraciones ya que los errores de punteros perdidos, doble liberación u otras formas de corrupción de memoria son uno de los tipos de errores de los más difíciles de depurar. Todo esto se puede evitar si se utiliza un sistema de recolección de basura. La utilización de estos sistemas se considera tan positiva que varios lenguajes los incorporan en sus entornos de producción. De nuevo, está el balance entre el rendimiento y la calidad del código que cada vez se inclina más hacia el segundo planteamiento.
4. **Sintaxis legible**. La sintaxis debería de ser consistente y fácil de leer. Este parece un requisito menor; pero hay varias razones que lo justifican. Un programa fácilmente legible también es más fácil de corregir. Es verdad que, una sintaxis más legible no garantiza la corrección del código; sin embargo, determinados casos de error se producen porque un programador no entiende otra parte del código preexistente. Si asumimos que la legibilidad incrementa la comprensibilidad, entonces es clara su importancia.

   Uno de los aspectos más importantes es que el lenguaje favorezca las palabras reservadas sobre los símbolos. Las palabras son mucho más intuitivas que los símbolos. Esto hace que el lenguaje sea más fácil de aprender y sus programas más fáciles de leer. El C++ es el más claro ejemplo de cómo el uso de símbolos puede ir en detrimento de la claridad, en este lenguaje se ha puesto mucho énfasis en tener un número lo más pequeño posible de palabras reservadas. Esto conduce a que haya palabras reservadas que se utilicen en distintas construcciones con distintos significado semántico; por ejemplo: static o el uso de símbolos como "= 0" después de una cabecera (lo cual podría sugerir una asignación) para indicar que es un método abstracto. Incluso programadores con experiencia tienen problemas con esto.

   Mucha gente para aprender un lenguaje o programa lee directamente código fuente de programas bien sean ejemplos o programas para modificar; no lee el texto explicativo de los programas o los comentarios que, si existen, muchas veces son incompletos. Este aprendizaje basado en ejemplos es una forma de aprender muy potente que todos hemos aplicado alguna vez.

   Otro aspecto de la legibilidad es la consistencia. La misma sintaxis debería ser usada para la misma semántica y diferente sintaxis para diferentes semánticas. Actualmente los programadores invierten mucho más tiempo leyendo código para refactorizarlo, corregirlo o ampliarlo que escribiendo código nuevo; si tenemos en

cuenta esto a la hora de elegir un lenguaje no debemos quedarnos con el que permita escribir código de forma más rápida sino con el que proporcione la forma más clara de lectura. Además, a la hora de escribir código los programadores deberían pensar en el futuro lector de este código y hacer las cosas lo más evidentes posibles y cuando no sea así añadir un buen comentario.

5. **Fácil transición**. Cuando se plantea un lenguaje para el aprendizaje de la programación, uno de los factores que hay que tener en cuenta es que las competencias adquiridas sean relevantes para el futuro profesional de los programadores. Aunque no hay que confundir el hecho de aprender un lenguaje de programación con aprender los conceptos, técnicas y habilidades necesarias para programar. Por tanto, la importancia está en los principios que sigue ese lenguaje y no tanto en los detalles. Esto proporciona a los programadores una gran versatilidad y capacidad de adaptación a futuros lenguajes, cosa muy necesaria con la velocidad actual de cambio de la industria del software. Por tanto, los conocimientos y habilidades adquiridas deben de ser fácilmente transferibles a otras situaciones que se darán en el futuro.
6. **Mecanismos para garantizar la corrección**. Es importante adiestrar a los programadores para que integren técnicas y buenas prácticas de ingeniería del software que les ayudarán a obtener un código de alta calidad. Entre otras técnicas están aquellas que permiten integrar en el código comprobaciones sobre su validez como técnicas de programación por contrato: precondiciones, postcondiciones e invariantes. Es muy importante que el lenguaje permita integrar mediante asertos estas técnicas: esto permite utilizarlos desde el principio, e integrarlos en el propio proceso de programación.

Razones de la elección del lenguaje Java como lenguaje para los prototipos

Teniendo en cuenta los requisitos generales enunciados en el apartado anterior se ha elegido Java como lenguaje soportado por los prototipos desarrollados.

Java [Arnold 1996] es uno de lenguajes orientados a objetos más populares actualmente. A continuación estudiamos este lenguaje bajo la óptica de los requisitos anteriores.

1. **Orientación a objetos pura**. Java es básicamente un lenguaje orientado a objetos puro. Todo el código es parte de una clase y las clases son la unidad estructural de código. Con la excepción de los métodos estáticos con el método main a la cabeza necesario para lanzar la ejecución.
2. **Seguridad**. Java es un lenguaje fuertemente tipado que combina el chequeo estático con el dinámico. La mayoría de las construcciones se comprueban en compilación, mientras que alguna se comprueba en tiempo de ejecución pudiendo lanzar una excepción. Java fuerza la comprobación de que los índices al acceder a un array permanezcan dentro de las dimensiones definidas para este.

   Un problema que puede surgir en relación a esto en Java es el tema relativo a la implementación de genericidad como base para las colecciones. En Java el mecanismo de genericidad se implementa mediante el uso de clases genéricas y el polimorfismo; lo cual no permite comprobar los tipos de los objetos que metemos en una colección y no proporciona seguridad al extraer un objeto de esta teniendo que hacer un cast a ciegas. Sin embargo, esta desventaja ha sido eliminada en la versión 1.5 de Java 2 con la introducción de plantillas (templates) que permiten

parametrizar el tipo de los elementos de una colección y así dan la posibilidad al compilador realizar una comprobación estática de tipos.

3. **Alto nivel**. Las construcciones en este lenguaje permiten un alto nivel de abstracción permitiendo al programador despreocuparse por los detalles de bajo nivel. La faceta más destacada es la eliminación de todo el trabajo explícito con punteros y la utilización de un recolector de basura que evita al programador tener que trabajar manualmente con la memoria dinámica.
4. **Sintaxis legible**. La sintaxis es un punto débil de Java. Esta sintaxis estilo C reduce la legibilidad de varias formas: en primer lugar la preferencia por los símbolos respecto a las palabras clave, hace que la lectura de los programas sea a veces poco intuitiva. En segundo lugar el estilo flexible de la sintaxis; permite que los programadores utilicen distintos estilos que si se mezclan dificultan el mantenimiento del código fuente. Esto obliga a la introducción de guías de estilo, que son susceptibles de no ser cumplidas, ya que el compilador no comprueba estas guías.
5. **Fácil transición**. El propio lenguaje Java está siendo cada vez más utilizado en la industria con lo que realmente no sería necesaria ninguna transición para el estado actual de la industria. De todas formas, Java recoge los principios modernos de desarrollo de software con lo que facilita la transición a otros lenguajes más específicos o a lenguajes que puedan surgir en el futuro.
6. **Mecanismos para garantizar la corrección**. A partir de la versión de Java 2 1.4 el lenguaje soporta asertos que permiten implementar pre, postcondiciones e invariantes de clase, con lo que podemos realizar una programación por contrato integrada dentro del proceso de desarrollo de una aplicación en este lenguaje.

Vemos que el lenguaje Java cumple de forma muy destacada los requisitos exigidos, en líneas generales vemos que es un lenguaje de alto nivel que implementa de forma clara los principios de la orientación a objetos, en el que se proporciona seguridad al programador en la detección de errores que proporciona mecanismos para garantizar la corrección y que es un lenguaje cada vez más aceptado en la industria y de todas formas permite dar el salto de forma sencilla a otros lenguajes orientados a objetos. Los problemas pueden venir por una sintaxis heredada del C que en algunos casos puede resultar difícil de leer, de todas formas esto puede ser atenuado estableciendo un buen estilo a la hora de escribir código que podrá ser reforzado por un entorno que incluya un buen editor de código.

A estas características podemos añadir tres más que son relevantes para nuestro entorno:

- Gracias a la comprobación estática de tipos es posible la captura de muchos errores en el código. Por otro lado, la utilización de excepciones como forma estándar notificación de errores en tiempo de ejecución permite su captura y tratamiento automáticos.
- Existen gran cantidad de herramientas para este lenguaje que permiten la comprobación del código de forma estática y que permiten reforzar las comprobaciones del compilador de código.
- Actualmente es el lenguaje que se emplea en la mayoría de las asignaturas de programación en la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo, sobre los cuales realizaremos la evaluación del prototipo.

5.2.2 Arquitectura del sistema: centrado en Internet

Actualmente estamos bastante cerca de aquella utopía planteada por Sun a finales de los 90: el "net – computer". Ya existen muchas aplicaciones donde el ordenador local debe disponer de unos recursos mínimos básicamente un navegador Web y una conexión a Internet, ya que el almacenamiento y el procesamiento lo realiza la o las máquinas remotas al otro lado de la red. Un ejemplo lo tenemos en el correo de Google (gmail[3]) no consiste en un simple lector Web de correo electrónico sino que está pensado para que los correo queden almacenados siempre en el servidor; y más aún que no haga falta borrar ninguno. En el campo del desarrollo de software tenemos alguna experiencia en esta línea como es Sourceforge[4]. Sourceforge es un gran "portal" para proyectos. Permite gestionar un proyecto software completo desde la ideas, el reclutamiento de desarrolladores, el desarrollo incluido el almacenamiento de la versión común de los fuentes, y el lanzamiento para los usuarios. Como servicios complementarios de comunicación incluye listas de correo de distintos tipos.

Hemos tomado la decisión de que SICODE vamos un poco más allá que SourceForge Internet no sólo permitirá que los desarrolladores puedan conectarse para acceder a la versión compartida de los archivos sino que la edición y la compilación también se realizarán en red.

5.2.3 Los errores centran el proceso de mejora del código

SICODE plantea un proceso de mejora de código no como algo teórico, donde se estudian los posibles defectos que pueden surgir al escribir código en un lenguaje determinado; sino como algo totalmente práctico, desde la idea de que "a programar se aprende programando" y que la experiencia es muy valiosa, SICODE es un sistema que permite desarrollar aplicaciones; pero en el momento en el que se detecta un posible error SICODE se convierte en **una herramienta que permite aprender de los errores propios y de la experiencia de los demás programadores**.

Para que esto funcione de forma adecuada debemos tomar varias precauciones:

- No debemos conformarnos con los errores que puede proporcionar un compilador, hay que realizar un análisis más profundo. Por eso recurrimos a otro tipo de procesadores de lenguaje, los **analizadores estáticos**, y los usamos en combinación con el compilador. De esta forma, podemos realizar un análisis más profundo del código y detectar distintos tipos de problemas.
- **Historia de compilación**. No sólo mostramos al usuario el resultado del análisis del código sino que lo guardamos para posteriormente procesarlo.
- **Historia activa**. Muchas veces no sólo tiene interés ver el estado actual de un proyecto, puede tener incluso, más interés ver la evolución de un proyecto. Esto es lo que nos proporciona la historia activa poder recorrer y comparar distintas versiones de un proyecto.
- **Análisis de errores de la historia de compilación**, el sistema calcula estadísticas sobre la información contenida en la historia de compilación: errores más frecuentes, evolución de errores. Esto permite a los desarrolladores darse cuenta de forma más fácil de cuales son sus puntos débiles y fuertes.

[3] Se puede acceder a esta aplicación de correo electrónico en http://www.gmail.com

[4] SourceForge: http://www.sourceforge.net

- **Base de conocimientos colaborativa**. Es la clave para que el desarrollador no vuelva a repetir errores y la forma de aprender de la experiencia de otros programadores ya que todos apartan su experiencia añadiendo casos concretos o ejemplos en la base de conocimientos y creando relaciones entre errores que pueden tener puntos en común.

5.3 Proceso de desarrollo del sistema

En los siguientes apartados describimos a grandes rasgos las fases de desarrollo del sistema SICODE.

5.3.1 Planteamiento iterativo

En el capítulo anterior se plantearon los requisitos del sistema; sin embargo, estos requisitos se han obtenido de forma iterativa. Dentro de una línea general se han planteado unos requisitos y se ha implementado un prototipo, que se ha evaluado para a partir de esto refinar los requisitos del sistema y construir nuevos prototipos.

5.3.2 Prototipo Compilador Web SICODE

Tras una primera obtención de requisitos se ha implementado el Compilador Web SICODE que pone en práctica los requisitos fundamentales del sistema y permite evaluar cómo funcionan todos integrados.

5.3.3 División en subsistemas

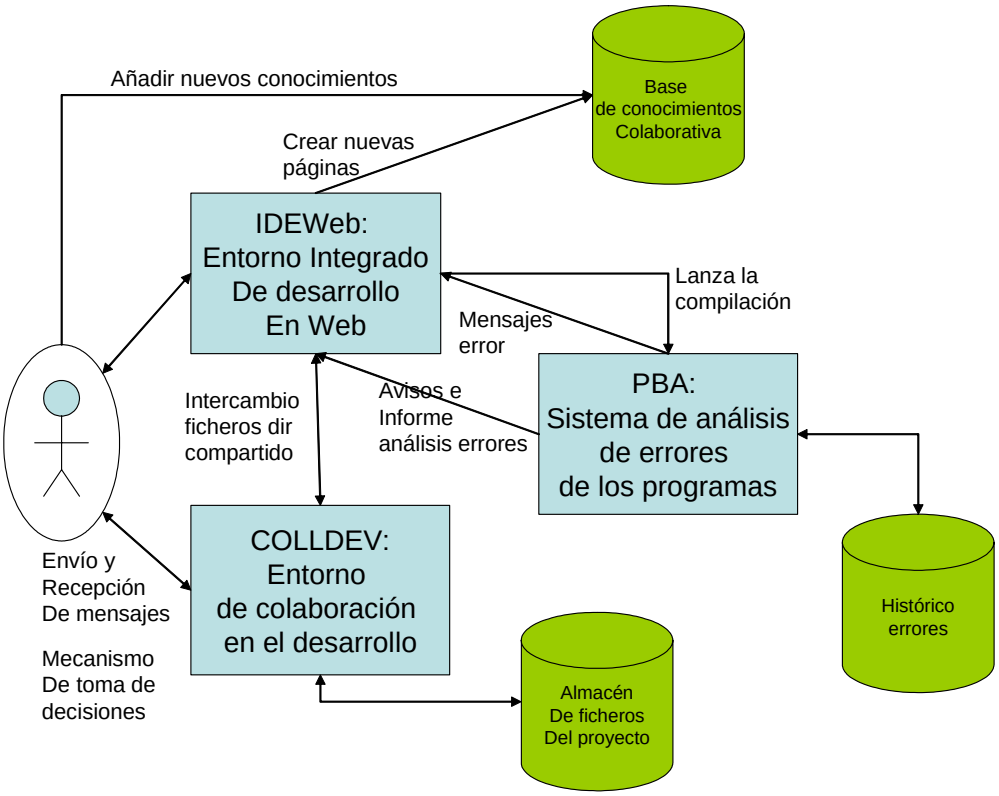


Figura 8. Diagrama de bloques que representa los subsistemas y la interacción entre ellos y con el desarrollador

Una vez desarrollado el primer prototipo, se decidió dividir el sistema en varios subsistemas que se centraran en diferentes aspectos, de forma que estos fueran modulares e independientes. Una vez concluidos estos prototipos podrían unirse para resolver el problema global y poder formar una herramienta útil para los profesores y los alumnos. A continuación se presenta un breve resumen de las tareas que realiza cada uno de los subsistemas.

- **Entorno de desarrollo integrado en Web (IDEWeb)**. Entorno de edición, compilación y depuración sobre Web y base de conocimientos colaborativa que permite introducir ayuda sobre los errores, buenas prácticas y en general ayuda a los errores de programación (errores de compilación y excepciones en ejecución) de forma colaborativa y una forma automática de enlazar esto con cada uno de los errores obtenidos en una compilación. Este es el módulo del sistema con el que el usuario va a interactuar la mayor parte del tiempo.
- **Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV, Collaborative Development)**: Sistema colaborativo que gestiona grupos de trabajo, maneja los usuarios, el grupo al que pertenecen y se encarga de gestionar las sesiones que abren para interactuar con el sistema, así como facilitar la coordinación entre los miembros del grupos. Gestiona el almacenamiento compartido del proyecto y permite el trabajo simultáneo de varios usuarios con los archivos, implementa un control de versiones para realizar un seguimiento del proceso de desarrollo mediante una historia activa del trabajo.
- **Sistema de análisis de errores de programas (PBA, Program Bug Analysis)**. Sistema de gestión de errores tanto en tiempo de compilación, como en tiempo de ejecución. El tratamiento incluye captura, clasificación, almacenamiento y procesamiento de los errores. El sistema debe de ser capaz de realizar estadísticas sobre los errores, para facilitar a los usuarios la comprensión del tipo de errores que comete.

5.4 Una panorámica general

En este apartado queremos reflejar un escenario típico de interacción entre un desarrollador y el sistema SICODE.

5.4.1 Requisitos iniciales del sistema

Para comenzar a trabajar con el sistema SICODE el desarrollador sólo debe disponer en su máquina local de un navegador Web con el plug-in de Java para poder ejecutar el applet que soporta el editor de código ampliado con el coloreado de sintaxis; no es necesario haber instalado ningún software especial.

5.4.2 Entrando en sesión

Para trabajar en el sistema, el desarrollador, debe disponer de un usuario registrado que le permita identificarse y autenticarse. Si se cumple esta condición el desarrollador simplemente se conectará a la dirección Web del sistema e introducirá su nombre de usuario y su contraseña; si estos datos son correctos el sistema abrirá una nueva sesión para el usuario.

5.4.3 Preparación para empezar a trabajar con el código fuente del proyecto

El sistema también tiene información sobre los grupos de trabajo y los proyectos que pertenecen a cada uno de ellos; por tanto el usuario al iniciar la sesión tendrá disponibles todos los proyectos que pertenecen a su grupo de trabajo.

Antes de empezar a trabajar sobre ellos el usuario debería comprobar que trabajo han realizado el resto de miembros del grupo sobre el proyecto; esto puede comprobarlo utilizando las opciones de "directorio de grupo" dentro de "Espacios de proyecto" del Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV, Collaborative Development).

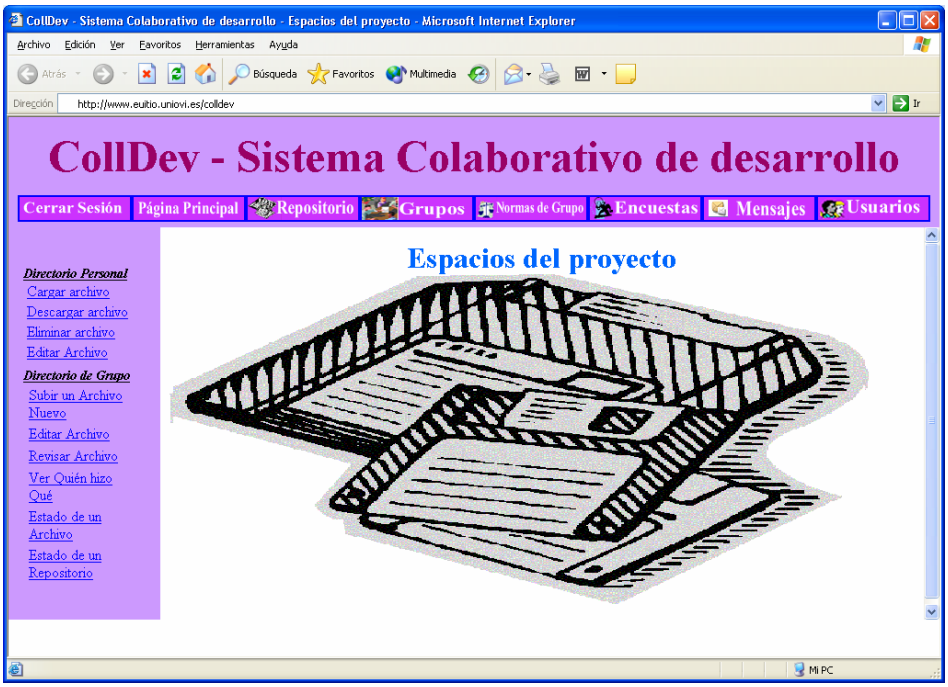


Figura 9. Interfaz del Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV)

El desarrollador deberá actualizar en su directorio personal las versiones de los archivos del proyecto para disponer de las últimas y empezar a modificar los archivos correspondientes. El sistema se encarga de comunicarse con un sistema de control de versiones (CVS) para permitir que los desarrolladores puedan editar concurrentemente cualquier archivo del proyecto y juntar los cambios al final del trabajo, sin necesidad de que un desarrollador espere a que otro termine.

5.4.4 Comunicación entre los desarrolladores

Muchas veces es necesario que los desarrolladores se comuniquen decisiones que toman en la escritura de código o en la implementación de determinada parte del diseño de una aplicación. El Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV, Collaborative Development) proporciona una forma sencilla de comunicación interna en formato texto entre los distintos miembros del grupo de desarrollo de un proyecto.

5.4.5 Edición del código fuente de un archivo del proyecto

Para editar un archivo concreto de nuestro proyecto podemos cargarlo desde el espacio de trabajo personal utilizando el gestor de archivos del Entorno de desarrollo integrado en Web (IDEWeb). Este subsistema constituye un entorno de desarrollo que integra distintas

herramientas necesarias para el desarrollo; pero con una interfaz Web, permitiendo su utilización desde cualquier sistema que tenga disponible un navegador.

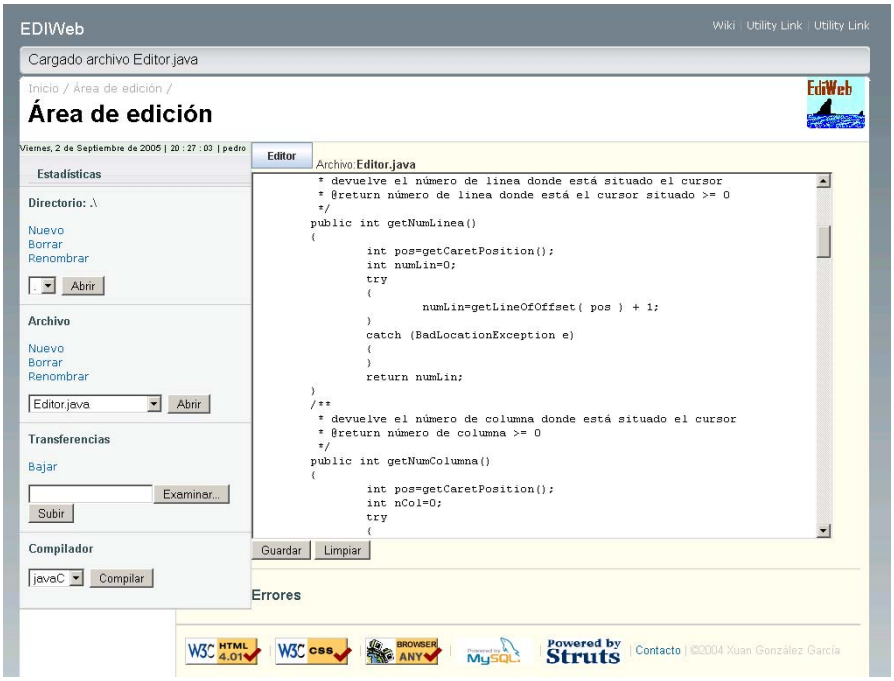


Figura 10. Vista del área de trabajo de usuario en el entorno IDEWeb

El área de trabajo de IDEWeb dispone de distintas opciones de gestión de los archivos del proyecto, además de las opciones de configuración de la compilación remota. Dispone de una gran área de edición donde se muestra el código fuente del archivo que queremos modificar y permite realizar sobre él los cambios que el desarrollador estime conveniente.

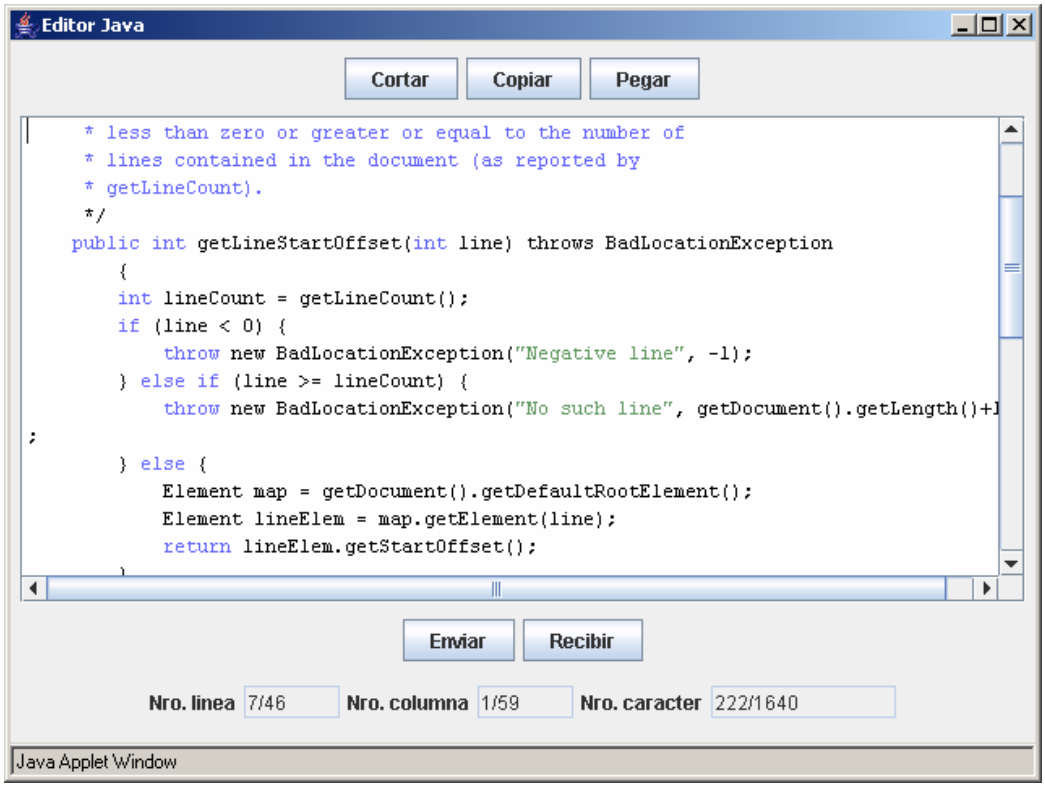


Figura 11. Applet Editor del IDEWeb

Para facilitar la edición de archivos fuente, IDEWeb dispone de un applet editor que dispone de coloreado de sintaxis de los archivos en Java y su forma de trabajar es más parecida a los editores estándar ya que se puede utilizar la tecla de tabulación.

5.4.6 Compilación

Desde IDEWeb se puede invocar una unidad de compilación perteneciente al Sistema de análisis de errores de programas (Program Bug Analysis) PBA. Las unidades de compilación están configuradas por los desarrolladores y especifican las herramientas que se van a utilizar para analizar el código fuente, las opciones de cada una de esta herramientas y el orden en el que se utilizan. Las unidades de configuración utilizan archivos XML compatibles con el estándar de Ant para configurar esto.

Las unidades de compilación proporcionan gran flexibilidad al sistema ya que se puede utilizar simplemente el compilador estándar de Java o se pueden combinar varias herramientas de análisis sintáctico para buscar problemas en el código de forma más exhaustiva.

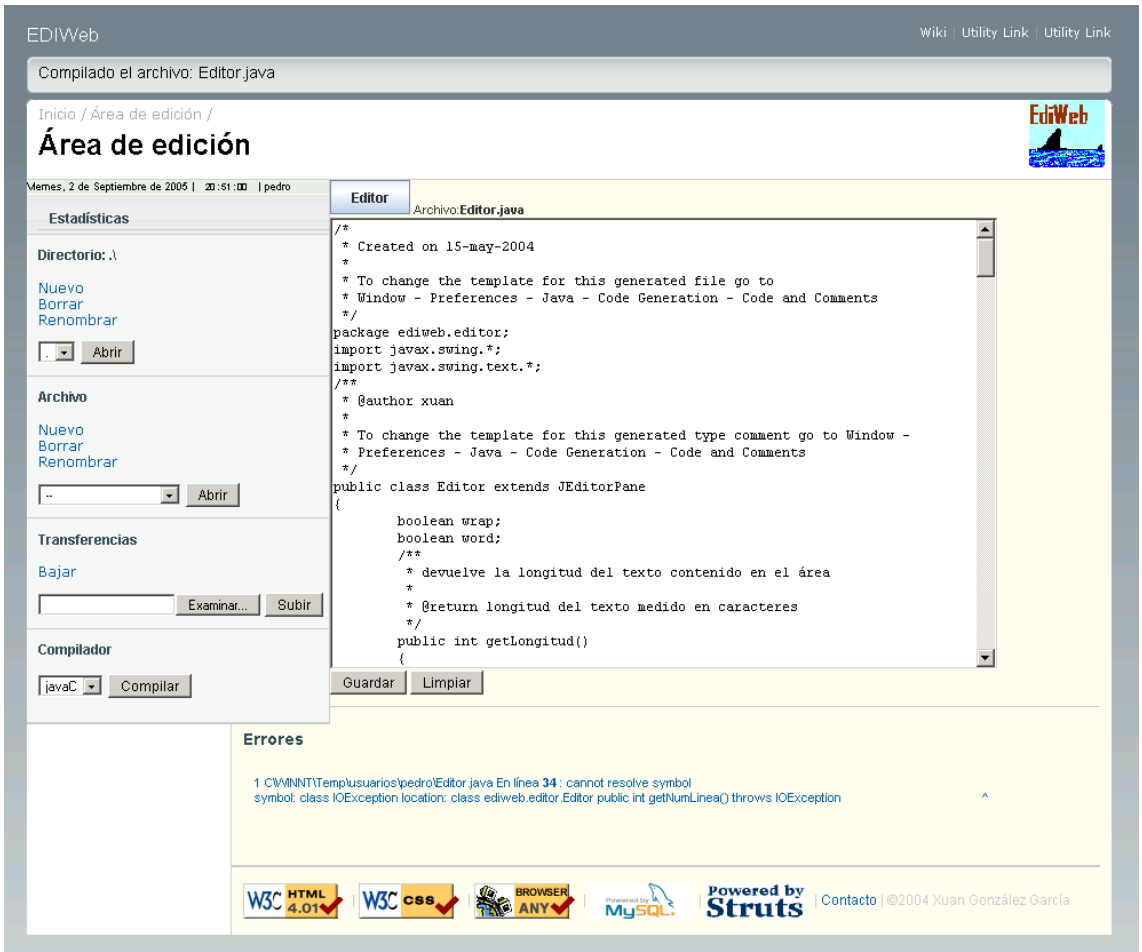


Figura 12. IDEWeb mostrando un mensaje de error después de compilar

El sistema PBA devuelve una vez que a finalizado su trabajo todos los mensajes de error a IDEWeb para que el entorno se los muestre al desarrollador y pueda estudiarlos y repararlos.

5.4.7 Búsqueda en la base de conocimientos para reparar un error

El sistema dispone de una base de conocimientos que permite acceder a información ampliada sobre el error, sus posibles causas, los distintos contextos en los que se puede dar y las alternativas para darles solución. La base de conocimientos también incluye hiperenlaces entre mensajes de error que tengan relación.

El desarrollador puede acceder a la información concreta sobre el error en la base de conocimientos utilizando el hiperenlace que tiene el propio mensaje de error que muestra IDEWeb.

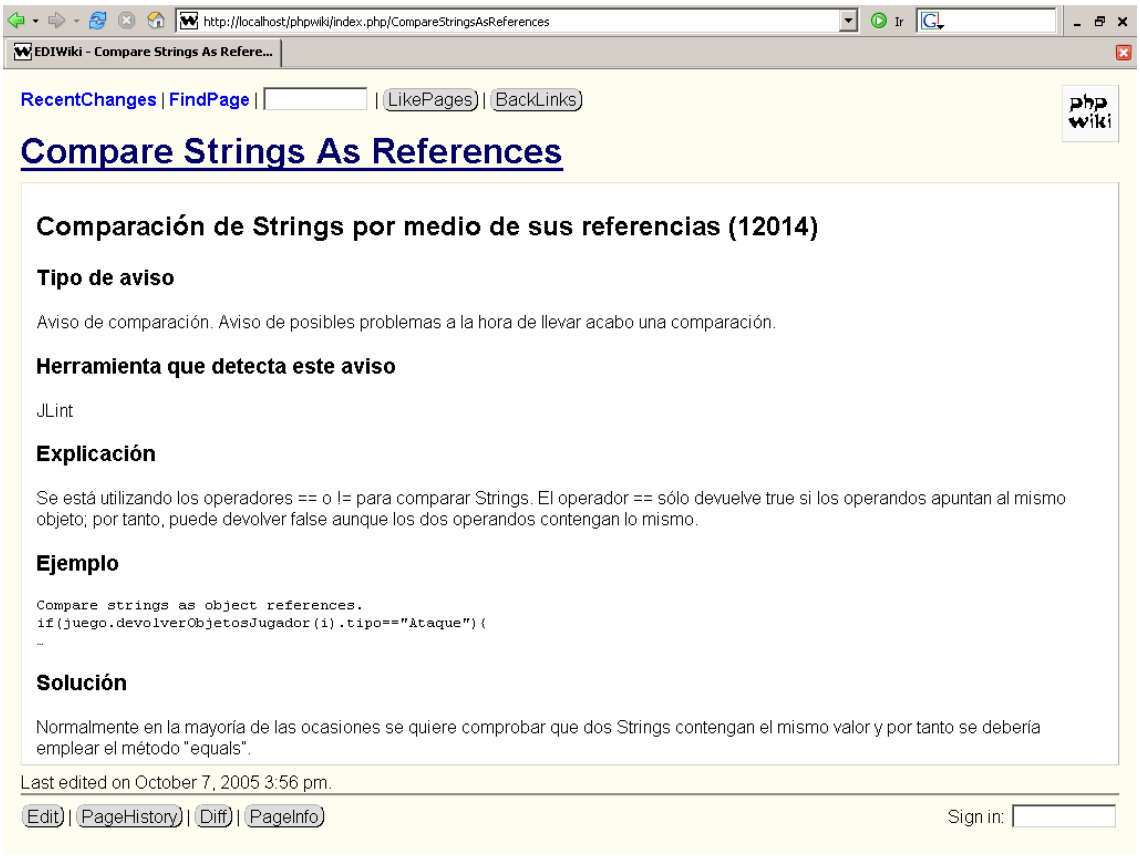


Figura 13. Vista de la página Web con toda la información sobre un aviso.

5.4.8 Almacenamiento de los mensajes en la historia de compilación

El Sistema de análisis de errores de programas (PBA) organiza la ejecución de las distintas herramientas de análisis estático mediante las unidades de compilación y recoge los mensajes generados por estas. Por un lado informa al desarrollador de los problemas encontrados a través de IDEWeb, como se ha descrito en el apartado anterior, y por otro envía estos mensajes a una base de datos para crear un histórico de errores relacionados con cada desarrollador perteneciente al proyecto, es lo que denominamos "historia de compilación".

5.4.9 Toma de decisiones en la corrección de un error

A veces, para solucionar un problema es necesario que los desarrolladores se comuniquen y tomen la decisión correcta sobre cual es la forma de afrontarlo. Tanto en problemas que requieren un rediseño del código como en errores las decisiones conjuntas entre varios programadores proporcionan varias ventajas: la toma de decisiones sobre una base más amplia suele dar como resultado mejores decisiones, permite el intercambio de experiencias

entre los distintos desarrolladores y por tanto el conocimiento de nuevos puntos de vista a la hora de afrontar determinados problemas que siempre resulta enriquecedor y permite separar quién escribió originalmente el código de quién realiza la corrección con lo que aumenta la flexibilidad del equipo. El inconveniente de este sistema es que se emplea mayor cantidad de tiempo para solucionar el error que si se hiciera individualmente; pero esto a largo plazo se compensa con un conocimiento más profundo de los errores y una eficacia mayor en su corrección.

5.4.10 Añadir contenidos a la base de conocimientos

Los desarrolladores utilizan la base de conocimientos para buscar información sobre los distintos errores que pueden aparecer; pero esta base de conocimientos, como las que tratan sobre un tema relativamente amplio, está siempre evolucionando ya que puede que sea necesario añadir información sobre una determinada causa de un error o el error detectado se ha producido en un contexto no previsto entonces el desarrollador tendrá que utilizar sus conocimientos para interpretar la nueva causa del error y buscar nuevas soluciones; pero una vez que consiga la solución, debería compartir estos nuevos conocimientos con el resto del grupo en la base de conocimientos. Para facilitar que se añadan conocimientos se ha intentado facilitar al máximo esta tarea y no se requieren herramientas especiales, el mismo navegador que utilizamos para consultar la base de conocimientos nos sirve para realizar modificaciones sobre ella: añadir nuevo contenido, corregir algún error o borrar datos desactualizados. De esta forma la base de conocimientos irá creciendo a medida que se vaya utilizando.

Por otra parte, el subsistema IDEWeb dispone de un módulo que permite crear nuevas páginas en la base de conocimientos cuando estas no existen. Por ejemplo, cuando se incorpora una nueva herramienta aparecen nuevos mensajes de error que hasta ese momento no existían, no hace falta que se vaya creando página a página las correspondientes a todos los errores sino que el sistema al ir a consultar un mensaje que todavía no tiene página la crea con la información básica sobre ese error para que no haya que empezar desde cero.

5.4.11 Análisis y elaboración de los avisos sobre errores

Código de Error	4	13019	34	10013	11019	13017	12027	11002
Frecuencia	3,333	2,222	1,111	0,667	0,444	0,444	0,222	0,222

Número de compilaciones : 9

Figura 14. Tabla de frecuencia de errores elaborada por PBA

El Sistema de análisis de errores de programas (PBA) trabaja sobre la historia de compilación de un usuario para elaborar avisos e informes que le puedan ser útiles al usuario para que se de cuenta de los errores que comete al programar, aprender sobre sus causas y así poder prevenir nuevos errores cuando escriba más código. Es decir, pretendemos trabajar sobre el estilo de programación y no solamente sobre los errores concretos de una compilación determinada. De forma análoga a como trabajamos con los estilos de aprendizaje en el trabajo del autor junto con la Dr. Paule Ruiz [Paule 2003], pretendemos aquí actuar sobre los estilos de programación.

El sistema permite muchas posibilidades de configuración para elaborar las estadísticas que permitirán seleccionar los avisos que se envíen al programador. Esta configuración se

especifica mediante archivos XML con una estructura claramente definida con un esquema XML que permite su validación automática. Se puede configurar el intervalo temporal del histórico sobre el que se aplica la estadística y el tipo de errores tratados siguiendo los criterios de herramienta, código interno, tipología, etc. De esta forma podemos obtener, por ejemplo, los errores más frecuentes de un usuario en las compilaciones de la última semana o los errores más frecuentes en el último mes. A partir de las estadísticas se seleccionarán los avisos que se mostrarán a cada usuario a través de IDEWeb.

5.4.12 Seguimientos de la historia de trabajo del proyecto

Tanto en la vida académica como profesional es necesario revisar no sólo el resultado final de un proyecto sino estados intermedios para encontrar las causas a determinados errores o conocer la forma en que fueron solucionados, esto es lo que denominamos historia de trabajo.

El Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV) permite no sólo examinar los proyectos en su estado final sino volver atrás y comprobar que miembro del equipo hizo determinada modificación y de esta forma explorar la evolución del proyecto.

5.4.13 Análisis global de los errores y su evolución

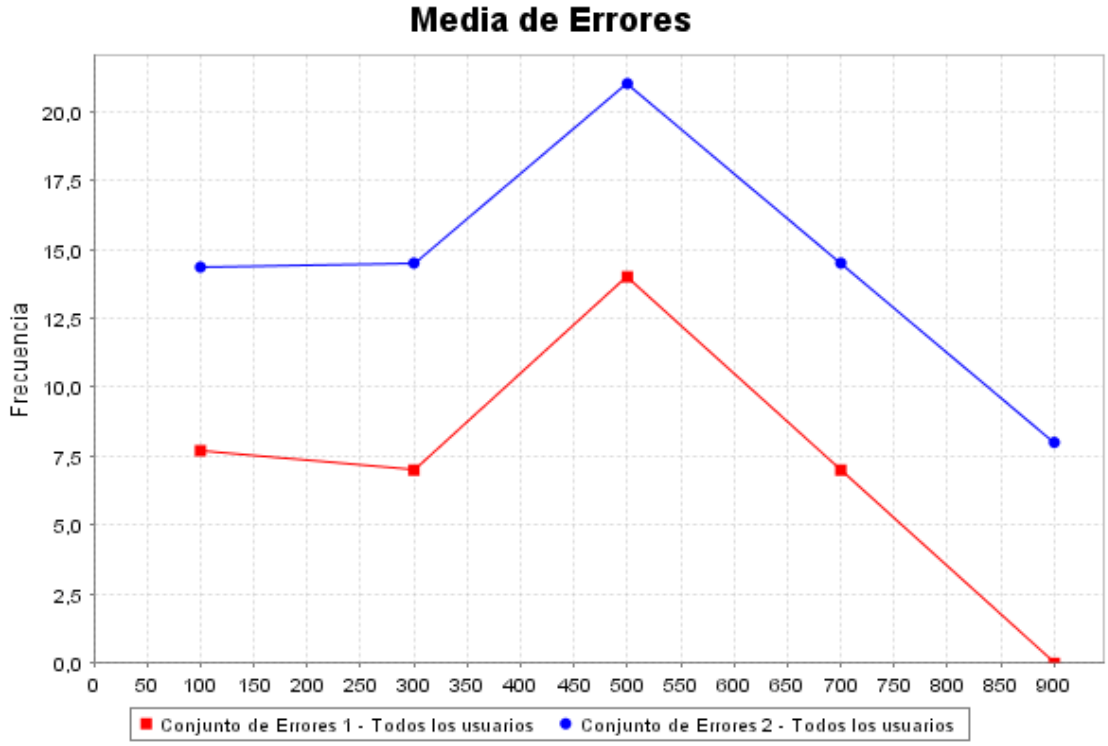


Figura 15. Gráfica de la evolución media de errores para dos conjuntos de errores distintos

De cara al seguimiento global del proyecto nos interesan unas estadísticas diferentes a las proporcionadas a los usuarios individuales, de esta forma es interesante disponer de estadísticas de evolución de errores para un determinado proyecto. El Sistema de análisis de errores de programas (PBA) permite configurar estadísticas que muestren esta evolución. El objetivo es que el estilo de programación evolucione para evitar errores antes de cometerlos y conseguir un código de calidad, y esto nos permite ver gráficamente la evolución de un desarrollador o de un grupo.

Capítulo 6. Compilador Web SICODE

En este capítulo, se describe un prototipo que aborda la implementación de los requisitos esenciales del sistema. Estos requisitos han sido planteados en el Capítulo 4. En esta primera fase de implementación se da prioridad a los requisitos que permiten la escritura de código fuente y compilar utilizando la Web como interfaz. Otro requisito abordado es la mejora del código centrada en el análisis de los errores de compilación, generando mensajes a partir de este análisis y posibilitando la creación de una ayuda semántica asociada a cada error.

Este prototipo se llevó a cabo en el año 2002 [Pérez 2003a][Pérez 2003b] y como el resto de los prototipos soporta Java [Arnold 1996] como lenguaje de desarrollo.

6.1 Objetivos

Los objetos que se plantearon a la hora de crear este prototipo fueron los siguientes:

- Crear un entorno de edición, compilación y corrección de errores, que permita a los usuarios escribir y llevar a cabo la compilación de sus archivos escritos en lenguaje Java, a través de un navegador Web, sin necesidad de tener que instalar en el equipo local ningún tipo de software.
- Permitir la gestión de los archivos que forman el proyecto que se está desarrollando, de forma que el entorno no limite el desarrollo a pequeños proyectos, sino que se puedan llevar a cabo proyectos con un número de archivos grande y para facilitar su gestión se puedan estructurar en paquetes.
- Generar para cada usuario un conjunto de avisos personalizados de manera automática, orientados a evitar errores de programación. Para realizar esto, se analizarán los errores previos que ha cometiendo propio usuario y el resto de los usuarios. Para conseguir una mayor efectividad, se analizarán en tiempo real los datos de las compilaciones y se generarán los avisos basados en este análisis.
- El entorno debe tener la posibilidad de asociar ayuda a cada tipo de error, y que sea fácilmente accesible por parte del usuario cuando esté corrigiendo los errores. Además, la ayuda debe ser fácilmente ampliable y modificable para adaptarla a los usuarios que están utilizando el entorno, en función de los errores frecuentes de cometen y del código que están escribiendo. Esta ayuda permitirá complementar la información de referencia que incluyen típicamente las plataformas de desarrollo, y debe ayudar al usuario a analizar cuál es el problema real que esta causando el mensaje de error, y cómo dar solución a este problema.

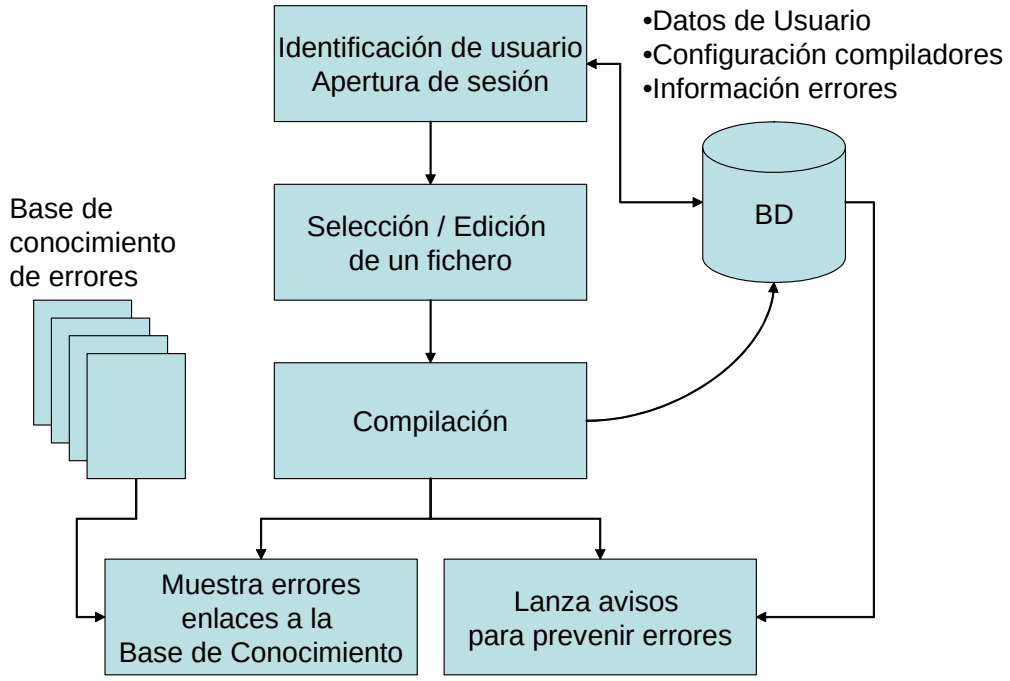


Figura 16. Modelo general del prototipo de Compilador Web SICODE

- Permitir a determinados usuarios (administrador) realizar un seguimiento de los errores de compilación cometidos por un usuario o conjunto de usuarios.

6.2 Requisitos que cumple el prototipo

A partir de los objetivos definidos en el apartado anterior definimos una serie de requisitos tanto funcionales como no funcionales, que especificamos a continuación.

6.2.1 Requisitos funcionales

Los usuarios de la aplicación se deben identificar y autenticar para poder hacer uso del sistema.

Almacenamiento de archivos en la carpeta personal del usuario. El usuario dispondrá de una carpeta personal donde se almacenarán todos los archivos que forman el proyecto o proyectos con los que trabaja el usuario.

Gestión de archivos. La aplicación permitirá realizar una gestión de archivos básica sobre los archivos del usuario: creación de nuevos archivos, selección de archivos existentes. También permitirá intercambiar archivos entre la carpeta personal y otros repositorios externos. Por último, permitirá la lectura y edición de un archivo seleccionado con el editor integrado en el sistema.

Compilación. El sistema permite configurar el compilador que debe utilizar el sistema que será el mismo para todos los usuarios. En la opción de compilación, el sistema utilizará la configuración actual para compilar el código fuente contenido en el archivo seleccionado en el gestor de archivos o que está actualmente en edición. El código objeto generado será almacenado en la carpeta personal. Si el compilador genera errores en la compilación, estos

se capturarán y se mostrarán al usuario para que pueda corregirlos y, por otro lado, se almacenarán para poder procesarlos posteriormente, creando la *historia de compilación*.

Visualización del archivo donde se localiza un error. El sistema debe de ser capaz de localizar el archivo y la línea relacionada con un determinado mensaje de error y mostrar el archivo correspondiente en el editor integrado.

Visualización de la ayuda asociada a un error. El sistema debe de ser capaz de buscar la ayuda hipertextual asociada a un mensaje de error determinado en la base de conocimientos.

Gestión de avisos o mensajes a los usuarios. El sistema realizará un análisis a partir de los mensajes de error almacenados en la historia de compilación y buscará los errores más frecuentes y los últimos cometidos, tanto por el usuario individual como por el conjunto de usuarios de la aplicación. El sistema mostrará información personalizada al usuario sobre sus errores y también sobre los del conjunto de usuarios.

6.2.2 Requisitos no funcionales

El sistema debe evitar que el usuario final tenga que realizar cualquier tipo de instalación y configuración del software.

El sistema debe poder ser utilizado por un grupo de usuarios simultáneamente, cada uno con su conjunto de archivos de la carpeta personal y utilizando conjuntamente los recursos comunes, como la ayuda sobre los errores y los mensajes de grupo, sin que se produzcan interferencias entre ellos.

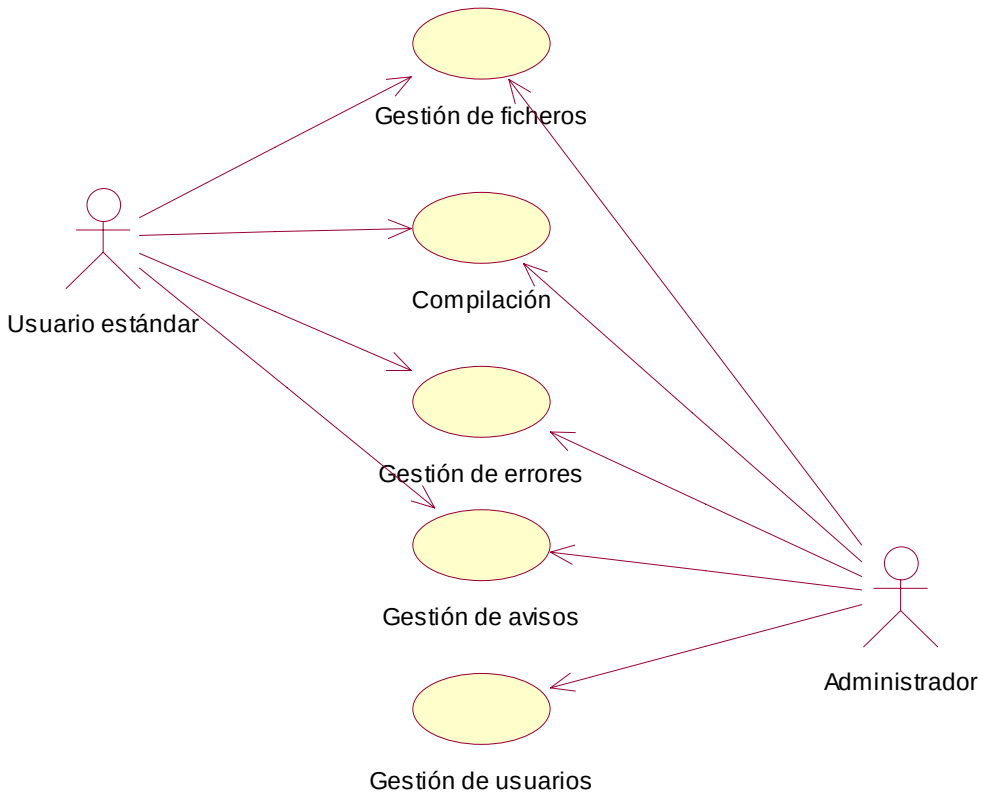


Figura 17. Diagrama de casos de uso del prototipo de Compilador Web SICODE

Se debe almacenar la ayuda de forma modular y en un formato que permita fácilmente su modificación manual para adaptarla a los usuarios que están utilizando la herramienta. Se pretende aquí incluir técnicas de adaptación conceptual [Fernández 2003].

6.3 Casos de uso

En este apartado describimos cada uno de los casos de uso y los actores del sistema, esto lo representamos en la Figura 17.

Administrador. Se encarga de administrar los usuarios del sistema, puede acceder a los ficheros de la base de conocimientos y además, es también usuario estándar de la aplicación. Es el encargado de hacer las altas, bajas, modificaciones y consulta de usuarios. Como usuario de la aplicación podrá abrir, cargar, descargar y compilar archivos, solicitar ayuda y visualizar el archivo que contiene un determinado error.

Usuario estándar. Podrá abrir, cargar, descargar y compilar archivos solicitar ayuda y visualizar el archivo que contiene un determinado error. No tendrá acceso a la información (cuentas y archivos) del resto de usuarios del sistema.

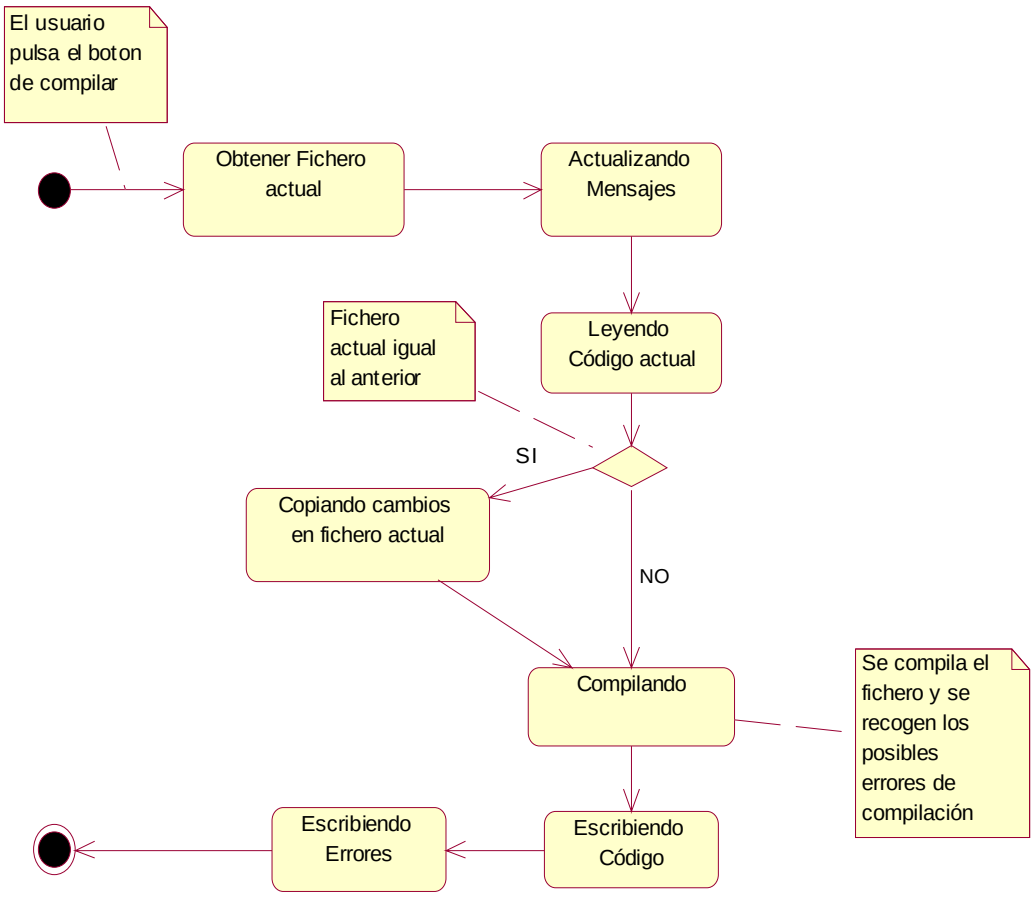


Figura 18. Diagrama de actividades del proceso de compilación de un archivo fuente

Entrada de un usuario a la aplicación. Tanto los Administradores como los usuarios estándar están identificados por su nombre de usuario y contraseña. El usuario deberá introducir su nombre de usuario y contraseña.

Gestión de usuarios. Se crean, borran, modifican o consultan las cuentas de los usuarios. Sólo los Administradores pueden realizar este tipo de operaciones.

Gestión de archivos. Un usuario puede abrir, enviar un archivo al directorio personal y extraerlo de ella. La carpeta personal es siempre el centro de estas operaciones. Por cada usuario se creará una carpeta personal, cada carpeta estará identificada por el nombre de usuario y contraseña.

Compilación. Permite tanto a un usuario estándar como a un administrador realizar una compilación de un archivo o conjunto de archivos del proyecto obteniendo un archivo objeto si la compilación fue correcta o un conjunto de errores si hubo problemas.

Gestión de errores. El sistema permitirá visualizar el error concreto de compilación y su localización dentro de un archivo de los que componen el proyecto; así mismo permitirá visualizar una página de ayuda para comprender las causas del error y abordar su solución. Por otro lado, el sistema almacenará en una base de datos la información sobre los errores encontrados.

Gestión de avisos. Envío de avisos a los usuarios elaborados por el sistema a partir del análisis de los errores previos del usuario almacenados por la gestión de errores. Serán de dos tipos: generales y personales. Contendrán información sobre los errores de compilación.

6.4 Actividad de compilación de un archivo

Describimos detalladamente el escenario de una compilación mediante un diagrama de actividad (Figura 18) para aclarar el proceso que debe realizar la aplicación.

El usuario selecciona el archivo que desee de su carpeta personal y elige la opción de compilar. Inicialmente se obtiene el archivo que el usuario desea compilar y se actualizan los mensajes de ayuda que se muestran al usuario con los datos de las anteriores compilaciones. A continuación se obtiene el código actual que se le está mostrando al usuario y una vez hecho esto se comprueba si el archivo actual, que es el que el usuario desea compilar, es el mismo que el usuario compiló la vez anterior (si es la primera compilación será nulo), en caso afirmativo se sobrescribe el archivo actual con el código actual que contiene las últimas modificaciones hechas por el usuario con lo que el archivo actual estará actualizado. En caso negativo no es necesario hacer ningún paso previo antes de compilar. Seguidamente se compila el archivo actual y se recogen los posibles errores de compilación y finalmente se muestra al usuario el código del archivo compilado y los errores de compilación que se han producido.

6.5 Diseño

6.5.1 Diseño de la arquitectura de la aplicación

Teniendo en cuenta los requisitos establecidos para el prototipo, se ha diseñado una arquitectura de Aplicación Web centrada en el servidor (Figura 19). Donde toda la lógica de compilación, gestión y almacenamiento de archivos del proyecto y gestión de errores y avisos se realiza en el servidor. Todo el espacio de almacenamiento de archivos residirá en el servidor haciendo que estos archivos estén disponibles independientemente del ordenador desde el que se conecte el usuario.

Proporcionaremos una interfaz Web que permite que el cliente esté compuesto, simplemente, por un navegador Web estándar que permitirá acceder a todas las opciones del entorno.

Con esta arquitectura se cumple el requisito de evitar la instalación de software en el ordenador del usuario final. Además, permite una configuración centralizada del sistema que permite que el administrador la realice para todo el sistema, evitando a los usuarios estándar preocuparse de estos temas. Proporciona a los usuarios independencia del lugar desde donde se conectan debido a que todos los recursos y archivos residen en el servidor.

Una tecnología que se adapta perfectamente a estas necesidades es la combinación de servlets / JSP basados en la plataforma Java de Sun Microsystems [Coward 2001] [Pelegrí-Llopart 2001] y es la que hemos empleado para la implementación.

6.5.2 Diseño de la navegación

La Figura 20 constituye un diagrama donde representa el mapa general de navegación y se muestran las posibilidades de navegación de los distintos usuarios a través del entorno. En el esquema se pueden ver las distintas partes de la aplicación, así como las relaciones entre las mismas. En dicho esquema, los rectángulos redondeados representan procesos que se realizan mediante servlets, los rectángulos representan páginas HTML.

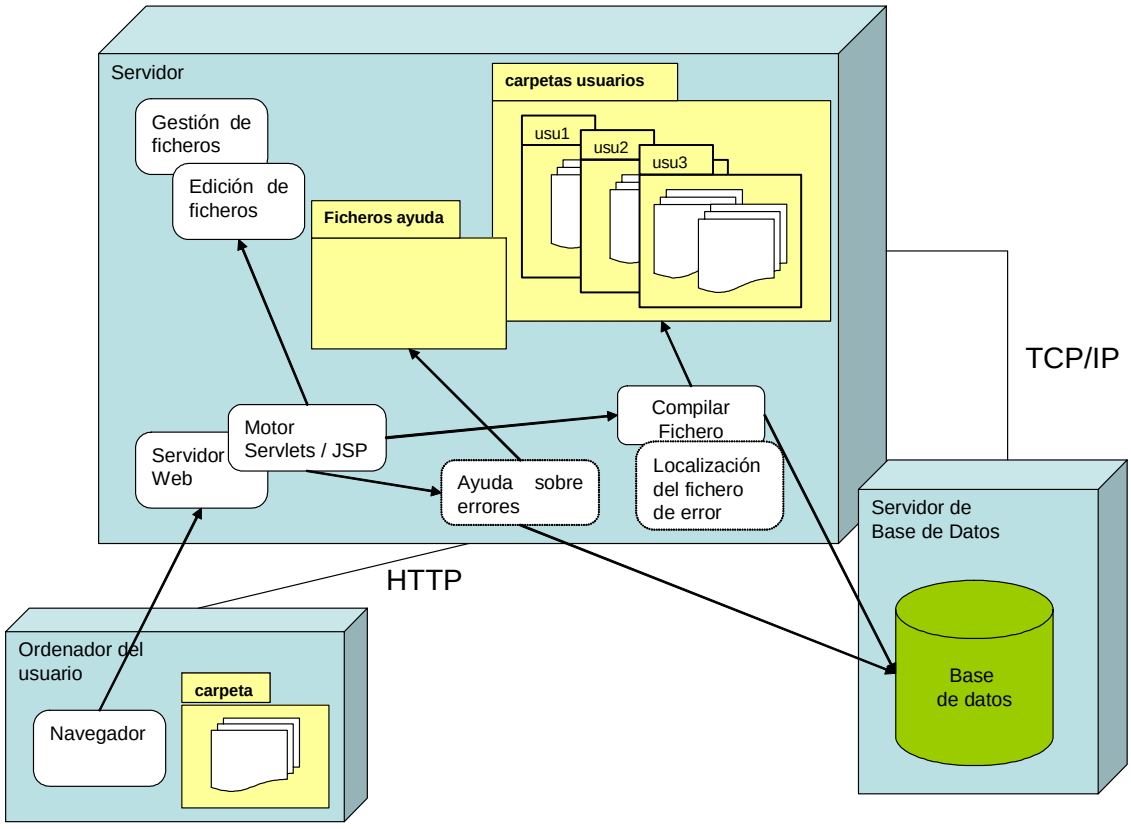


Figura 19. Arquitectura general del sistema

En el módulo de acceso tenemos una página HTML en la que el usuario introduce su nombre de usuario y su contraseña para poder utilizar la aplicación, dichos datos serán validados con los almacenados en la base de datos. En caso de no ser correctos se informará al usuario mediante la página No registrado ("NoRegistrado.html") de que no dispone de acceso, si son correctos, el usuario tendrá acceso a la aplicación.

Una vez iniciada la sesión y dependiendo de si se trata de un usuario estándar o de un administrador, se le mostrará la página principal de compilación, creándose dinámicamente (de ahí que tenga un contorno discontinuo) mediante un servlet; o la página opciones de administrador que es una página HTML normal.

Vamos a suponer que el acceso lo ha realizado un usuario estándar. La página principal reúne la funcionalidad de un entorno integrado sencillo, en la que el usuario podrá realizar la gestión de archivos, la edición y compilación de código; además de ser donde se muestran al usuario los avisos de prevención de errores. En dicha página, podrá abrir un archivo, crear un nuevo archivo, importar un archivo desde el equipo del usuario a su carpeta personal y exportar un archivo desde su carpeta personal a su equipo, en todo momento el usuario podrá cancelar la operación seleccionada, en cuyo caso volverá a mostrársele la página principal para que pueda seleccionar otra opción.

En el módulo de compilación, tenemos una primera fase en la que se seleccionan los mensajes de ayuda que se muestran al usuario en función del análisis de la historia de compilación (registro de los mensajes generados en todas las compilaciones precedentes); y una segunda fase en la que se lleva a la compilación propiamente dicha, recogiendo los posibles errores de compilación y actualizando los datos de la base de datos.

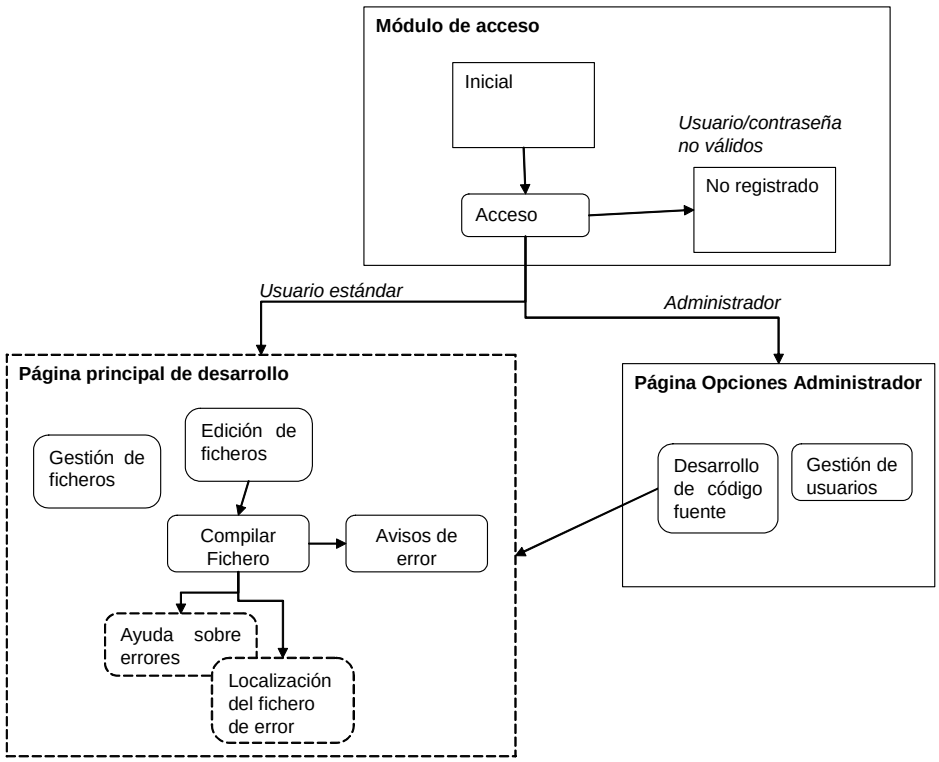


Figura 20. Diagrama de navegación de la interfaz del sistema

Además de lo anterior, en la página principal de compilación también intervienen dos módulos que sólo estarán activos si en la compilación se ha producido algún error de compilación (de ahí que tengan un contorno discontinuo) y que le permitirán al usuario: acceder a la ayuda sobre un determinado error, en cuyo caso extrae de una carpeta incluida en la aplicación el archivo HTML de ayuda; o abrir el archivo que contiene un determinado error de compilación si este se ha producido en un archivo distinto al actualmente activo.

Si el acceso lo realiza un administrador se le mostrará una página HTML donde le aparecerán las opciones de administrador relacionadas con la gestión de usuarios: dar de alta, dar de baja, realizar modificación y realizar consulta. Como ocurría en el caso de la gestión de archivos, una vez seleccionada una opción el administrador podrá cancelar la ejecución de la misma volviendo a la página de opciones del administrador. Además, contará en dicha pantalla con una opción que le permite utilizar la aplicación como si fuera un usuario estándar para poder realizar la compilación de sus archivos. Las opciones de administrador tendrán repercusión tanto en la base de datos como en las carpetas personales de los usuarios ya que ambas deben de reflejar los cambios realizados por el administrador.

6.5.3 Diseño de la interfaz de la página principal de desarrollo

En esta sección describiremos detalladamente la interfaz correspondiente a la página principal de la aplicación (Figura 21) que el usuario utilizará en el proceso de desarrollo. Se trata, de reunir en una misma página toda la funcionalidad e información necesaria para que el usuario desarrolle una aplicación; sin necesidad de navegar por la aplicación buscando diferentes opciones.

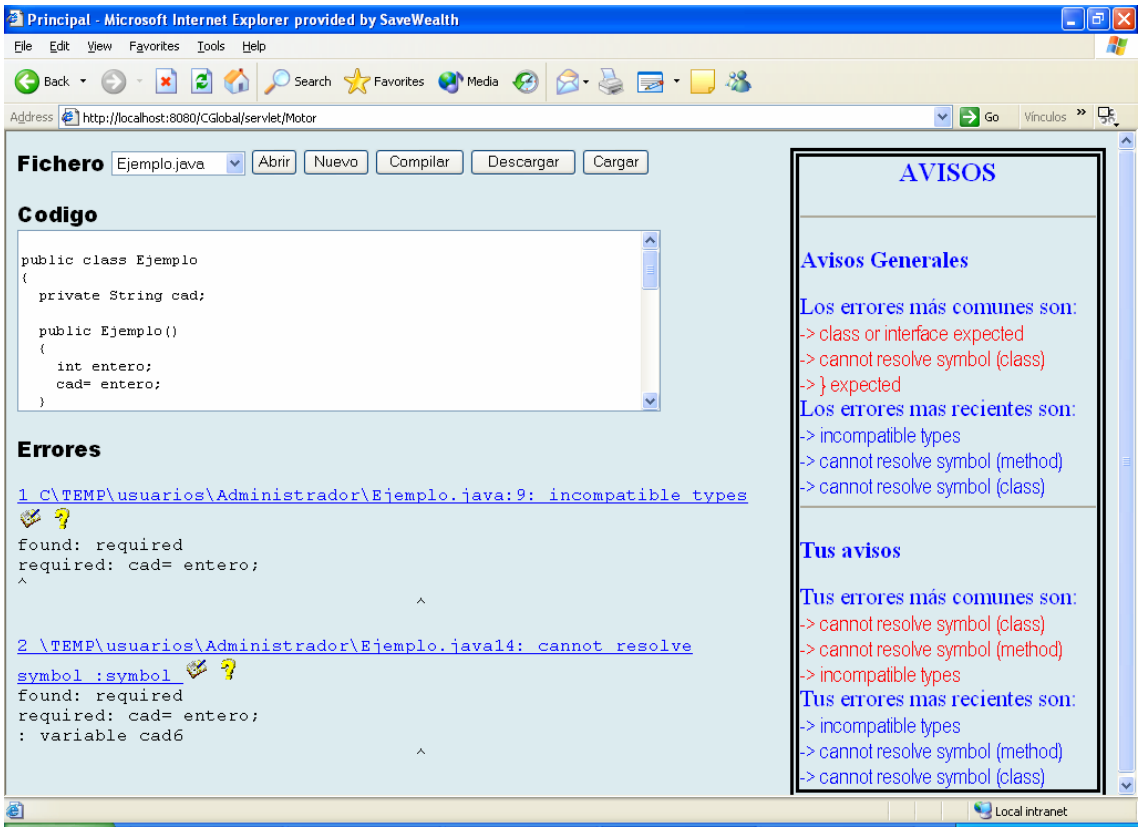


Figura 21. Pantalla del entorno de desarrollo en Web

En la pantalla principal destaca un **área de edición** etiquetada como "Código" donde el usuario puede escribir el código fuente de los ficheros y modificar los ficheros existentes.

En la pantalla principal de la aplicación el usuario podrá realizar las operaciones que se describirán a continuación y todas son accesibles a través del **menú de desarrollo** (Figura 22).

Abrir: Al pulsar sobre esta opción se abrirá el archivo actualmente activo en el campo "Fichero" y se mostrará al usuario el código de dicho archivo.

Nuevo: Al pulsar en esta opción se mostrará una pantalla en la que el usuario podrá introducir un nombre para un nuevo archivo que se creará en su carpeta personal.

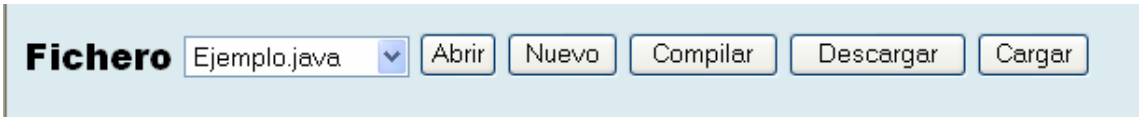


Figura 22. Detalle del menú de desarrollo del sistema

Compilar: Al pulsar sobre este botón se compilará el archivo previamente abierto con el botón Abrir. Si el archivo que se compila contiene errores de compilación, al pulsar Compilar aparecerá cada uno de dichos errores el en **área de notificación de errores** (Figura 23) e irán acompañados de dos iconos: al pulsar en el primero, abrirá el archivo que contiene ese error; con el segundo icono obtendrá una ayuda sobre ese tipo de error.

- Abrir el archivo que contiene el error: Al pinchar sobre el enlace que indica la ubicación del error o sobre el icono del libro (ver Figura 23) se abrirá el archivo que contiene el error para que el usuario pueda modificarlo.
- Solicitar ayuda: Al pinchar sobre el icono de la interrogación (ver Figura 23) se le mostrará al usuario una página con la ayuda correspondiente a ese tipo de error o bien se le indicará que no hay ayuda disponible para ese tipo de error.

Cargar: Al pulsar sobre este botón se mostrará una pantalla que permitirá al usuario seleccionar un archivo para que se copie de su equipo a su carpeta personal.

Descargar: Este botón se le mostrará al usuario una pantalla en la que aparecerá todos los archivos que haya actualmente en su carpeta personal y en la que podrá seleccionar aquel que desea copiar desde su carpeta personal a su equipo.

Figura 23. Detalle del área de notificación de errores de la página principal del sistema

Además, de las zonas ya descritas la parte derecha de la pantalla está reservada para el **área de avisos** (Figura 24). Aquí pretendemos ofrecer al usuario avisos cortos que le sirvan de

recordatorio para no volver a repetir los errores a los cuales es más propenso. Esta área está subdividida en dos partes con dos categorías cada una:

- Avisos generales. Avisos referidos a todo el grupo de usuarios de la aplicación; pero que pueden tener validez para este usuario individual. Se muestran los tres errores que con más frecuencia cometen los usuarios de la aplicación en su conjunto y los últimos errores cometidos, que al haber varios usuarios trabajando en paralelo, no tienen que coincidir con los del usuario actual.
- Tus avisos. Avisos referidos al desarrollador individual. Se muestran los tres errores más frecuentes en la historia de compilación y los tres más recientes para el usuario que está en sesión.

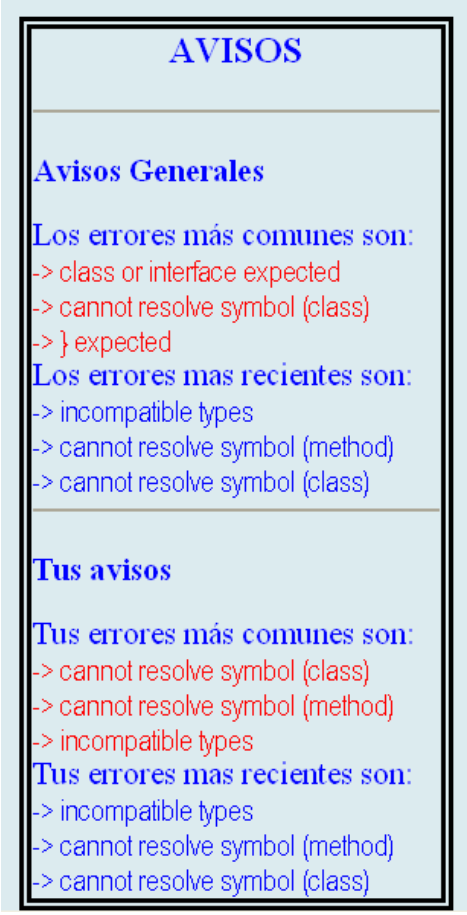


Figura 24. Área de avisos

6.5.4 Modelado de datos

Mostramos el diagrama entidad – relación (Figura 25) para mostrar la forma en la que se guardan la información sobre los errores, como se asocian a los usuarios y los datos auxiliares para realizar las estadísticas. En el diagrama se ven las entidades de información que guarda el entorno, se podrían clasificar estás entidades en tres grupos: información relacionada con los usuarios, información relacionada con los errores e información relacionada con el análisis de errores (estadísticas).

A continuación explicamos cada una de las entidades y relaciones que forman el diagrama:

Entidad: **usuarios** (Contiene el login, password y grupo de todos los usuarios de la aplicación). Se utiliza para validar los datos de acceso de los usuarios a la aplicación y en ella se reflejan las altas, bajas y modificaciones realizadas por los administradores.

Entidad: **claves_errores** (Contiene las palabras clave que identifican a los errores, el código que se corresponde con cada error y el mensaje que se corresponde con cada código). Se utiliza para analizar la cadena del mensaje de error que proporciona el compilador y clasificarlo según un tipo de error para poder contabilizarlo y almacenarlo así como para obtener los mensajes de ayuda para mostrarselos al usuario.

Entidad: **estadísticas_errores** (Contiene información acerca de los errores cometidos por un determinado usuario). Se utiliza para saber los errores más frecuentes y más recientes cometidos por cada usuario y nos indicará los códigos de los errores cuyos avisos debemos mostrar al usuario. Dichos avisos serán de tipo personal.

Entidad: **estadística_total** (Contiene información acerca de los errores cometidos por todos los usuarios de la aplicación). Se utiliza para saber los errores más frecuentes y más recientes cometidos por todos los usuarios y nos indicará los códigos de los errores cuyos mensajes debemos mostrar al usuario. Dichos mensajes serán de tipo general.

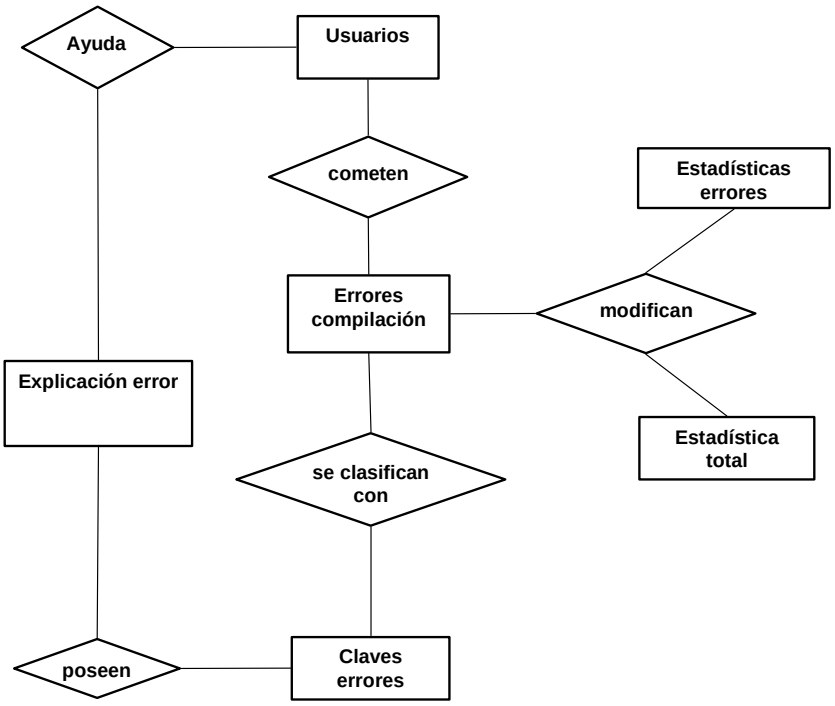


Figura 25. Diagrama entidad - relación de la base de datos del sistema

Entidad: **explicación_error** Sirve para obtener la dirección URL en la que se encuentra el archivo de ayuda correspondiente a un determinado error para poder mostrar dicha ayuda al usuario.

Entidad: **errores_compilación** (Almacena para cada usuario, la información de los mensajes de error generados por el javac al compilar). Se utiliza para almacenar los datos obtenidos de la salida del compilador javac y a partir de los cuales se actualizarán los datos de las entidades estadistica_errores y estadistica_total.

Relación: cometen (Asocia cada usuario con los errores de compilación que comete).

Relación: modifican (Asocia cada error cometido por el usuario con el número de veces y frecuencia con que se comete dicho error).

Relación: se_clasifican_con (Asocia cada error de compilación con el código y mensaje de error correspondiente).

Relación: poseen (Asocia cada código de error con el archivo HTML donde está la ayuda correspondiente a ese error).

Relación: ayuda (Asocia cada usuario con el archivo de ayuda que se le debe mostrar).

6.6 Descripción del prototipo

El prototipo trata de desarrollar una herramienta que facilite al usuario el aprendizaje activo de distintas técnicas de programación, utilizando el lenguaje Java [Arnold 1996], para ello le permite la visualización, modificación y compilación de sus propias aplicaciones realizadas en dicho lenguaje, y realiza una gestión de errores avanzada para ayudar al programador a no repetir errores.

6.6.1 Roles y apertura de una sesión en la aplicación

Se distinguen dos tipos de usuarios de la aplicación. Por un lado, los usuarios estándar que puede ser cualquier programador que quiera realizar un proyecto con la herramienta, y por otro lado están los administradores que dispondrán de toda la funcionalidad de los usuarios estándar y alguna extra como la gestión de cuentas de usuario, la modificación y ampliación de la ayuda y el seguimiento de los errores de un usuario. Al tratarse de una herramienta enfocada hacia el ámbito de la enseñanza de un lenguaje de programación los usuarios estándar pueden identificarse con los alumnos y los administradores con los profesores.

La forma de acceder a la aplicación será similar para los usuarios estándar y los administradores y se realiza mediante la introducción de un nombre de usuario y una contraseña que los identifica de forma única. La información que se les muestra y las tareas que pueden realizar unos y otros varían dependiendo del tipo de usuario que se trate. La comunicación entre los usuarios estándar y los administradores para lo relacionado con la solicitud de altas, bajas y modificaciones queda fuera del ámbito de este prototipo. La evaluación de los proyectos se lleva a cabo con herramientas externas, por ejemplo vía e-mail.

6.6.2 Gestión de archivos y espacio de almacenamiento

Por cada usuario, tanto si se trata de un administrador como si se trata de un usuario estándar, se crea una carpeta personal en el servidor, identificada por el nombre de usuario de los mismos, en la cual se almacenan sus archivos fuente Java con los que trabajan y los ".class" resultado de las compilaciones. Los usuarios pueden acceder a los archivos de su carpeta personal pero no a los de los demás usuarios. Esto ocurre, para este prototipo, incluso en el caso del Administrador, que sólo podrá acceder a sus archivos.

El usuario puede importar (cargar) archivos que se encuentren en su equipo local con lo que estos se copian en su carpeta personal para poder trabajar con ellos y de forma similar puede exportar los archivos de carpeta personal a su equipo local. Así mismo, se pueden crear nuevos archivos y manipular los ya existentes en la carpeta personal.

6.6.3 Gestión de cuentas de usuario

Los administradores llevan a cabo todo lo relacionado con la gestión de todos los usuarios, tanto los usuarios estándar como de los propios administradores, es decir, pueden dar de alta a un nuevo usuario, eliminar un usuario existente, modificar los datos de un usuario o consultar los datos de los usuarios actualmente registrados. Al dar de alta a un usuario se crea en el servidor una nueva carpeta personal para ese usuario. Al eliminar una cuenta de usuario se borra del servidor su carpeta personal y todo su contenido. Al modificar los datos de un usuario, si se modifica su nombre de usuario, su carpeta personal pasará a identificarse por el nuevo nombre de usuario.

Otra característica de los administradores es que también tienen la posibilidad de utilizar la aplicación como si fueran usuarios estándar, es decir, crear sus propios archivos para llevar a cabo la compilación, sin necesidad de tener otro nombre de usuario y contraseña de usuario estándar.

6.6.4 Compilación, gestión de avisos y ayuda a la corrección

En la parte de la aplicación dedicada a la compilación propiamente dicha, el usuario tiene la posibilidad de abrir un archivo de su carpeta personal y modificarlo o crear uno nuevo desde cero para luego compilarlo, es decir unas tareas básicas de cualquier entorno de compilación estándar. Podemos compilar un archivo de dos formas: si el archivo se abre previamente se compilará el código del archivo que es visible en ese momento en pantalla; si no hay un archivo abierto, se compilará el archivo que está actualmente seleccionado en el gestor de archivos.

Un usuario solicita la compilación de un archivo y el entorno lo compila mostrándole los errores de compilación que se produjeron, si hubo alguno. Además, estos errores se almacenan en una base de datos, clasificados por tipo de error y relacionándolos con el usuario que realizó la compilación del archivo, para posteriormente analizarlos y generar los avisos que se explican a continuación.

Otro punto importante, es lo relacionado con los avisos que se les muestran a los usuarios. Aparte de visualizar los mensajes de error que se producen al compilar un archivo tal cual los genera el compilador, se le muestran al usuario una serie de avisos de ayuda referidos a los errores de compilación y que serán de dos tipos:

- Avisos Generales: Son avisos acerca de los errores más comunes y los más recientes cometidos por todo el conjunto de usuarios de la aplicación.
- Avisos Personales: En este caso se trata de los errores más comunes y los más recientes cometidos por el usuario. Este tipo de avisos están totalmente personalizados para cada usuario.

Estos avisos están visibles permanentemente mientras el usuario se encuentre en la parte de la aplicación destinada a la edición y se renuevan cada vez que el usuario entra en la aplicación y cada vez que se realice la compilación de un archivo, haciendo de este modo que en todo momento el usuario tenga una información acerca de los errores lo más actualizada posible. A la hora de decidir que avisos se deben mostrar al usuario en cada momento, se utilizaran una serie de datos referidos a los errores de compilación previamente cometidos. Dichos datos se almacenaran en una base de datos y nos permitirán seleccionar los avisos correspondientes a los errores más frecuentes y numerosos.

Otro punto a destacar en la aplicación, es lo concerniente a la ayuda para depurar los errores de compilación. Para un subconjunto de todos los errores de compilación que se

pueden producir, se mostrará una ayuda en formato de hipertexto, lo más completa posible donde se indicarán las posibles causas del error, las posibles soluciones, y más datos que facilitará a los usuarios la tarea de corrección de código. Un usuario solicita visualizar la ayuda respecto a un determinado error de compilación y si esta existe, se le muestra al usuario. Esta ayuda es fácilmente modificable y ampliable por el administrador, ya que básicamente se compone de un conjunto de páginas HTML.

Relacionado con esta tarea de depuración de código, el usuario tendrá la posibilidad de, cuando se visualizan los errores de compilación que se han producido, acceder fácilmente al archivo en el que se ha producido un determinado error si este se produce en un archivo diferente al que actualmente se está mostrando al usuario.

6.6.5 Clasificación de los mensajes de error por tipos para facilitar la gestión de errores

Una de las tareas centrales que debe realizar el sistema es la de gestionar datos sobre los errores cometidos. Para llevar a cabo esta tarea, se almacenarán una serie de datos respecto a los errores cometidos por los usuarios y se elaborarán unas estadísticas para informar a los mismos, el problema está en: ¿Qué datos de los errores se almacenan? Dependiendo del tipo de error se almacenan datos asociados diferentes. Por tanto, es importante clasificar los errores para poder agruparlos y para saber que información tenemos que guardar sobre cada uno. El compilador *javac* puede indicar un número de tipos de error de compilación considerable, además el mensaje de error incorpora palabras correspondientes a los identificadores de los elementos relacionados con el error: nombres de archivos, clases, métodos, variables, etc. La posibilidad de que dos errores de compilación sean exactamente iguales es muy reducida; por tanto, la clasificación de los errores no es directa. Veamos un ejemplo:

Supongamos que al compilar un archivo llamado HolaMundo.java genere el siguiente error:

```
HolaMundo.java:7:cannot resolve symbol
symbol  : class Usuario
location: class HolaMundo
Usuario user = new Usuario ();
^
```

Y ahora supongamos que se compila un archivo llamado Helloword.java y que le produce el siguiente error:

```
Helloword.java:8:cannot resolve symbol
symbol  : class Admin
location: class Helloword
Admin ad = new    Admin ();
^
```

En este ejemplo, se ve claramente que se trata del mismo tipo de error pero los datos de ambos son distintos salvo ciertas palabras, y estas son en las que nos apoyaremos para llevar a cabo la comparación. Estas palabras se repiten en todos los errores que son del mismo tipo. En este caso, por ejemplo, estas palabras son, en la primera línea: "**cannot resolve symbol**", y en la segunda línea: **class**. Estas palabras "clave" son las que nos permitirán determinar que nos encontramos ante el mismo tipo de error. Por cada tipo de error se buscarían dichas palabras y se almacenarían de manera que ante cada error de compilación que se produzca, se realice una comparación con todo el conjunto de palabras clave almacenadas y si se produce una coincidencia de palabras "clave" entre las del error y las guardadas sabremos que se trata de un determinado tipo de error y por lo tanto podremos cuantificarlo para su posterior uso en la generación de estadísticas y avisos que sirvan de ayuda al usuario.

En este momento surge otra pregunta importante: ¿cómo obtener las palabras clave de todos los tipos de errores cuyo número, como se dijo, es muy considerable? Lo que se ha

hecho en este prototipo del Compilador Web SICODE es acotar el número de tipos de errores a estudiar y reducirlo a un conjunto de los más comunes, sobre los que se ha hecho un estudio lo más exhaustivo posible, tratando de proporcionar la mejor ayuda posible al usuario y con el resto lo que se hace es simplemente mostrar el error al usuario pero sin guardar ninguna información específica del mismo.

Una vez que se ha clasificado los errores esta información se utiliza para varias tareas dentro del proceso del procesamiento de errores (Figura 26):

- La clasificación del error permite saber que información se debe almacenar en la base de datos.
- Además, se utiliza en el análisis de errores para hacer métricas por tipos de error. De esta forma se generan avisos relacionados con los tipos de errores.
- Por último, se utiliza para enlazar de forma automática cada uno de los errores mostrados al usuario con la información de la base de conocimientos correspondiente a ese tipo de error.

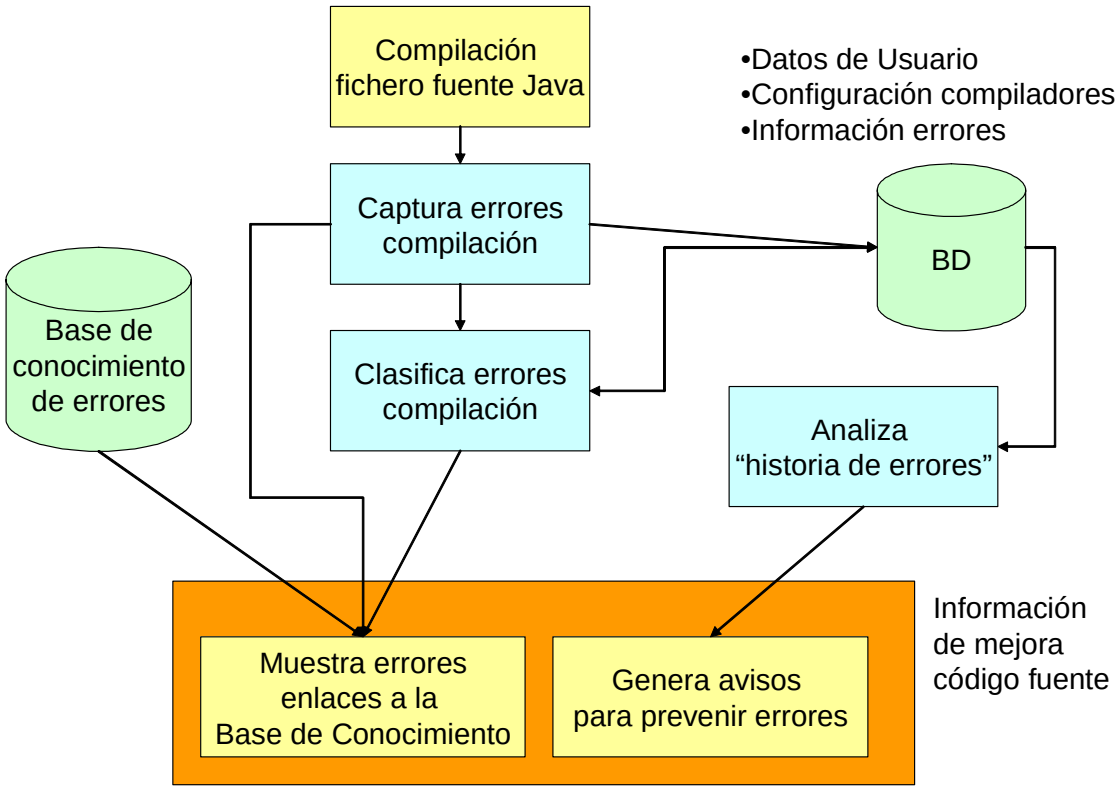


Figura 26. Modelo de la gestión de errores

6.7 Limitaciones del prototipo

El prototipo realizado tiene una serie de limitaciones en cuanto a los requisitos generales planteados en Capítulo 4 para un sistema de este tipo. Enumeraremos a continuación estas limitaciones que serán resueltas, en su mayor parte, por los siguientes prototipos desarrollados.

El análisis de código fuente se limita al que realiza un compilador, actualmente existen gran cantidad de herramientas que permiten un análisis de código fuente más exhaustivo y que proporcionará mayor información a los programadores; además, sería muy interesante poder combinar varias herramientas de este tipo para mejorar el análisis estático.

Sólo se captura información de errores en tiempo de compilación, no se proporciona ninguna forma de realizar las ejecuciones y capturar las excepciones en tiempo de ejecución generadas. Si un programador quiere realizar pruebas de ejecución de una clase (unitarias) o de un conjunto de ellas (de integración) para comprobar su buen funcionamiento, el único camino para realizarlo es descargando el archivo ".class" generado en las compilaciones; ya que el sistema no dispone de ninguna forma de realizar las ejecuciones en el servidor. Además, no tendríamos almacenadas las excepciones que puedan surgir en las pruebas.

Las métricas y estadísticas que se proporcionan sobre los errores únicamente se proporcionan como datos numéricos y solamente clasificadas por tipo de error, esto puede ser suficiente para generar avisos hacia un programador; pero para hacer una evaluación general o un diagnóstico sobre la calidad del código y su evolución estos datos son insuficientes, es necesaria una mayor flexibilidad y una presentación gráfica de los datos.

La lógica de clasificación de errores se encuentra inmersa en el código lo cuál dificulta la ampliación y extensión del sistema a otros compiladores y herramientas, es necesario disponer de un sistema de codificación y poder guardar en la base de datos las reglas para realizar este trabajo.

Desarrollo de aplicaciones en este entorno se realiza de forma individual. No hay herramientas para el apoyo al trabajo en grupo lo cual limita esta forma de trabajo.

La comunicación entre los usuarios estándar y los administradores para realizar operaciones de gestión queda fuera del ámbito de esta aplicación y se lleva a cabo externamente a ella, por ejemplo vía correo electrónico. Esto impide la simplificación de tareas específicas del desarrollo de aplicaciones, como puede ser la revisión de un archivo fuente, la toma de decisiones para solucionar un error en el código fuente, etc.

Modificación y ampliación de la ayuda se debe realizar manualmente directamente sobre los archivos HTML del servidor. Es necesario el planteamiento de un sistema colaborativo que facilite y motive a los usuarios a aportar información a la base de conocimientos común.

No es posible realizar un seguimiento del proceso de desarrollo del proyecto sino que sólo se puede examinar el resultado actual del trabajo. Para pequeños proyectos no hay problema en la revisión del estado final de este; pero para grandes proyectos es fundamental a la hora de analizar las causas de un error poder analizar los pasos en el proceso de desarrollo de un proyecto, por tanto necesitamos un sistema que soporte este requisito.

La revisión de los mensajes de error de los usuarios también debe realizarse manualmente sobre la base de datos que almacena todos los mensajes de error de cada usuario. Es necesario un sistema que permita un seguimiento cómodo del historial de errores.

6.8 Conclusiones sobre el prototipo de Compilador Web SICODE

Con este prototipo se han cumplido los dos objetivos fundamentales:

- Desarrollar un entorno que se pudiese ejecutar en cualquier ordenador sin necesidad de instalación y fácil de utilizar. Para lo cual se ha utilizado una

arquitectura Web y una interfaz de un entorno integrado de desarrollo sobre un navegador.

- Desarrollar un modelo centrado en los errores para la mejora de la calidad del código. Captura de errores de compilación, clasificación y almacenamiento en una base de datos, para su posterior análisis que permite una emisión de avisos preventivos que visualizará el entorno y por otro lado la base de conocimientos basada en página HTML, asociada a los distintos tipos de error.
- El entorno permite a los desarrolladores gestionar proyectos reales gracias a la posibilidad de configuración de distintos compiladores y bibliotecas. Además, el tamaño de los proyectos puede crecer ya que se dispone de un gestor de archivos flexible y también integrado en el entorno.

En los siguientes capítulos propondremos prototipos más evolucionados para cada uno de los subsistemas de este Compilador Global SICODE que nos permitan acercarnos más a los requisitos ideales especificados en el Capítulo 4.

Capítulo 7. Sistema de análisis de errores de programas: PBA

Como ya se comentó en el Capítulo 5 la política de desarrollo del sistema ha consistido en implementar un prototipo del Compilador Web SICODE con la funcionalidad básica para luego dividirlo en subsistemas relativamente independientes para poder profundizar más en la funcionalidad de cada uno de ellos. En este capítulo presentamos el Sistema de análisis de errores de programas (Program Bug Analysis) PBA, que se centra en la captura y procesamiento de errores del proceso de desarrollo de software para facilitar la mejora en la calidad del código fuente de los programas.

7.1 Objetivos del prototipo

La meta de este prototipo es obtener un sistema capaz de capturar, analizar y clasificar los errores de los desarrolladores a la hora de programar. Para conocer los errores, emplearemos como información base la proporcionada por el compilador e intentaremos ampliar el grado de detección de errores de los compiladores utilizando herramientas de análisis adicionales. Dichas herramientas, mediante análisis del código fuente o del código objeto generado, permitirán detectar errores que los compiladores no detectan. Toda la información obtenida se guardará, clasificada por el tipo de error y estará asociada al usuario que haya escrito el código correspondiente, para ser consultada y obtener información estadística a partir de ella.

No es objetivo de este prototipo el crear nuevas herramientas o técnicas para el análisis de código sino que se utilizarán las herramientas existentes vistas en el apartado 3.8.4. Herramientas análisis estático de código del Capítulo 3. Utilizando los resultados que generan, combinándolos y calculando métricas para presentarlas de forma gráfica.

El sistema será fácilmente configurable para ser capaz de trabajar con diferentes lenguajes de programación, aunque en nuestro prototipo nos centraremos en el análisis de código y de aplicaciones en el lenguaje Java. Se dejará la posibilidad del uso de diferentes compiladores y diferentes versiones para facilitar la actualización de los mismos. El sistema también estará preparado para utilizar herramientas de análisis de código distintas de las que hay configuradas.

También se le permitirá al usuario del sistema ejecutar el código compilado, siendo capaz de interactuar con las aplicaciones, pudiendo de esta manera hacer pruebas para detectar errores. En caso de que durante la ejecución se produzca una excepción, se dará al usuario información sobre cual es el error que puede haberla provocado.

El sistema dispondrá de un módulo que proporcionará información estadística al usuario sobre sus errores. Cada desarrollador podrá consultar información referida a los datos sobre sus errores, y a los datos del total de los desarrolladores.

A la hora del uso de la herramienta, habrá que diferenciar entre dos tipos de usuarios: los supervisores y los desarrolladores. El sistema de gestión de errores no será idéntico para todos los tipos de usuarios. La diferencia esta en la confección de las estadísticas, donde los usuarios supervisores, tendrán las mismas capacidades que los desarrolladores y además podrán usar los datos individuales de todos los usuarios del sistema para confeccionar estadísticas.

7.2 Ámbito del prototipo y relación con el resto de prototipos

Este prototipo se encarga exclusivamente del tratamiento de errores. Está concebido como un módulo que funcionará en segundo plano, sin interfaz de usuario, con la excepción de generación de páginas Web de resultados personalizados para cada usuario; serán otros módulos los que realicen llamadas a este, en función de las acciones del usuario.

El subsistema utiliza la información de usuarios para poder realizar métricas en relación a esta información; aunque no incluye ninguna gestión de grupos o usuarios que delega en otros subsistemas. Al carecer de interfaz con el usuario es otro el módulo que proporciona el entorno de edición de código y colaboración entre usuarios para el desarrollo de software. Este módulo trabaja con los ficheros fuente ya creados realizando la compilación y el procesamiento estático y una vez creados los ficheros objeto permite su ejecución y captura de excepciones.

7.3 Requisitos que cumple el prototipo

A continuación, se enumerar los requisitos de este prototipo, dividiéndolos en tres módulos.

7.3.1 Módulo de análisis estático

El módulo de análisis realiza el análisis estático de los ficheros fuente, captura de mensajes de error y almacenamiento los resultados del análisis del código que se realiza mediante un compilador y otros procesadores de lenguaje que realizan este análisis estático del código.

El sistema debe ser configurable:

- El sistema debe ser capaz de trabajar con diferentes lenguajes de programación.
- El sistema debe permitir el uso de diferentes compiladores y diferentes herramientas de detección de errores en un mismo análisis.
- Combinar la ejecución de distintas herramientas.

Se debe de permitir al usuario la elección de la forma de realizar los análisis: configuración de las herramientas que se van a utilizar, control de las dependencias que pueden tener unas respecto a otras y configurar las opciones de ejecución de cada una de las herramientas. Todo esto de una forma sencilla y homogénea para todas las herramientas, sin que tenga que conocer la forma específica en que se organizan las opciones necesarias en cada herramienta.

Se permitirá cambiar y añadir herramientas de una forma sencilla, con lo que lograremos que el sistema se adapte fácilmente a los cambios. Se permitirá la ampliación de las herramientas que son usadas en el sistema, pudiendo instalar nuevas herramientas de una forma sencilla.

Todos los errores o avisos detectados en un análisis serán guardados en una base de datos para su posterior uso creando un registro que denominaremos "historia de compilación".

Se utilizarán patrones búsqueda para clasificar los mensajes de error según el tipo al que pertenecen.

7.3.2 Módulo de análisis de la ejecución

El sistema debe permitir al usuario ejecutar las clases Java compiladas. Lo cual permitirá a los usuarios hacer pruebas unitarias para comprobar que su clase se ejecuta correctamente sin tener que exportar las clases fuera del sistema.

En caso de que se produzca una excepción durante la ejecución de la clase se capturará y se almacenará en la base de datos para poder realizar un seguimiento posterior de estas excepciones.

7.3.3 Módulo de elaboración de resultados

El módulo de elaboración de resultados realiza estadísticas sobre los errores almacenados por el sistema en el registro ("historia de compilación"). Los resultados se presentarán tanto de forma numérica como gráfica.

Se platearán varios tipos de estadísticas que tienen utilidad para los supervisores y los desarrolladores que usen la herramienta, y hacer que estén disponibles para los usuarios. Se debe hacer que las estadísticas sean fáciles de interpretar, para ello se pueden emplear gráficas que reflejen de una forma clara los resultados obtenidos en las estadísticas.

El sistema de estadísticas debe de ser fácilmente ampliable y configurable.

A la hora de realizar las estadísticas, se tendrá en cuenta el tipo de usuario. Si el usuario es un desarrollador sólo podrá usar sus datos y los del conjunto de los usuarios. Mientras que un supervisor además podrá tener acceso a los datos de cualquier usuario de forma individual.

7.4 Descripción de los actores y casos de uso

7.4.1 Actor Desarrollador

El actor Desarrollador representa a un usuario que realiza todo el ciclo de desarrollo de aplicaciones. Las acciones que tendrá a su disposición serán las de compilar un archivo, ejecutar sus aplicaciones. Podrá realizar estadísticas y gráficas sobre sus errores, los errores de su grupo de trabajo y los que tuvieron todos los usuarios del grupo desarrolladores colectivamente, pero no tendrá acceso a las del resto de desarrolladores individualmente, ni a la de los supervisores.

7.4.2 Actor Supervisor

El actor Supervisor representa a un supervisor del trabajo de los desarrolladores. Tendrá a su disposición todas las posibilidades de un usuario desarrollador, y además tendrán acceso a todas las estadísticas sobre los errores individuales de todos los usuarios del sistema.

7.4.3 Análisis estático

Este módulo permite realizar una exploración del código fuente para verificar su corrección. Utiliza el compilador y otros procesadores de lenguaje para realizar el análisis estático del código fuente.

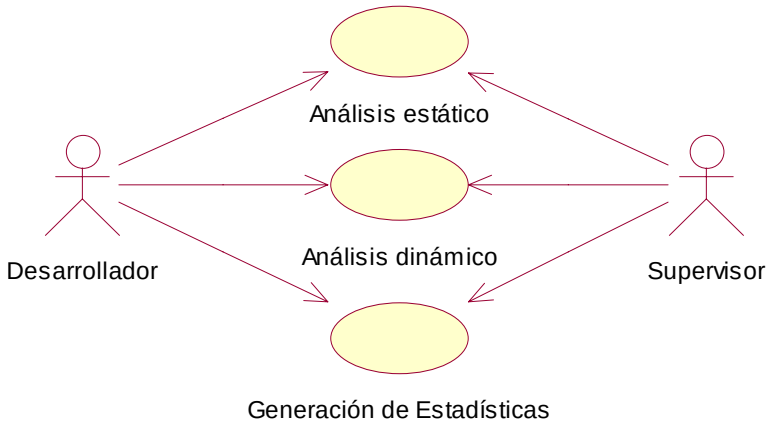


Figura 27. Diagrama de casos de uso del prototipo de análisis de errores de programas (PBA)

Los análisis son lanzados por los usuarios del sistema. Es técnicamente posible lanzar este análisis desde línea de comandos; pero lo normal es que sea otro subsistema del sistema SICODE el que lo haga. Como entrada para la ejecución de este módulo se deberá especificar:

- La unidad de compilación con la que se quiere compilar los archivos.
- La ruta donde están los archivos fuente del proyecto que se desea analizar.

Una unidad de compilación, permite indicar al sistema los directorios donde se encuentran los archivos fuente y donde tiene que dejar los resultados; las herramientas que debe utilizar el sistema, las opciones de estas herramientas y en que orden debe utilizarlas; incluso permite condicionar el uso de determinadas herramientas a que otras proporcionen determinados resultados. Entre las herramientas utilizadas siempre se incluye un compilador; de ahí el nombre de unidad de compilación. Se pueden crear tantas unidades de compilación como se crean convenientes y cada una tiene un identificador único con lo que se pueden utilizar como bloque que unifica todo el trabajo que debe realizar el sistema sobre el conjunto de archivos de un proyecto.

En caso de que durante el proceso de análisis se detecte errores en el código fuente, se guardará toda la información sobre el mismo en la BD en el registro de "historia de compilación", para que se pueda realizar un estudio sobre ella y mostrar los resultados al usuario a través de una interfaz.

7.4.4 Análisis dinámico

Un usuario lanza la ejecución de una clase. Se permitirá al usuario interactuar con ella haciendo llamadas a los métodos con distintos valores en sus parámetros, en caso de producirse una excepción se guardará toda la información sobre esta excepción en la BD. El análisis dinámico está pensado para hacer pruebas de ejecución de las clases sin tener que extraer estas clases del sistema.

7.4.5 Generación de estadísticas

Esta operación será lanzada por los usuarios del sistema. El usuario deberá indicar un archivo de configuración de estadísticas, donde se indicará: los datos que se deben analizar, la forma de confeccionarlas, y el archivo donde se guardará el resultado de las mismas.

Si el usuario que realiza la operación es un desarrollador, podrá usar para la confección de las estadísticas sus datos y los datos de todos los usuarios tomados en conjunto. Si el usuario es un supervisor, además podrá usar para las estadísticas los datos de todos los usuarios del sistema tomados individualmente.

7.5 Escenarios del prototipo

7.5.1 Análisis estático

El usuario ordena el análisis de los archivos fuente de un proyecto concreto con una unidad de compilación determinada. En esa unidad de compilación, además del compilador se podrán configurar herramientas para la comprobación del código fuente, que se puede usar para indagar posibles errores en la aplicación no detectados por el compilador. El resultado del análisis será uno de los siguientes:

1. Los parámetros del análisis son erróneos. Se indicará al usuario que error hay en los parámetros y además se le mostrará la forma de uso correcta.
2. Al realizar el proceso de compilación y ejecución de las herramientas de comprobación de código se produce un error de ejecución por una configuración errónea. Se mostrará al usuario que se ha producido un error durante este proceso, así como el mensaje de la excepción.
3. El proyecto compila correctamente y no tiene ningún aviso. La compilación finaliza correctamente y las herramientas adicionales no encuentran ningún error, por lo que se le indicará al usuario que no hay errores de compilación y que no se produjo ningún aviso.
4. El proyecto compila correctamente, pero tiene avisos. Se muestra un mensaje de compilación correcta, pero se han detectado algún posible error en la aplicación, y se da un aviso sobre él. Se le ofrecerá al usuario un enlace a la información sobre ese aviso.
5. Algún archivo del proyecto no compila. Se indica al usuario que el archivo tiene errores de compilación, y se indican los errores detectados, así como un enlace a información sobre los mismos.

Para almacenar información sobre el análisis y poder consultarla y procesarla posteriormente a cada análisis se la asignará un código de identificación. El código del análisis se hará corresponder con el orden de los análisis para ese usuario, es decir, el primer análisis de un usuario deberá ser el análisis 0 y de ahí en adelante se les dará de código a cada uno de forma correlativa.

7.5.2 Análisis dinámico

Un usuario podrá lanzar la ejecución de sus aplicaciones. El sistema se ocupará de capturar la salida de la aplicación y de los errores que se pudieran provocar durante la ejecución.

1. Los parámetros de la ejecución son erróneos. Se indicará al usuario que error hay en los parámetros y además se le mostrará la forma de uso correcta.

2. La clase se ejecuta sin lanzar excepciones. En caso de que no haya ninguna excepción se mostrará la salida de la aplicación, así como un mensaje indicando que la ejecución no produjo excepciones.
3. La clase finaliza con una excepción. Se mostrará la salida de la ejecución de la aplicación hasta el momento en el que se produjo la excepción. Además se mostrará un mensaje con la excepción producida y un enlace a la información sobre dicha excepción.

7.5.3 Generación de estadísticas

El usuario deberá indicar un archivo de configuración de estadísticas, donde se indicará la forma de confeccionarlas, y el archivo donde se guardará el resultado de las mismas. A partir de esta información y los datos de errores y avisos almacenados en la base de datos el módulo genera informes con información numérica y gráfica dependiendo de la configuración elegida.

Escenarios según el tipo de estadísticas que queremos generar:

1. Generación de una estadística de Media de Errores en un Periodo. En el archivo de configuración se manda generar unas estadísticas sobre la media de errores de algunos usuarios durante un periodo dado. Los datos requeridos por el usuario son accesibles para él y se puede generar las estadísticas que se presentarán en la página Web resultante.
2. Generación de una estadística de evolución de la Media de Errores. En el archivo de configuración se manda generar unas estadísticas sobre la evolución de la media de errores. Los datos requeridos por el usuario son accesibles para él y se puede generar las estadísticas que se presentarán en la página Web resultante.
3. Generación de una estadística de Errores más Frecuentes en un Periodo. En el archivo de configuración se manda generar unas estadísticas sobre los errores más frecuentes de algunos usuarios durante un periodo dado. Los datos requeridos por el usuario son accesibles para él y se puede generar las estadísticas que se presentarán en la página Web resultante.

En la generación de estadísticas se pueden dar los siguientes escenarios de error:

4. Los parámetros de la estadística son erróneos. Se indicará al usuario que error hay en los parámetros y además se mostrará al usuario la forma de uso correcta.
5. El archivo de configuración contiene errores. Se indicará al usuario que en el archivo se han detectado errores en el archivo de configuración, indicando la línea donde se produce el error y el tipo de error encontrado.
6. Un desarrollador intenta acceder a la información individualizada de otros usuarios del sistema. En caso de que un desarrollador intente obtener información de otros desarrolladores de forma individual, se parará la ejecución del archivo de configuración y se avisará de que se está intentado acceder a una información para la cual no se tienen permisos.

7.6 Herramientas de búsqueda de errores utilizadas en el proyecto

En el apartado 3.8.4. Herramientas análisis estático de código del Capítulo 3 se ha realizado un análisis detallado de las herramientas de análisis de código fuente más utilizadas en la

actualidad. En el desarrollo de este prototipo hemos seleccionado algunas de estas herramientas para su integración en el sistema. A continuación enumeramos estas herramientas y las razones por las que las hemos elegido:

- **Jlint** [Knizhnik 2003]. Los avisos que generados por esta herramienta son interesantes, permiten la detección de algunos errores difíciles de encontrar. Está implementada en C++, lo cual hace que sea muy eficiente respecto las que están implementadas en Java. Su código fuente es compilable en Linux y Windows, lo que hace que pueda ser usada en varias plataformas.
- **Antic** [Knizhnik 2003]. Los errores detectados por esta herramienta son errores muy simples que se suelen presentar sobre todo en las primeras fases de aprendizaje de la programación. Al igual que la anterior es muy eficiente puesto que está implementada en C, pero puede ser compilada bajo Windows y Linux. Además de Java, puede analizar código C y C++.
- **Findbugs** [Hovemeyer 2004a]. Esta herramienta usa el concepto de Patrón de Error, lo que la hace interesante desde el punto de vista teórico. Comprueba la existencia de los patrones en el código fuente. Algunos de estos patrones no son aplicables en el ámbito del aprendizaje de la programación, pero permite que estos no sean usados en el análisis, seleccionando los que sean de interés.
- **PMD** [PMD]. Esta herramienta también permite definir patrones de error para su detección y comparte algunos de los tipos de análisis que realizan FindBugs y Jlint, pero proporciona muchos tipos de avisos que no son tratados por las otras como avisos sobre estilo o sobre código innecesario o ineficiente.

En el estudio aparecían más herramientas que finalmente hemos descartado, exponemos a continuación las razones para descartarlas:

- **Jwiz** [Geis 1999]. Fue descartada debido a que proporciona una documentación pobre.
- **DoctorJ** [DoctorJ]. Descartada al no tener ningún tipo de análisis distinto de los que hace Jlint. Además está implementada en C++ y sólo dispone de una versión para Linux.
- **Checkstyle** [Checkstyle] y **JCSC** [JCSC]. Estas dos herramientas tienen el mismo sistema de funcionamiento que PMD, y en muchos casos realizan el mismo tipo de análisis de código. La elección de PMD frente a las otras dos se debe a que PMD se centra más en generar avisos sobre posibles errores, mientras que Checkstyle y JCSC se centran más en comprobar que el documento sigue un buen estilo.

7.7 Estadísticas de análisis de errores

Es muy importante disponer de una gran flexibilidad a la hora de generar estadísticas; por ello, los usuarios dispondrán de la posibilidad de configurar una serie de estadísticas sobre sus errores de compilación o avisos detectados por el sistema al realizar el análisis estático. Todas las métricas y estadísticas se podrán elaborar para un usuario individual, o para todos los usuarios del sistema conjuntamente. En el caso de desarrolladores el usuario individual será el propio usuario; pero en el caso de los supervisores podrán seleccionar el usuario sobre el cual elaborarlas.

7.7.1 Consideraciones generales para las estadísticas

Para realizar las estadísticas se necesita un periodo que delimite los datos que van a ser analizados. En el sistema se permitirán dos formas de indicar el periodo. Por un lado se podrá especificar de forma temporal, y por otro se podrá indicar por el código de análisis. Por ejemplo se permitirá expresar los periodos de las siguientes formas:

- 4 de Agosto de 2003 a 16 de Agosto de 2003
- Análisis 23 hasta el análisis 65.

Además se permitirá usar periodos temporales relativos, es decir, que no se basen en una fecha en concreto. Por ejemplo, se podrá indicar que el periodo es:

- Desde hace quince días hasta hoy.

En cuanto a las medidas de los periodos, para los periodos temporales se usará el día como unidad de medida, mientras que para los análisis se usarán el número de estas.

Se podrán crear conjuntos de errores en los cuales se podrá indicar los errores que pertenecen o no pertenecen por medio de su código, su tipo o la herramienta que los genera. Por ejemplo, se permitirá expresar los siguientes conjuntos de errores:

- Errores de Compilación.
- Error 122, Error 345 y Error 346 (por el código de error).
- Avisos de sincronismo que no han sido generados por Findbugs (por el nombre de la herramienta).
- Avisos de PMD menos el error 12330 (todos excepto el error del código dado).
- Todos los avisos que no han sido generados por Antic (excepciones por herramienta).

De la misma forma, se podrá especificar que se usen sólo los datos de unas determinadas unidades de compilación para generar las métricas.

7.7.2 Tipos de estadísticas disponibles

Para crear los tipos de estadísticas hemos intentado ante todo que tengan utilidad para los desarrolladores y los supervisores de forma que se puedan usar para prevenir errores típicos y poder conocer la evolución de los desarrolladores a lo largo de un periodo.

Cada una de las estadísticas proporcionadas tendrá una gráfica asociada que simplifica la lectura de los datos. Estas gráficas deberán tener una leyenda para mejorar su comprensión por parte de los usuarios.

Errores más frecuentes durante un periodo

Esta métrica nos permite conocer cuales han sidos los errores más frecuentes de un usuario en un determinado periodo. Los resultados se proporcionarán en errores por análisis. Desde el punto de vista de los desarrolladores puede servir de recordatorio de sus fallos más comunes de forma que puedan prevenirlos mediante el estudio de sus soluciones. Desde el punto de vista de los supervisores puede servir para tener una visión general de los errores más frecuentes de los diferentes desarrolladores y de todo el sistema en general. Con esta información puede preparar mejor las clases futuras sabiendo de antemano donde están los mayores problemas.

Los parámetros de este tipo de estadística serán:

- El número de errores más frecuentes. Deberá ser un entero entre 1 y 10. Por defecto 5.
- El periodo que se va a analizar.
- Los usuarios sobre los que se va a realizar.
- Los conjuntos de errores que van a intervenir.
- El periodo de los análisis que se van a usar.

Ejemplos:

- 5 errores de compilación más frecuentes durante la semana pasada.
- 10 avisos de Antic más frecuentes entre los análisis 340 y la 360.
- El aviso de Jlint más frecuente de todos los análisis.
- 6 errores de g++ más frecuentes de las compilaciones de la unidad de compilación "compila_cpp".

La gráfica asociada a esta métrica será una gráfica de barras donde se represente el nivel de errores por análisis de cada error. En el informe de ejemplo del apartado 7.9.2 la cuarta estadística es de este tipo, que se corresponde con los 8 errores más frecuentes de cualquier tipo entre los análisis 0 y 10. En la Figura 44 se representa la gráfica de la estadística y en la Figura 45 la tabla con los datos numéricos.

Media de errores en un periodo

Se calculará para un conjunto de errores la media de errores por análisis en un determinado periodo. Esta métrica permitirá la comparación de los niveles de error de los usuarios en el sistema. Además permite conocer cuales son las herramientas más útiles o los conjuntos de errores que más nivel de error tienen.

Los parámetros de este tipo de estadística serán:

- El periodo donde se va a realizar.
- Los usuarios sobre el que se va a realizar.
- Los conjuntos de errores que van a intervenir.
- El periodo de los análisis que se van a usar.

Ejemplos:

- Media de errores de compilación en el mes pasado.
- Media de avisos de diseño de FindBugs desde el análisis 200 hasta el análisis 220.
- Media de avisos de PMD del 10 al 13 de Junio.

La gráfica asociada a la media de errores en un periodo será una gráfica de barras donde se represente el porcentaje de errores por análisis de cada conjunto de error. En el informe de ejemplo del apartado 7.9.2 la tercera estadística es de este tipo, representa la frecuencia para dos conjuntos de errores: todos los errores excepto los de la herramienta JLint, todos los errores que sean de JLint o sean del tipo de sincronización; el periodo de análisis empieza el 10 de septiembre de 2004 y finaliza el 17 de septiembre de 2004; además se calcula para poder compararlo los errores para un usuario individual y para todos los usuarios. En la

Figura 41 aparece la representación gráfica y en la Figura 42 aparece la tabla con los datos numéricos.

Evolución de media de errores

Básicamente es la misma idea que la métrica anterior, sólo que tomada en varios periodos consecutivos. Por lo tanto permite tener una visión de como se comporta la media de errores en el tiempo (ya sea medio en unidades temporales o en el número de análisis).

Los parámetros de este tipo de estadística serán:

- Número de periodos en los que se va a realizar.
- Tamaño del periodo.
- Los usuarios sobre los que se va a realizar.
- Los conjuntos de errores que van a intervenir.
- El periodo de los análisis que se van a usar.

Ejemplos:

- Media de errores de compilación de cada mes en los últimos 5 meses.
- Media de avisos de sincronismo cada 20 análisis desde el análisis 350.
- Media de avisos de Jlint en esta semana y en las 10 semanas anteriores.

La gráfica asociada será una gráfica de líneas donde cada punto represente un periodo. Estos puntos estarán unidos por líneas. En el informe de ejemplo del apartado 7.9.2 la segunda estadística es de este tipo, representa la evolución de la media de errores para un conjunto de errores: todos los errores de tipo error de compilación; el periodo total del análisis empieza el 26 de agosto de 2004 y finaliza el 9 de septiembre de 2004, dividiendo todo en subperiodos de 200 análisis. En la Figura 38 aparece la representación gráfica y en la Figura 39 aparece la tabla con los datos numéricos.

7.8 Diseño

7.8.1 Definición de la secuencia de utilización de las herramientas de análisis

Uno de los requisitos fundamentales de este sistema es la organización de la ejecución del compilador y de las herramientas de análisis estático a la hora de realizar un análisis. Normalmente deberíamos poder configurar varias herramientas de análisis y permitir configurar un número indeterminado de ellas. Además, debemos definir una secuencia de ejecución de estas herramientas, ya que algunas utilizan los resultados de otras. Por otra parte, es necesario definir las opciones de ejecución de cada una de las herramientas. Además, es necesario gestionar los resultados situándolos en un directorio temporal. Todo esto respetando la portabilidad del sistema para que pueda ser ejecutado, tanto en máquinas finales de los usuarios como en servidores. Por último, es necesario controlar que no haya problemas en el proceso y si los hay tener un mecanismo para notificarlos.

Para solucionar este problema se pensó en un primer momento en guardar la información sobre como organizar los análisis en la Base de Datos del sistema. Frente a esta solución se pensó en el uso de Ant [Ant 2003]. Como pudimos comprobar, Ant es una herramienta de

gran utilidad a la hora de organizar la ejecución de las tareas. Al final, se uso Ant, puesto que se adaptaba mucho mejor a los requisitos del proceso que debemos de realizar.

La herramienta Ant [Ant 2003] es una herramienta de software libre desarrollada por Apache Software Foundation. Esta es una herramienta del tipo de "make" cuya función es permitir la definición de una secuencia de ejecución de comandos necesarios en la compilación, despliegue, prueba y distribución de una aplicación. Pero que supere parte de los inconvenientes que tiene el "make" original, como la falta de portabilidad y la fuerte dependencia del sistema operativo. Esta completamente implementada en Java, por lo que puede ser usada en cualquier plataforma. Para especificar los objetivos y el orden de los mismos se usa un archivo de construcción codificado en XML.

Cada archivo de construcción define la forma de tratar el proyecto de una aplicación, que está compuesto por objetivos (targets), los cuales pueden depender unos de otros. En cada uno de estos objetivos se realizan tareas (tasks), las cuales están implementadas por una clase Java. Se dispone de muchas tareas predefinidas, relacionadas con las principales tareas de trabajo con archivos y compilación en el proceso de desarrollo. También se pueden establecer propiedades dentro del archivo de construcción, pasárselas como parámetros mediante la línea de comandos, y usar las propiedades de sistema.

Ant dispone de varias características que lo hacen muy versátil:

- Gran cantidad de tareas (Task) predefinidas.
- Fácilmente ampliable. La creación de nuevas tareas y la extensión de los existentes es algo tan sencillo como la programación de una clase Java.
- Uso desde Java. Las tareas de Ant pueden ser usadas como una clase más desde nuestro código Java, permitiendo la reutilización del código existente.
- Buena documentación. Ant dispone de una buena documentación, lo que facilita tremendamente su uso.

7.8.2 Empleo de la herramienta Ant en el análisis estático

La herramienta Ant es una de las piezas clave en el sistema de análisis, puesto que se encarga de todo lo referente a la ejecución de distintas herramientas especificadas en las unidades de compilación.

En un primer momento el sistema de análisis ejecutaba Ant desde la línea de comandos mediante la clase Runtime, esto presentaba un serio problema de comunicación entre las clases, puesto que en caso de producirse un error en la ejecución de Ant, no se tenía acceso a la información sobre el error, es más, ni siquiera se tenía constancia de que se hubiese producido tal error. Tras estudiar el código fuente y la documentación de Ant, se llegó a la conclusión de que se podía ejecutar directamente un archivo Ant desde el código Java del proyecto usando la tarea "Ant". Por lo tanto, la solución a este problema fue crear en la clase Compilacion que agrega clases del proyecto de Ant y permitir ejecutar la tarea "Ant" con los parámetros adecuados para realizar la ejecución del script y acceder a todos los errores que se produzcan en la ejecución de Ant.

Algunas de las herramientas utilizadas proporcionan una tarea Ant. Viendo que la herramienta Ant ofrecía muchas posibilidades, se buscó la forma poder ejecutar también, las aplicaciones que no tenían interfaz Ant. En un primer momento, se usó para ello la tarea "Exec" de Ant, la cual permite la ejecución de comandos. Sin embargo, se pensó que era mucho más elegante el crear unas tareas para estas herramientas, lo cual simplifica mucho la edición de los archivos Ant de las Unidades de Compilación. Por ello, se estudió la forma en que se pueden crear tareas Ant. En ese momento se crearon las clases

AnticTask, JlintTask y AnalizadorTask. Esta última hereda de la clase Task, mientras que las dos primeras heredan de la clase MatchingTask. Se eligió esa clase porque está pensada para que hereden de ellas las clases que quieran trabajar con paths y conjuntos de archivos.

El último problema que se tuvo que resolver en el Ant fue que la tarea predefinida "Javac" no ofrece la posibilidad de redirigir la salida del compilador hacia un archivo. Esto hace que no sea posible el manejo de los errores detectados por el compilador. La solución que se dio a este problema fue la creación de una nueva tarea, JavacTask, que permitiera realizar las compilaciones y además guardar la salida a un archivo. Para construir esta clase se emplearon los conocimientos adquiridos en la implementación de AnticTask y JlintTask.

Diagrama de clases de la Clase Compilación

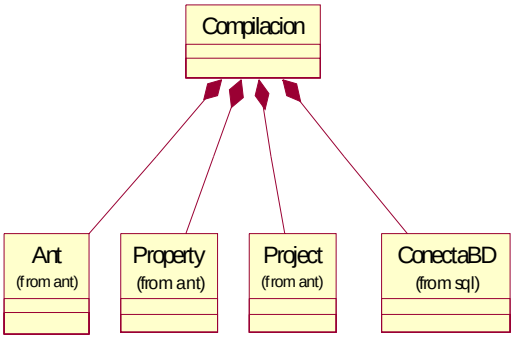


Figura 28. Diagrama de clases de las relaciones de la clase compilación

La clase Compilación (Figura 28) se encargará de realizar los análisis en el sistema. Para ello usa clases ajenas procedentes de la herramienta Ant, las cuales son usadas para la ejecución del script Ant con el que se configura la Unidad de Compilación que se le indicó a la clase Compilacion. La clase ConectaBD es una clase de nuestro proyecto la cual se usa para la comunicación con la Base de Datos.

Diagrama de clases del paquete proyecto.utilidad.ant

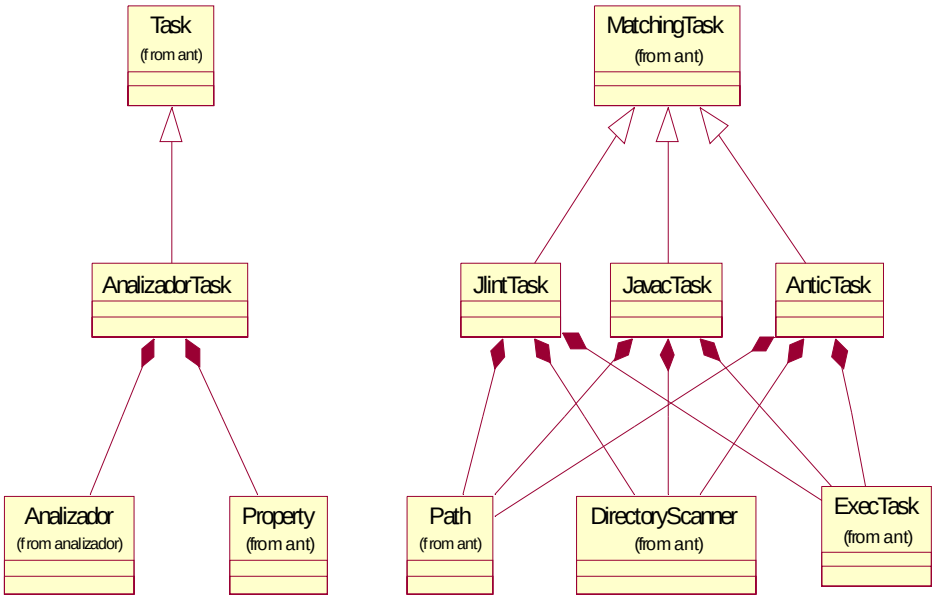


Figura 29. Diagrama de Clases del paquete proyecto.utilidad.ant

AnalizadorTask (Figura 29) es una clase que hereda de Task, lo que permite su ejecución desde un script Ant. El objetivo de esta clase es el proporcionar un interfaz Ant a la clase Analizador del paquete proyecto.analizador. Para este trabajo usa las clases Analizador, que se encarga de realizar el análisis al archivo indicado, y la clase Property de Ant, la que hace posible el uso desde el script Ant del número de errores o avisos que había en el archivo analizado.

Las clases JlintTask, JavacTask y AnticTask son muy similares. Las tres tratan de ser interfaces para que una herramienta sea ejecutada desde Ant. Estas herramientas son Jlint, Javac y Antic respectivamente. Estas clases heredan de MatchingTask, por lo que, además de poder ser ejecutadas desde Ant, heredan de una serie de métodos y campos muy útiles. En estas clases se usarán clases de Ant para simplificar las tareas que se tienen que realizar, como la localización de los archivos fuentes en un árbol de directorios o la propia ejecución de la herramienta.

Diagrama de clases del paquete proyecto.analizador

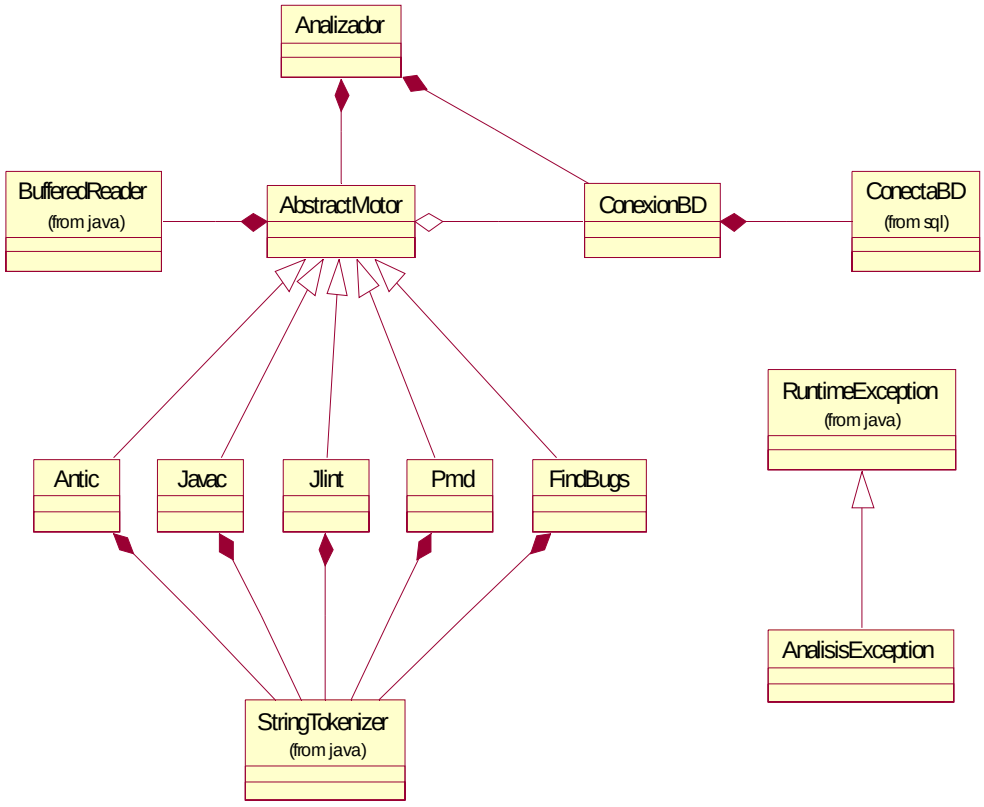


Figura 30. Diagrama de clases del paquete proyecto.analizador

En el diagrama de la Figura 30, se pueden apreciar todas las relaciones que tienen las clases del paquete proyecto.analizador la cuales resuelven el problema del análisis de los archivos de errores o avisos de varias herramientas.

Para la conexión con la Base de Datos se usa la clase ConexionBD que proporciona un interfaz simple para las operaciones que realizan las clases del paquete con la Base de Datos. Esta clase usa ConectaBD del paquete proyecto.utilidad.sql para realizar la conexión propiamente dicha.

En el diagrama se puede ver una jerarquía de herencia que tiene como cúspide la clase AbstractMotor. Ésta es una clase abstracta que se usa para simplificar la tarea de crear

motores de análisis de archivos de errores o avisos de las herramientas. Esta clase usa la clase BufferedReader de la API de Java para poder leer el archivo. Además utiliza ConexionBD para guardar los errores analizados. Todos los motores de análisis de errores deberán heredar de esta clase.

Cada uno de sus hijos son clases que implementan motores de análisis de archivos de error de herramientas. Cada una de estas clases lleva el nombre de la herramienta para la que está diseñada, de forma que sean fácilmente reconocibles. Todas estas clases usan la clase StringTokenizer de la API estándar de Java.

La clase principal del paquete es Analizador, la cual servirá de interfaz con el resto del proyecto. Esta usa las clases que heredan de AbstractMotor para realizar el análisis y también usa la clase ConexionBD para poder realizar consultas en la Base de Datos.

Por último hay una excepción AnalisisException, la cual hereda de RuntimeException. En esta excepción se guardarán los problemas que se puedan producir durante el análisis y se lanzarán para ser tratados por entidades superiores.

Diagrama de secuencia para el escenario: El archivo compila correctamente, pero tiene avisos

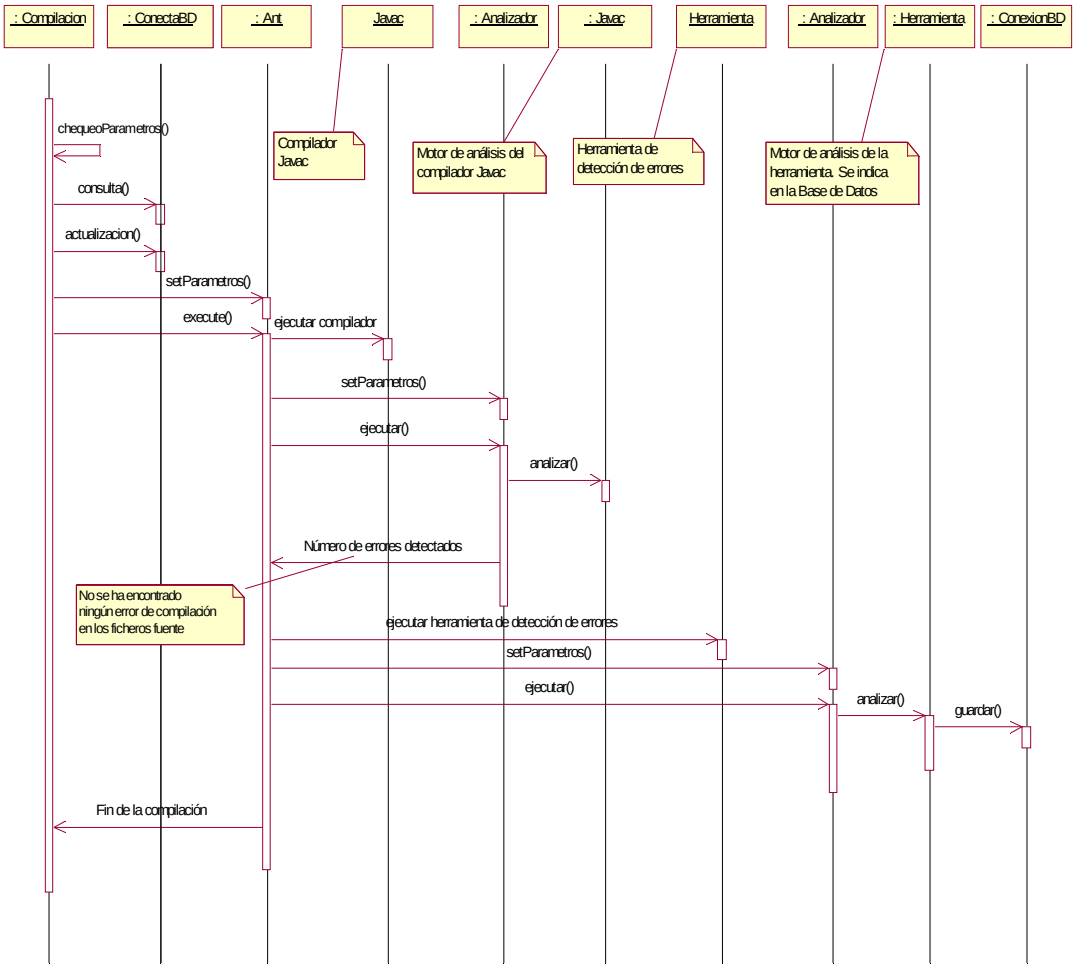


Figura 31. Diagrama de secuencia de un archivo que compila sin errores; pero otras herramientas de análisis generan avisos

En este caso tenemos un análisis en la que el compilador no detecta errores de compilación, lo cual hace que se puedan ejecutar las herramientas de detección de errores, las que detectan posibles errores en el código fuente. Esta información será guardada en la Base de Datos.

7.8.3 Empleo de la herramienta Ant en el análisis dinámico

El sistema de ejecución o análisis dinámico utiliza una filosofía muy similar para ejecutar las clases de los usuarios. Anteriormente se pensó en la creación de clases que trabajaban con hilos de ejecución para llevar a cabo la ejecución de las clases de los usuarios. El problema es que esta solución requería conocimientos sobre aplicaciones multihilo y sincronismo lo cual es muy costoso en tiempo. Por lo tanto, se buscó otra vía para solucionar el problema y esta solución se encontró otra vez en Ant. En este caso, se usa la tarea predefinida "Java". Esta tarea proporciona muchas funcionalidades y permite un tratamiento sencillo de las excepciones lanzadas durante la ejecución de la clase del usuario.

7.8.4 Diseño arquitectónico

Este subsistema está diseñado para su funcionamiento en segundo plano invocado por otros módulos del sistema SICODE, en concreto este subsistema será utilizado por el entorno de desarrollo integrado en Web (IDEWeb) (ver Capítulo 9), a través del cual, el usuario lanzará el análisis y la compilación para obtener los archivos en código objeto a partir del código fuente escrito. También, se utilizará este mecanismo para la fase de pruebas de ejecución, donde el entorno de desarrollo permitirá invocar el código ya compilado para realizar las pruebas oportunas. Todos los resultados, tanto de análisis estático del código fuente como dinámico de la ejecución, se almacenarán en una base de datos. Estos datos serán analizados por este subsistema y los resultados de este análisis podrán ser mostrados al usuario, bien a través de un navegador en una página Web independiente, o bien a través del entorno de desarrollo integrado (IDEWeb), de nuevo.

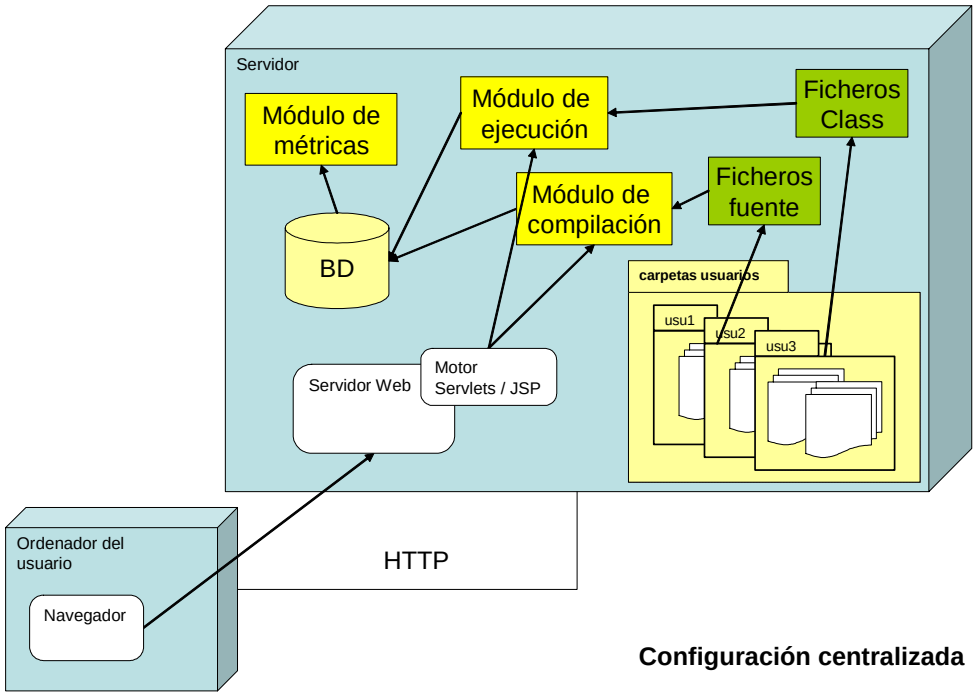


Figura 32. Diagrama que representa la arquitectura del sistema en una configuración centralizada

La independencia con la interfaz de usuario y la separación entre los módulos de análisis y el de generación de métricas permiten una gran flexibilidad en la configuración de la arquitectura del sistema. Planteamos aquí dos alternativas posibles en esta configuración:

- Arquitectura centralizada (Figura 32). En esta configuración todos los módulos residen en el servidor del sistema. Reciben las peticiones desde el entorno de desarrollo, que también podría residir en el servidor, y la interfaz con el usuario consistiría en un simple navegador, que se comunicaría con el servidor utilizando un protocolo HTTP. En esta configuración, los archivos de los proyectos que están desarrollando los usuarios, también residen en el servidor. Esta arquitectura permite una independencia total del usuario respecto al lugar de conexión, simplemente con una conexión a Internet y un navegador estándar, el usuario podría acceder a todos los servicios del sistema y al estado actual del trabajo que está desarrollando, sin necesidad de instalaciones de software adicional, ni de llevar en algún tipo de almacenamiento extraíble los ficheros del proyecto.

- Arquitectura distribuida (Figura 33). Gracias al planteamiento de módulos independientes, se puede optar por distribuir los módulos de análisis junto con un entorno de desarrollo a los ordenadores de los usuarios. Permitiendo la utilización de entornos de desarrollo más ricos y potentes, y enviando al servidor solamente los resultados del análisis para su almacenamiento en la base de datos. Esta comunicación puede realizarse mediante Servicios Web [Champbell 2001], que sobre un protocolo HTTP permita transmitir los resultados del análisis estático o dinámico que se realiza en el cliente. Es importante mantener la base de datos centralizada debido a la importancia de elaborar resultados de grupo y hacer análisis no sólo a nivel individual. El módulo de métricas que ofrece los resultados estadísticos seguirá situándose en el servidor ya que hace un uso intensivo de los datos almacenados en la base de datos, y no necesita de una interacción intensiva con el usuarios, simplemente la visualización de resultados gráficos y numéricos.

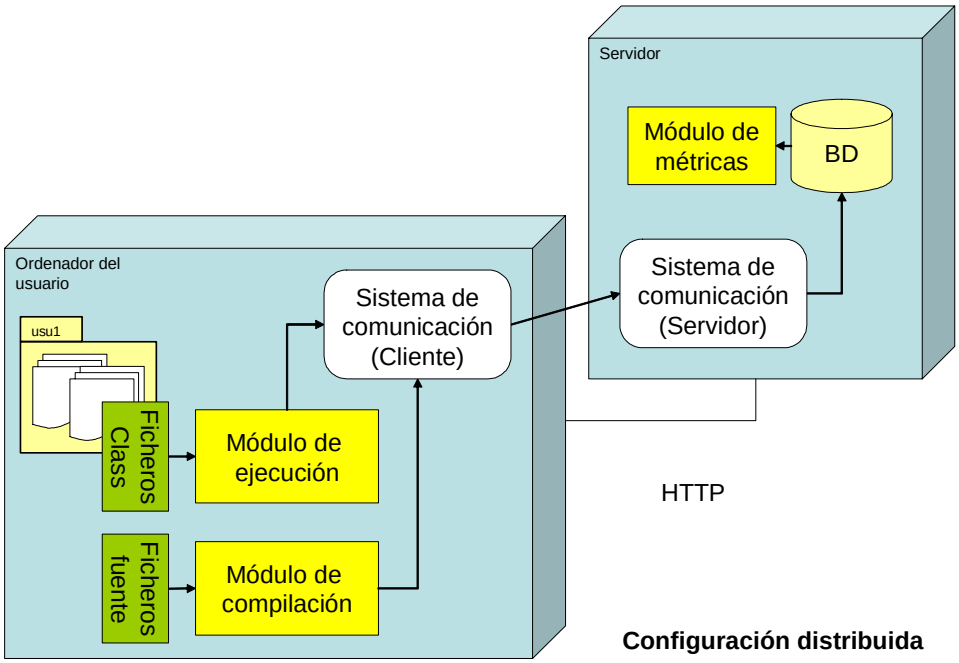


Figura 33. Diagrama que representa la arquitectura del sistema en una configuración distribuida

El sistema SICODE presentado en este documento utiliza la arquitectura centralizada; sin embargo, ya se dispone de prototipos que utilizan la segunda arquitectura presentada y se propone como una de las líneas de trabajo futuro (Capítulo 11) el potenciar esta arquitectura y explorar las mejoras que podemos obtener con ella.

7.8.5 Modelado de datos

Se considera importante reflejar el modelo de datos elegido para almacenar la información tanto sobre las herramientas como sobre los propios avisos almacenados, ya que sobre ella basaremos el análisis posterior y la construcción de los informes.

Debemos de tener en cuenta varios puntos a la hora de diseñar el modelo de datos. Por un lado, debemos de tener a nuestra disposición la información sobre cada una de las operaciones que puede llevar a cabo un usuario en el sistema: análisis estático utilizando diferentes unidades de compilación, análisis dinámico. Por otro lado, debemos de disponer de información que nos permita reconocer y clasificar los diferentes avisos y mensajes de error que pueden ocurrir al realizar estas operaciones. Por último, debemos relacionar todo, de forma que toda la información sea accesible.

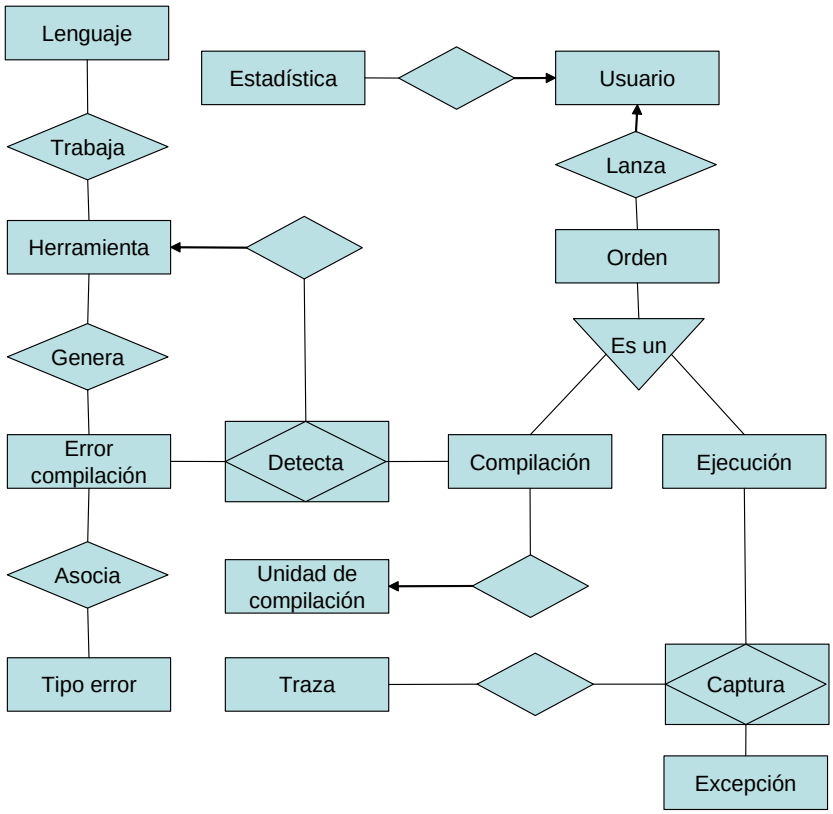


Figura 34. Diagrama entidad-relación subsistema de análisis de errores

Información sobre la actividad de los usuarios

El usuario puede realizar dos operaciones con el sistema: un análisis estático o una ejecución. De cada una de ellas, se deberá guardar información específica de cuándo y cómo se ha realizado, así como los mensajes que se han generado en cada caso. En un análisis, se guardan todos los errores de compilación o avisos detectados por la unidad de compilación en los ficheros de código fuente de un proyecto; mientras que en una ejecución, se guardan las excepciones que se producen durante la ejecución de la aplicación Java.

Información sobre la configuración del sistema

El sistema debe de tener constancia de los lenguajes y las herramientas con las que puede trabajar, así como la información sobre cada uno de los errores que pueden ser generados por cada una de las herramientas. Es esencial, disponer de información para poder clasificar correctamente los mensajes de error detectados para, posteriormente poder realizar estadísticas sobre los tipos de error. Por otro lado, se debe de disponer de una lista de las excepciones que puede detectar nuestro sistema.

Además, se deberá decidir como se almacenarán la información sobre los errores detectados en los análisis y las ejecuciones. También se debe almacenar la información sobre las herramientas que puede usar el sistema, los lenguajes de que se dispone e información sobre los usuarios. En la Figura 34 aparece el diagrama entidad – relación con todas las entidades y las relaciones existentes entre ellas.

En primer lugar, debemos relacionar la información que genera este subsistema con el resto de la base de datos, fundamentalmente la información sobre los usuarios. Esta relación se lleva a cabo a través de la entidad 'Usuario'.

La entidad 'Usuario' está relacionada con la entidad 'Estadística' mediante una relación de dependencia por existencia. La misión de esta entidad es el almacenamiento de algunos datos que pueden ser útiles para hacer estadísticas sobre los errores del usuario, o sobre el uso de la herramienta.

A su vez, la entidad 'Usuario' estará relacionada con la entidad 'Orden', la cual representa a las ordenes lanzadas por los usuarios. Estas órdenes podrán ser de dos tipos distintos, análisis estáticos representados por la entidad 'Compilación' y análisis dinámicos representados por la entidad 'Ejecución'. Las dos relaciones anteriores serían entidades hijas de la entidad 'Orden' en una relación de herencia.

La entidad ejecución estará relacionada con la entidad 'Excepción' mediante la relación 'Captura'. La entidad Excepción guardará la información sobre los distintos tipos de excepciones que pueden ser lanzadas por una aplicación. La relación 'Captura' guardará la información sobre la captura de excepciones durante la ejecución de una aplicación.

Mediante una agregación la relación 'Captura' se relaciona con la entidad 'Traza'. Esta última sirve para guardar la información sobre el volcado de la pila de llamadas por las que se fue propagando la excepción capturada.

La entidad 'Compilación' estará relacionada con la entidad 'Unidad de compilación' por una relación de dependencia por existencia. La 'Unidad de compilación' guardará la información sobre que pasos deben seguirse para llevar a cabo el análisis.

Por otro lado, la relación 'Compilación' se relaciona por medio de la relación 'Detecta' con la entidad 'Error de compilación'. La relación 'Detecta' guarda información sobre cada error de compilación o aviso encontrado por alguna herramienta durante un análisis determinado. 'Detecta' forma un agregado que se relaciona por relación de bitácora con la entidad 'Herramienta'.

La entidad 'Error de compilación' guarda la información sobre los errores que pueden ser detectados con nuestro sistema. Además de estar relacionada con 'Compilación', lo está también con la entidad 'Herramienta' a través de la relación 'Genera' y con la entidad 'Tipo Error' por medio de la relación 'Asocia'.

La entidad 'Tipo Error' guarda la información sobre los diferentes tipos en que se clasifican los errores y avisos detectados en el análisis.

La entidad 'Herramienta' guarda la información sobre los procesadores de lenguaje que emplea el sistema, como pueden ser compiladores o analizadores estáticos. Esta entidad tiene una relación con la relación 'Lenguaje' la cual guarda información sobre los diferentes lenguajes que pueden ser utilizados en la herramienta.

7.9 Archivos de estadísticas: Configuración y ejemplos

En este apartado, especificaremos la forma en la que se han resulto los requisitos establecidos para la generación de informes de estadísticas establecidos en el apartado 7.3.3 que se concreta en el escenario descrito en 7.5.3 y se detalla en 7.7.

7.9.1 Descripción de archivos XML para la configuración de las estadísticas

Los archivos de configuración de estadísticas proporcionan a este módulo una gran versatilidad. Están creados para que un usuario pueda crear fácilmente consultas complejas y avanzadas que se adapten a sus necesidades y luego pueda reutilizarlo siempre que lo necesite. En este apartado daremos es una visión general sobre las posibilidades que ofrece la configuración de estadísticas a los usuarios.

Empleo de XML y XML Schema

Cuando se pensó en la forma de configurar las estadísticas, se trato de buscar una forma de que estas fueran sencillas de utilizar para los usuarios y de procesar para el sistema. Por ello, se eligió XML [XML 2004] para guardar esa información. Puesto que un usuario puede leer fácilmente la configuración de las estadísticas y modificarlas, incluso con un simple editor de texto y de la misma manera el tratamiento de esa información se hace de forma estándar; además es posible validar el fichero.

Una vez elegido XML, se pensó en las posibles formas de validar los datos contenidos en el archivo. Las dos tecnologías más usadas actualmente son DTD y XML-Schema [XML Schema 2004]. La elección de XML-Schema se debe a que tiene características más avanzadas que DTD y parece que se va a imponer como estándar en un futuro próximo. Si se sabe diseñar bien el XML-Schema, uno puede ser capaz de validar los archivos XML con mayor precisión que un DTD. Desde el punto de vista del diseño del módulo de estadísticas esto presenta una ventaja enorme, puesto que no se necesita crear clases para el tratamiento de los errores del archivo, lo que es un ahorro de tiempo y esfuerzo, pero además, hace que el sistema sea menos dependiente de la forma que tengan los archivos de configuración de estadísticas. Esto hace que si se introducen nuevos tipos de estadísticas, no se deba modificar ninguna de las clases existentes, tan sólo se deberá de crear una nueva clase que interprete ese nuevo tipo.

XML-Schema es una tecnología relativamente nueva, por lo que se dispone de pocos analizadores (parsers) de XML que sean capaces de implementar la recomendación XML-Schema al completo. El único analizador que cumple la especificación XML-Schema al completo en el momento de desarrollar el prototipo es Xerces [Xerces 2004]. Se usó esta herramienta para chequear que los archivos XML validen el XML-Schema.

Después de la validación, se utiliza la tecnología DOM [DOM 1998] para acceder a los datos de los archivos, se eligió esta tecnología porque es muy sencilla de usar. Esta tecnología genera un árbol a partir del documento, el cual puede ser navegado usando métodos que permiten tratar con la estructura de datos.

Archivo de configuración de estadísticas

El archivo de configuración de estadísticas (ver apartados A.1 y A.2 del Apéndice A para ver dos ejemplos), es un archivo cuyo formato cumple la especificación XML 1.0 [XML 2004] y dispone de un archivo de XML-Schema (ver apartado A.3. para ver un ejemplo de este tipo de archivo) que permite su validación utilizando un analizador XML como Xerces.

El elemento *estadisticas* es el elemento raíz de los archivos, el cual puede tener anidados uno o más elementos que representan las diferentes estadísticas que queremos representar en el informe. Cada una de estas estadísticas está especificada por un elemento distinto. Para este prototipo se concibieron dos tipos de estadísticas distintas; pero el sistema permite que estos puedan ser ampliables en un futuro:

- Errores más frecuentes.
- Media de errores.

Hacemos, a continuación, una descripción de los atributos de cada estadística:

1. La estadística de Errores más frecuentes posee un atributo "numero" que representa la cantidad de tipos de errores más frecuentes que se desean mostrar, por defecto es 5. Su otro atributo "manejador" indica al sistema la clase Java que debe ser empleada para generar esta estadística. El valor por defecto es "proyecto.estadisticas.ErroresFrecuentes", pero puede ser cambiada por los usuarios.
2. La estadística de "Media de Errores" posee un atributo "evolucion" que representa el número de periodos para los que se desea conocer la media de errores, por defecto es 1. Su otro atributo "manejador" indica al sistema la clase que se debe emplear para generar esta estadística. El valor que tiene por defecto es "proyecto.estadisticas.MediaErrores", pero puede ser cambiada por los usuarios.

Los dos tipos de estadísticas anteriores tienen la misma estructura, que está compuesta por los cuatro elementos que se describen a continuación, siguiendo este orden:

1. Definición del elemento “periodo”. Indica los límites inicial y final de los datos que analizamos. Debe aparecer sólo en una vez.

   Habrá tres formas posibles de indicar los periodos:

   - Mediante un intervalo temporal relativo (unidad día). El final es hoy menos los días que se indiquen como retardo.
   - Mediante un intervalo temporal absoluto (unidad día). El final es la fecha indicada. El formato de la fecha es: AAAA-MM-DD (año - mes – día)
   - Mediante un intervalo entre análisis absoluto (unidad compilación). El final es el análisis indicado.

   Siempre habrá que especificar la duración del periodo, cuya unidad será el día o cantidad de análisis dependiendo del método que se use para indicar el periodo. El inicio de un periodo de final "f" y de duración "d" será "f-d"

2. Definición del elemento “usuarios”. El elemento “usuarios” sirve para indicar los usuarios para los cuales se va a realizar el estudio estadístico. Al igual que el anterior, debe aparecer una vez.

   La forma de incluirlos es indicando dentro de elemento anidado “usuario” el código de usuario que se desee estudiar. El elemento usuarios tiene dos atributos booleanos:

- "autor". Se usa para indicar que se haga la estadística para el usuario que lanza el estudio estadístico. Por defecto es "true".
- "todos". Se usa para indicar que se haga la estadística a todos los usuarios del sistema. Por defecto es "false".

3. Definición del elemento “errores”. Permite especificar que tipos de errores se incluirán en el análisis. Este elemento debe aparecer al menos en una ocasión, pero puede aparecer más de una vez.

Se puede indicar que errores se quiere usar en la muestra de estadísticas. En primer lugar se pueden meter condiciones de dos tipos:

- elementos "incluir", para usar los errores que cumplan una condición.
- elementos "excluir", para usar los errores que no cumplan una condición.

Si se especifica más de una condición se deberán indicar como se relacionan esas condiciones, para ello se emplean los elementos "and" y "or".

Las condiciones pueden ser de tres tipos:

- "codigo". Donde se indica el código del error.
- "tipo". Donde se indica un tipo de errores.
- "herramienta". Donde se indica la herramienta que generó los errores.

4. Definición del elemento “compilaciones”. Indica que unidades de compilación se quieren usar a la hora de hacer las estadísticas. Puede aparecer una vez o ninguna vez.

Como se puede ver la forma de configurar los archivos de estadísticas ofrece muchas posibilidades, pero de la misma forma es simple e intuitiva. En el Apéndice IV se presenta un archivo de ejemplo de como configurar las estadísticas, además ese archivo fue el usado para generar los resultados que se presentan en el siguiente apartado.

7.9.2 Descripción del informe de estadísticas

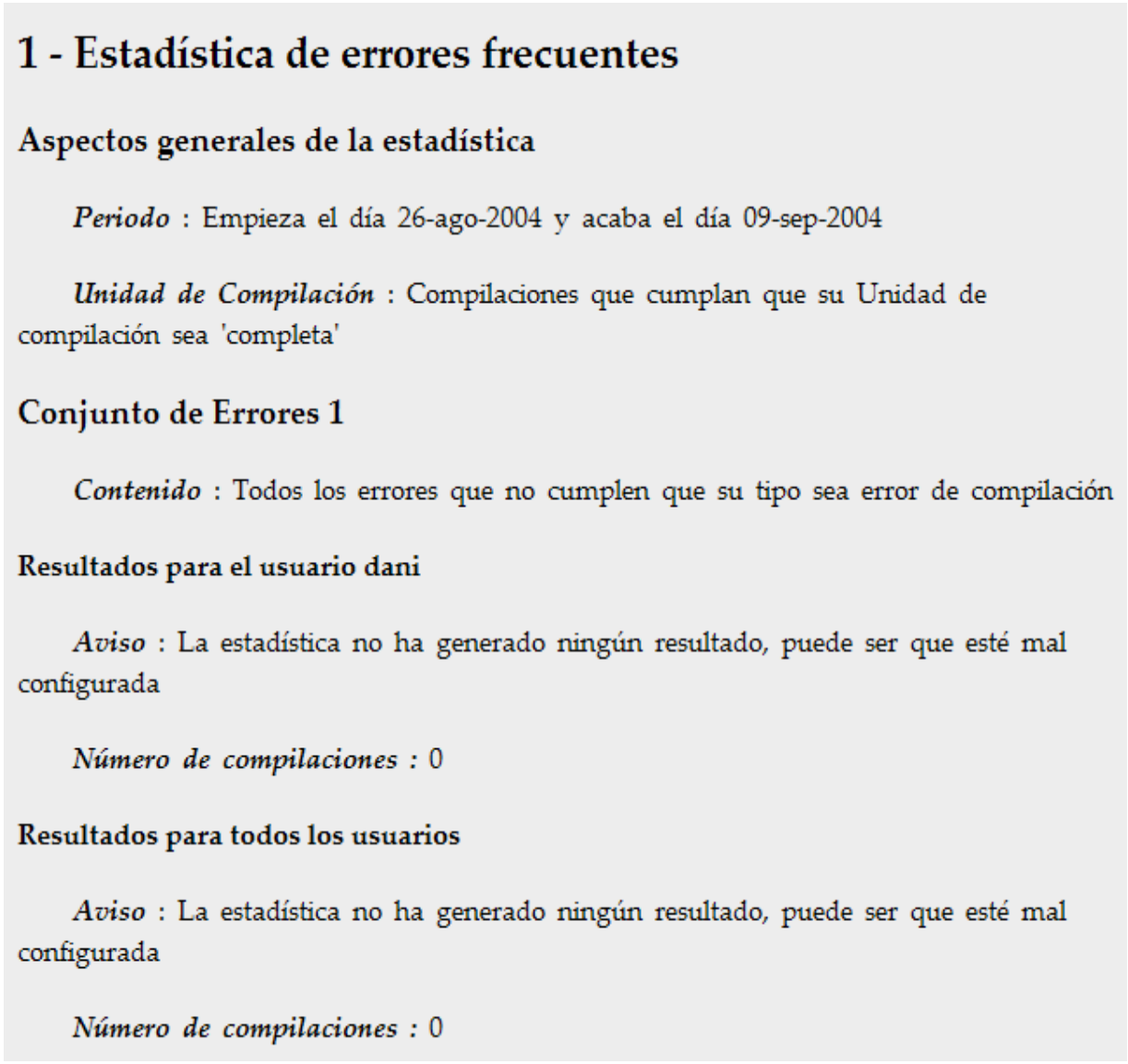
En este apartado hablamos de como se deben interpretar los informes que contienen los resultados de las estadísticas. Vamos a verlo con un informe de ejemplo resultado del procesamiento archivo de configuración de estadísticas que aparece en el apartado A.4. Archivo ejemplo de configuración de estadísticas del Apéndice A.

Figura 35. Esquema general del informe de estadísticas

Un esquema de la estructura del informe (Figura 35) es el siguiente:

- En primer lugar, el título principal del informe.
- Cada una de las estadísticas está numerada y aparecen consecutivamente, no hay límite en el número de estadísticas para un informe.
- Al comienzo se indica el tipo de la estadística: errores frecuentes o media de errores (en un periodo o la evolución a través de varios periodos).
- A continuación, se presentan una serie de datos sobre la estadística que dependen del tipo. Para errores frecuentes aparece: periodo, conjunto de errores y usuario; para media de errores: conjunto de errores y si no es una evolución, el periodo.
- Gráfica de los resultados del análisis, utilizando el tipo de gráfica adecuado al tipo de estadística realizado.
- Si la estadística realizada es una evolución de la media de errores, después de la gráfica, se presenta una descripción de los periodos para los que fueron realizadas las medias (Figura 39).
- Por último, se dan los resultados de la estadística de forma numérica mediante una tabla.

A continuación vamos a ir describiendo más detalladamente los aspectos destacables de cada una de las estadísticas del informe de ejemplo.



1 - Estadística de errores frecuentes

Aspectos generales de la estadística

Periodo : Empieza el día 26-ago-2004 y acaba el día 09-sep-2004

Unidad de Compilación : Compilaciones que cumplan que su Unidad de compilación sea 'completa'

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su tipo sea error de compilación

Resultados para el usuario dani

Aviso : La estadística no ha generado ningún resultado, puede ser que esté mal configurada

Número de compilaciones : 0

Resultados para todos los usuarios

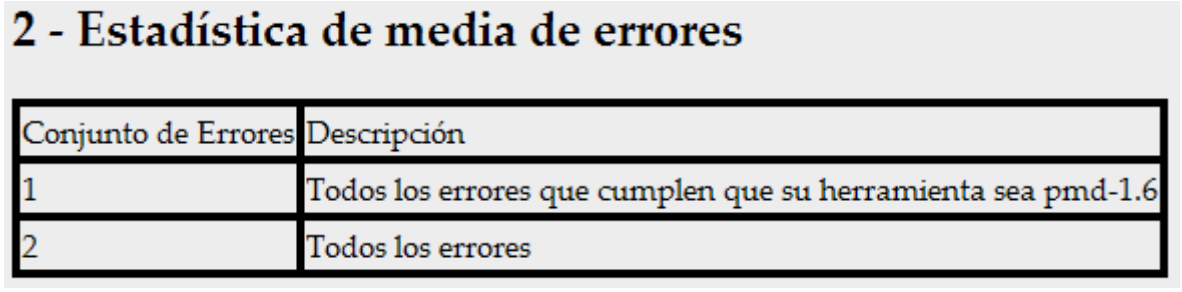
Aviso : La estadística no ha generado ningún resultado, puede ser que esté mal configurada

Número de compilaciones : 0

Figura 36. Resultados estadística errores frecuentes (primera) con posible error de configuración

En el informe la primera estadística que se intenta mostrar es del tipo de errores frecuentes (Figura 36). Sin embargo, este es un caso especial, puesto que las consultas que se han llevado a cabo no han proporcionado ningún resultado, lo que puede significar que la estadística esté mal configurada; aunque sintácticamente sea correcta, por lo tanto, se avisa al usuario para que compruebe que todo es correcto.

La siguiente estadística es de evolución de la media de errores, en ella se compara dos conjuntos de errores, que se indican en la tabla que hay detrás del título (Figura 37).



2 - Estadística de media de errores

Conjunto de Errores	Descripción
1	Todos los errores que cumplen que su herramienta sea pmd-1.6
2	Todos los errores

Figura 37. Información general de la segunda estadística (evolución media de errores)

La gráfica de líneas (Figura 38) muestra como se ha desarrollado la evolución de la media de estos dos conjuntos de errores, en los periodos que se establecieron en la configuración. En la leyenda se puede ver que colores se corresponden a cada pareja conjunto de errores – usuario.

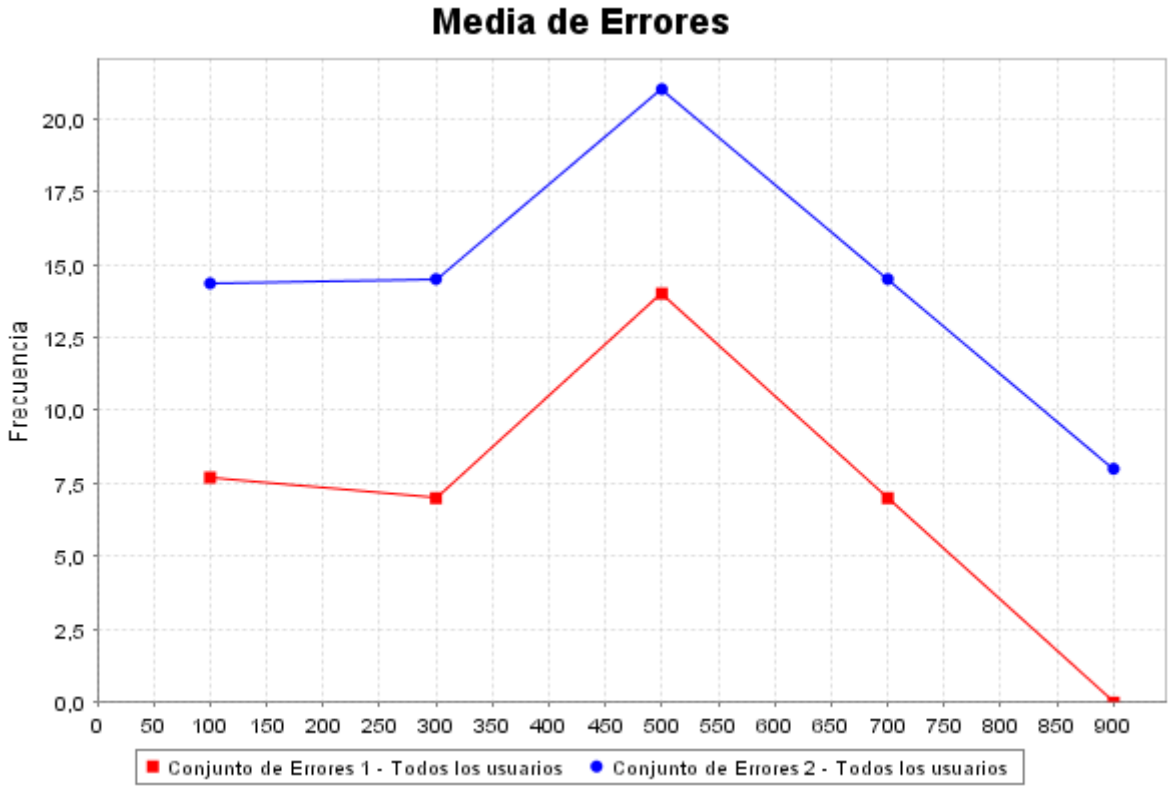


Figura 38. Gráfica de líneas de la evolución de la media de errores (segunda estadística)

Después de la gráfica, se presenta una descripción de los periodos para los que fueron realizadas las medias (Figura 39). Puede comprobarse que a la hora de representar la gráfica se tomó el punto medio del periodo como representante de todo el intervalo siguiendo los criterios estadísticos. A continuación, se dan los resultados de la estadística de forma numérica mediante una tabla.

Periodos

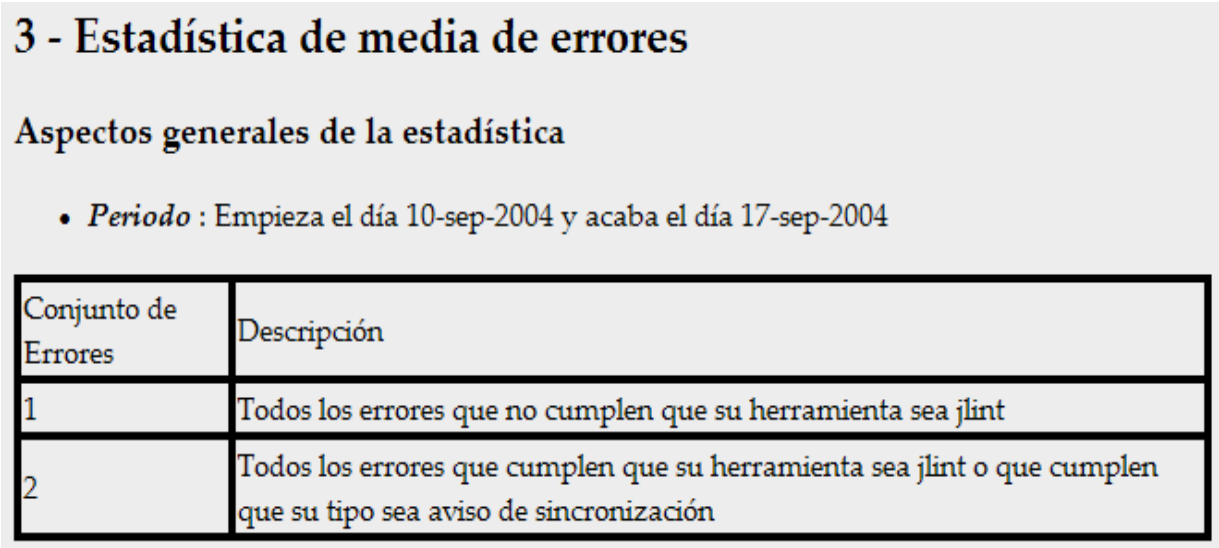
P1	P2	P3	P4	P5
Empieza en la compilación 0 y acaba en la compilación 200	Empieza en la compilación 200 y acaba en la compilación 400	Empieza en la compilación 400 y acaba en la compilación 600	Empieza en la compilación 600 y acaba en la compilación 800	Empieza en la compilación 800 y acaba en la compilación 1000

Tabla de Resultados para todos los usuarios

	P1	P2	P3	P4	P5
Conjunto de Errores 1	7,70	7,00	14,00	7,00	0,00
Conjunto de Errores 2	14,35	14,50	21,00	14,50	8,00

Figura 39. Tabla con la descripción de los periodos y tabla con los resultados de forma numérica

La siguiente estadística es también una estadística de media de errores; pero en este caso, no hay evolución, por lo tanto sólo se estudian los avisos para un periodo y este se indica al comienzo, en la información general de la estadística. Además, aquí se describen los conjuntos de errores que intervienen en la estadística (Figura 40).



3 - Estadística de media de errores

Aspectos generales de la estadística

- ***Periodo*** : Empieza el día 10-sep-2004 y acaba el día 17-sep-2004

Conjunto de Errores	Descripción
1	Todos los errores que no cumplen que su herramienta sea jlint
2	Todos los errores que cumplen que su herramienta sea jlint o que cumplen que su tipo sea aviso de sincronización

Figura 40. Comienzo de la tercera estadística

Al tratarse de una gráfica de media de errores en un periodo, la gráfica que se genera es una gráfica de barras (Figura 41). En la que aparece una barra para cada conjunto de errores y además se duplican para poder comparar la información para un usuario individual con la del conjunto de todos los usuarios.

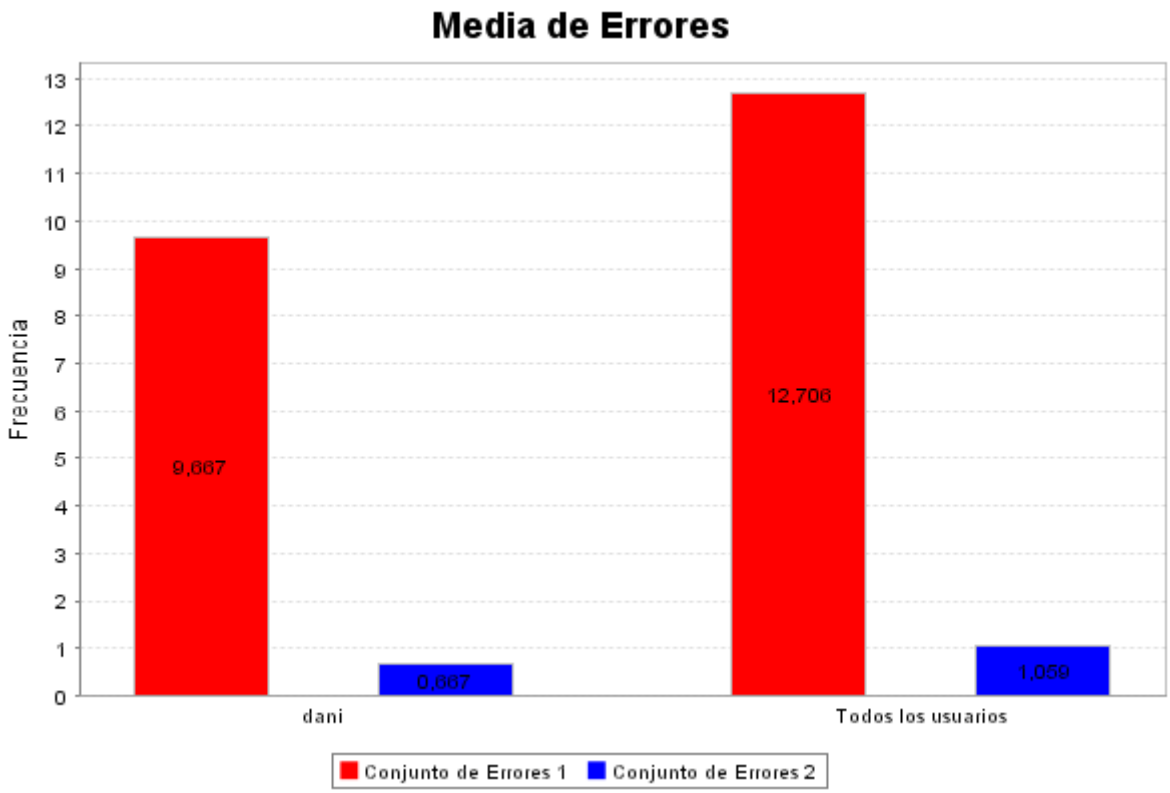


Figura 41. Gráfica de barras para una media de errores

Por último se presentan los resultados de forma numérica en una tabla (Figura 42).

	Conjunto de Errores 1	*Conjunto de Errores 2*
dani	9,667	0,667
Todos los usuarios	12,706	1,059

Figura 42. Resultados numéricos de la media de errores (tercera estadística)

La siguiente estadística, es una estadística de errores frecuentes, en este caso no hay ningún problema a la hora de realizar las consultas. Por lo que, además de la información general, se generarán las gráficas y los resultados numéricos.

Al principio de la estadística, se presentan los datos generales sobre la estadística como el periodo, el conjunto de errores que intervienen en la estadística y el usuario afectado (Figura 43).

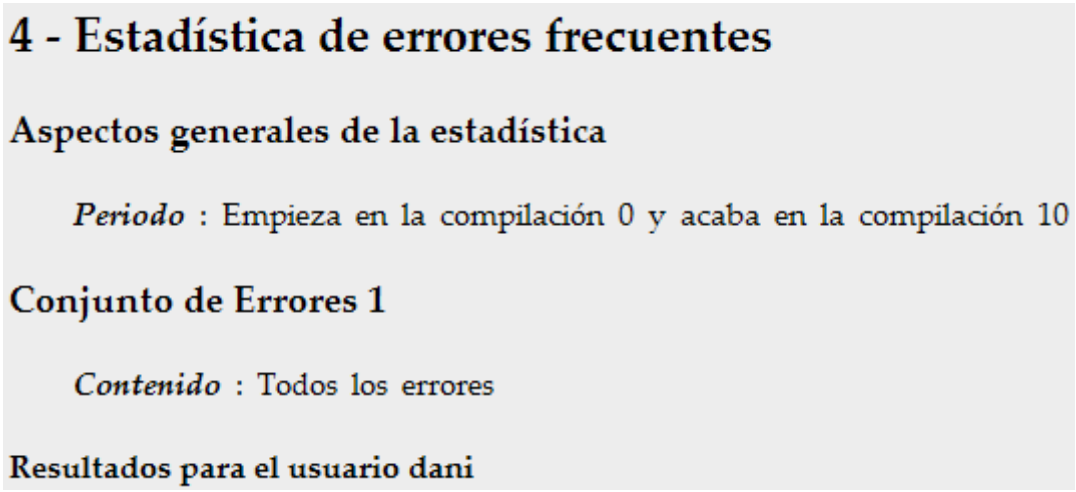


Figura 43. Descripción de los aspectos generales de la cuarta estadística

Para mostrar gráficamente los resultados se utiliza una gráfica de barras. Los errores frecuentes están ordenados desde el más frecuente hasta el menos frecuente. En la leyenda se puede ver a que errores corresponde cada barra (Figura 44).

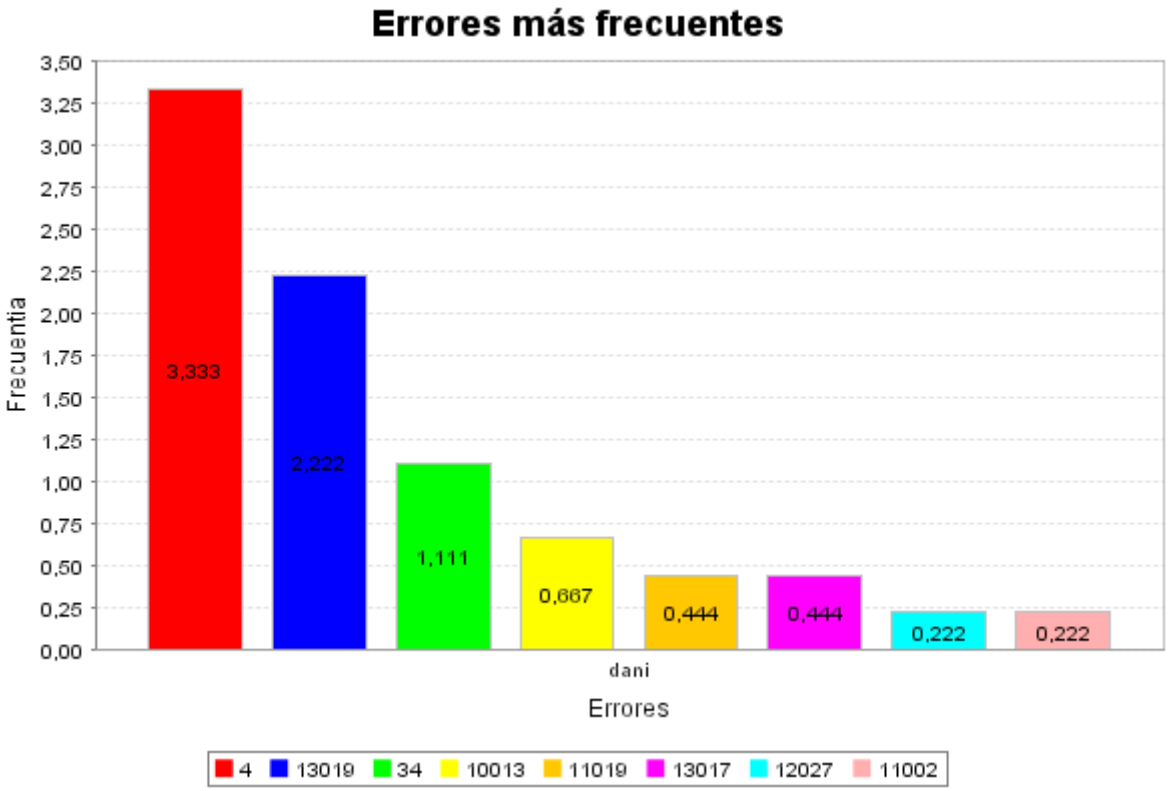


Figura 44. Gráfica de barras de errores frecuentes (cuarta estadística)

Después de la gráfica, se presenta una tabla con los resultados de forma numérica (Figura 45). Además del valor número, se puede ver que el código del error tiene un hiperenlace a la página Web de la base de conocimientos de errores (Capítulo 9. Entorno de desarrollo integrado en Web y base de conocimientos colaborativa: IDEWeb) donde está la información semántica sobre el error, lo cual permite que se pueda consultar esta información con facilidad.

Código de Error	4	13019	34	10013	11019	13017	12027	11002
Frecuencia	3,333	2,222	1,111	0,667	0,444	0,444	0,222	0,222

Número de compilaciones : 9

Figura 45. Tabla de errores más frecuentes (cuarta estadística)

Por último, tenemos una estadística de errores frecuentes que presenta un problema (Figura 46). La estadística está configurada para obtener los cinco errores más frecuentes de un conjunto de errores, pero en el análisis del conjunto de errores sólo se encuentran dos errores (Si se comprueba el contenido del conjunto de errores 1, se verá que es lógico que sólo haya dos). Para avisar al usuario se pone un mensaje de aviso en la sección de información general de la estadística.

5 - Estadística de errores frecuentes

Aspectos generales de la estadística

Periodo : Empieza en la compilación 0 y acaba en la compilación 10

Conjunto de Errores 1

Contenido : Todos los errores que cumplen que su código sea 13019 o que cumplen que su código sea 13000

Resultados para el usuario dani

Aviso : La estadística está configurada para dar 5 de errores pero sólo hay resultados de 2 errores.

Figura 46. Estadística de errores frecuentes con menos errores de los necesarios

Como se puede ver en la Figura 47, la gráfica de errores frecuentes en vez de ser de cinco errores frecuentes pasa a ser de los dos errores más frecuentes.

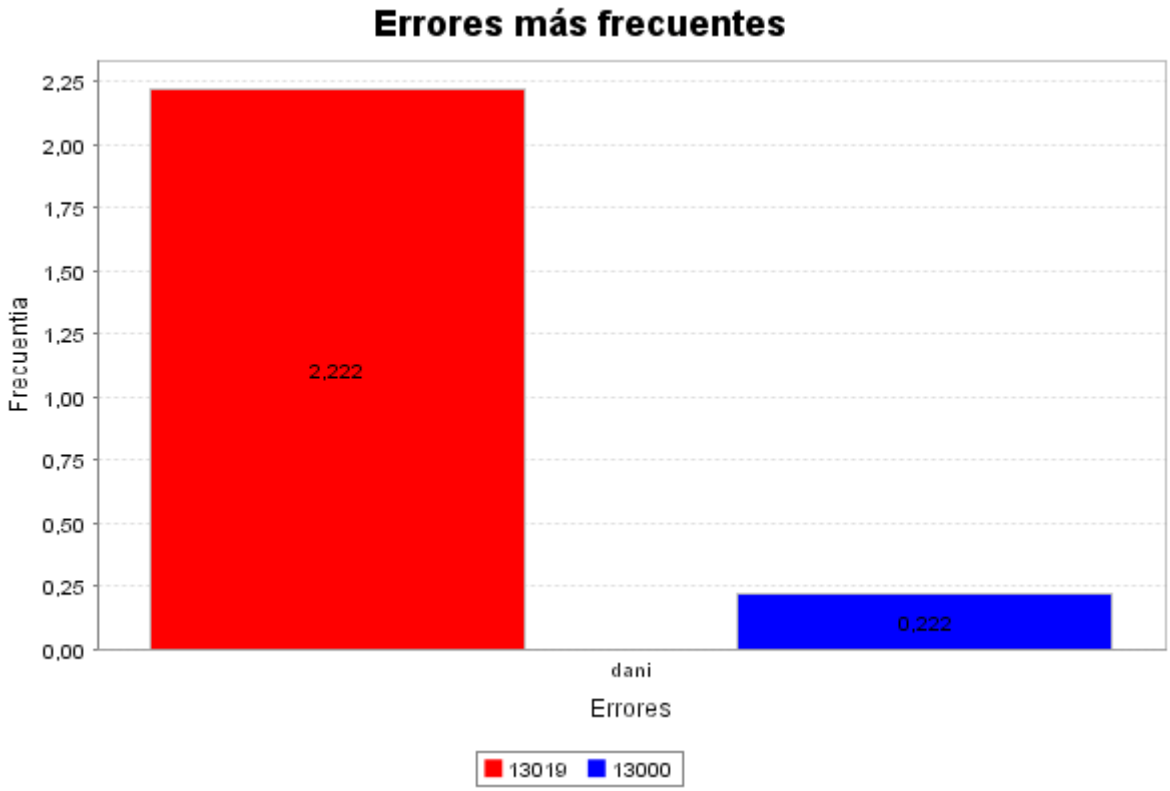


Figura 47. Gráfica de errores frecuentes con menos errores de lo normal

Por último, se pone la tabla con los resultados para los dos errores con su frecuencia y los enlaces a sus páginas Web (Figura 48).

Las estadísticas de errores frecuentes siempre ponen al final de sus resultados el número de análisis que se emplearon para realizar la estadística, así se puede tener una idea del uso del sistema en ese periodo.

Código de Error	13019	13000
Frecuencia	2,222	0,222

Número de compilaciones : 9

Figura 48. Tabla de resultados de la quinta estadística

Para finalizar el informe de resultados, se incluyen los logotipos del W3C: xhtml y CSS informando de que esta página cumple los estándares y permite su comprobación.

7.10 Evaluación del Sistema de análisis de programas

Para comprobar la viabilidad de la utilización de este subsistema con proyectos reales y en condiciones parecidas a las que tendría en un entorno real, se llevó a cabo un experimento [Pérez 2004c]. Para ello empleamos diez proyectos elegidos al azar, entre todos los disponibles de la asignatura de Metodología a la Programación de primer curso (segundo cuatrimestre) de la Escuela Universitaria Ingeniería Técnica de Informática de Oviedo[5].

[5] Página Web de la asignatura de Metodología de la Programación: http://www.euitio.uniovi.es/~mp

Para desarrollar los proyectos de esta asignatura se utilizó el entorno de desarrollo de Borland JBuilder 9.0.

Estos proyectos fueron analizados usando la unidad de compilación "completa", que empleaba todas las herramientas de detección de errores configuradas en el sistema, es decir Antic, Jlint, FindBugs y PMD. Se guardaron todos los avisos en la base de datos del proyecto y se comprobó que fuesen errores reales y que no constituyesen falsos positivos.

Tabla 7. Contabilización y medias de los avisos generados por cada una de las herramientas.

	Código error	**Práctica**										
		A1	**a2**	**a3**	**a4**	**a5**	**a6**	**a7**	**A8**	**a9**	**a10**	**media**
Antic	10012	0	0	0	0	1	0	0	0	1	1	0,3
	10013	2	0	0	0	0	0	0	1	0	0	0,3
	10016	0	0	0	0	1	0	0	0	0	0	0,1
	Total	2	0	0	0	2	0	0	1	1	1	0,7
FindBugs	11004	0	0	0	0	0	1	2	0	0	0	0,3
	11010	1	0	0	0	0	0	0	0	0	0	0,1
	11014	0	0	0	1	0	0	0	0	7	0	0,8
	11015	0	0	0	0	0	0	0	0	1	0	0,1
	11020	0	0	6	0	0	0	0	0	7	0	1,3
	11021	2	0	0	0	0	0	0	0	3	0	0,5
	11022	0	6	0	0	0	0	0	3	0	0	0,9
	11023	0	0	0	0	0	1	2	4	0	0	0,7
	Total	3	6	6	1	0	2	4	7	18	0	4,7
Jlint	12002	1	3	0	5	7	3	1	2	1	0	2,3
	12004	0	0	0	0	0	0	0	0	0	1	0,1
	12005	1	0	0	0	0	0	0	0	0	0	0,1
	12014	0	0	0	0	0	11	3	0	0	0	1,4
	Total	2	3	0	5	7	14	4	2	1	1	3,9
PMD	13036	0	0	0	0	0	0	0	0	0	1	0,1
	13038	0	0	2	0	0	0	0	0	0	0	0,2
	13039	0	3	0	8	17	6	15	9	1	3	5,9
	13040	1	1	3	0	2	0	0	4	6	0	1,7
	13042	0	0	0	0	1	1	0	0	0	0	0,2
	13043	0	0	0	1	2	0	1	0	1	0	0,5
	13044	4	3	6	0	14	0	2	6	2	0	3,7
	13046	2	0	0	0	0	0	0	0	3	0	0,5
	13061	0	0	1	0	0	0	0	0	0	0	0,1
	13071	0	0	1	0	0	10	0	0	0	0	1,1
	13072	0	1	0	0	0	1	0	0	0	0	0,2
	13073	2	1	1	0	0	0	1	0	2	0	0,7
	13080	2	2	2	2	5	2	2	2	6	3	2,8
	13083	0	0	0	0	0	0	0	0	1	0	0,1
	13084	2	2	1	2	2	7	0	4	2	1	2,3
	Total	13	13	17	13	43	27	21	25	24	8	20,1
	Totales	20	22	23	19	52	43	29	35	44	10	

Los resultados de este experimento pueden verse en la Tabla 7 donde se presentan el número y tipo de avisos generados por las herramientas, para cada uno de los proyectos, así como la media de cada uno de ellos. Los avisos están representados por su código en la base de datos del proyecto. Para poder conocer el significado de cada aviso se muestra una segunda tabla Tabla 8 donde se asocia a cada aviso una descripción breve sobre su significado.

Tabla 8. Correspondencia de cada código con el mensaje de aviso.

Código	Descripción
10012	May be wrong assumption about IF body
10013	May be wrong assumption about ELSE branch association
10016	Possible miss of BREAK before CASE/DEFAULT
11004	Comparison of String objects using == or !=
11010	Possible null pointer dereference
11014	Uninitialized read
11015	Useless control flow
11020	Unwritten field
11021	Unread field: should this field be static?
11022	Unused field
11023	Unread field
12002	Local variable '*name*' shadows component of class '*name*'.
12004	Method *name* can be invoked with NULL as *number* parameter and this parameter is used without check for NULL.
12005	Value of referenced variable '*name*' may be NULL.
12014	Compare strings as object references.
13036	All methods are static. Consider using Singleton instead.
13038	Avoid unnecessary if..then..else statements when returning a boolean
13039	Avoid unnecessary comparisons in boolean expressions
13040	Switch statements should have a default label
13042	Avoid reassigning parameters such as '*name*'
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.
13044	Avoid calls to overridable methods during construction
13046	This final field could be made static
13061	Avoid unused imports such as '*name*'
13071	Method name does not begin with a lower case character.
13072	Class names should begin with an uppercase character and not include underscores
13073	Abstract classes should be named 'AbstractXXX'
13080	The same String literal appears *number* times in this file; the first occurrence is on line *number*
13083	Avoid unused private fields such as '*name*'
13084	Avoid unused local variables such as '*name*'

A continuación se presentan los comentarios sobre los resultados obtenidos de cada una de las herramientas de detección de errores que se usaron en el análisis.

Antic: Se puede ver claramente que detecta pocos errores, la explicación de esta escasez de avisos es el uso de un entorno de desarrollo, en este caso JBuilder. Un IDE facilita que los archivos fuentes tengan una buena indentación. Como el análisis de Antic se centra en

errores provocados por una mala indentación, apenas se generan errores. Su ejecución en casi instantánea.

Jlint: Genera avisos importantes que claramente son síntomas de errores, como las comparaciones de String como una referencia o la ocultación de variables en una herencia. Su ejecución en casi instantánea.

FindBugs: Al igual que Jlint los errores generados son importantes, y casi siempre se complementan los que indican Jlint y FindBugs. Su ejecución es bastante lenta.

PMD: La herramienta que más avisos da es PMD. Entre los avisos hay algunos que no son demasiado importantes, pero que pueden llevar a confusiones, como los nombres de clases y métodos que no cumplen las normas habituales de nombramiento. Algunos de los avisos están destinados a indicar zonas de código donde sería una buena medida llevar a cabo una refactorización. La ejecución de esta herramienta es bastante lenta.

Las prácticas constaban de unas 1100 líneas de código repartidas en 14 archivos fuente java. En la ejecución de estas se tardó una media de aproximadamente 7 segundos.

7.11 Limitaciones del prototipo

Una de las líneas de mejora de este sistema es la creación de herramientas que faciliten el mantenimiento y la ampliación del sistema, para simplificar el trabajo del administrador del sistema. Entre esas mejoras se incluirían las herramientas que faciliten:

- La creación de nuevos scripts Ant para implementar nuevas Unidades de Compilación.
- La creación de nuevos archivos de configuración de estadísticas.
- La ampliación del archivo XML-Schema para crear nuevos tipos de estadísticas.
- El registro de nuevas herramientas en el sistema.

El sistema actual permite realizar todas estas operaciones; pero se realizan manualmente sobre los archivos de configuración, sin ningún apoyo de ninguna herramienta y requieren un cierto grado de preparación, si se quiere hacer que el sistema pueda ser administrado más fácilmente se deberá crear herramientas que simplifiquen estas tareas.

La parte de ejecución permite realizar la ejecución de clases para realizar pruebas de funcionamiento, estas pruebas podrían ser mucho más completas si integrásemos este módulo con una herramienta de pruebas como puede ser JUnit [JUnit 2002].

En el módulo de estadísticas sería necesario mejorar la usabilidad del informe de estadísticas y dejarlo en un formato que pueda ser procesado por otros servicios.

7.12 Conclusiones

En el sistema realizado se cumplen los objetivos planteados en la introducción y el análisis. Se ha logrado crear un sistema capaz de:

- Gestionar los errores cometidos por los usuarios a la hora de escribir código, los errores no sólo servirán como indicación de algo que hay que corregir; sino una oportunidad para aprender a hacer las cosas mejor para obtener un código fuente de mayor calidad.

- Detectar automáticamente estos errores al realizar cada compilación y también al ejecutar las clases en las pruebas, para ello se utiliza el compilador y otras herramientas de análisis estático y sus resultados son guardado en una base de datos.
- Crear una historia de compilación, compilación tras compilación se van acumulando los resultados de esos análisis con lo que podremos realizar un seguimiento de los errores de un usuario o de un grupo de usuarios trabajando en un proyecto.
- Analizar los datos del análisis actual y de la historia de compilación, guardados en la base de datos para generar estadísticas. Para proporcionar información al usuario no sólo se procesan los datos del último análisis sino que se recoge toda la historia de compilación con lo que podemos obtener no sólo frecuencia de errores sino también evolución de estos errores.
- Se presenta la información de forma que el usuario pueda obtener conocimiento. Se proporciona información numérica; pero también gráfica adaptada a cada usuario, cada usuario obtiene información sobre sus compilaciones; pero además puede configurarla para poder analizar ciertos datos que le pueden interesar a él y no a otro usuario.

Los ámbitos donde este proyecto tiene utilidad práctica son:

- El aprendizaje de la programación. En este contexto el papel de supervisor lo desempeñaría un profesor y el de desarrollador los estudiantes. Estos últimos disponen de un sistema de análisis avanzado capaz de detectar más errores que un simple compilador, proporcionándoles una información más precisa acerca de la corrección de su código. El sistema de ejecución capaz de capturar las excepciones, proporciona la posibilidad de realizar pruebas unitarias y almacenar los resultados. El módulo de generación de estadísticas hace que tanto desarrolladores como supervisores tengan todos los datos a su alcance de una forma sencilla y así poder utilizarlos para ir aprendiendo a realizar un código de mayor calidad y evitar errores los estudiantes y ver los puntos en donde el grupo necesita más apoyo, los profesores.
- El desarrollo de aplicaciones en general. El sistema es útil sobre todo desde el punto de vista de los desarrolladores. Se trata de obtener un código, no sólo que cumpla los requisitos funcionales, sino también, que sea fácilmente modificable y mantenible. Por tanto, podría ser usado por desarrolladores que tengan conocimientos avanzados de programación; pero quieran mejorar la calidad de su código fuente. Para ello, el sistema permite realizar un análisis más profundo y proporcionar las pautas de mejora.

El sistema de análisis es lo suficientemente flexible para que pueda ser ampliado sin necesidad de tener que hacer cambios en el modelo existente. El sistema está preparado para la ampliación de las herramientas registradas, ya sean compiladores o otras herramientas de detección de errores. También es sencillo crear nuevas Unidades de Compilación que amplíen la funcionalidad del sistema.

El sistema de generación de estadísticas al igual que el anterior puede ser ampliado fácilmente. Usando la tecnología XML-Schema el sistema puede ser ampliado y modificado sin que sea necesaria la modificación del núcleo del sistema.

Todas estas características hacen que este subsistema sea potente y flexible; pero para conseguir toda la funcionalidad de SICODE necesitamos una gestión de grupos y

colaboración sobre los ficheros del proyecto que proporcionará el subsistema COLLDEV que veremos a continuación en el Capítulo 8 y un entorno para la escritura de los programas y que proporcione una base de conocimientos como es IDEWeb que veremos en el Capítulo 9.

Capítulo 8. Sistema para la colaboración en el desarrollo de aplicaciones: COLLDEV

En este capítulo describimos el prototipo de Sistema para la Colaboración en el Desarrollo de Aplicaciones (COLLDEV, Collaborative Development). Este prototipo trata de desarrollar un Sistema Colaborativo que gestione los grupos de trabajo para el desarrollo de aplicaciones de forma asíncrona a través de la Web, permitiendo que se compartan los archivos fuente del proyecto en desarrollo, y que proporcione, a su vez, una "historia activa" del trabajo realizado hasta el momento; para permitir el seguimiento del proceso de desarrollo. Este es uno de los subsistemas que junto con PBA e IDEWeb forman el sistema SICODE.

8.1 Objetivos

COLLDEV [Pérez, 2004b] es un sistema colaborativo orientado al aprendizaje (Computer Supported Collaborative Learning, CSCL) vía Web, que facilita la coordinación y colaboración de grupos de trabajo para trabajar sobre un proyecto software y proporciona una *"historia activa"* del trabajo realizado por los miembros del grupo.

Los sistemas colaborativos tratan de encontrar un modelo que englobe los distintos tipos de participantes, las tareas a realizar y los modos de colaboración, siendo tema principal de gran número de investigaciones y artículos. Además presentan una gran funcionalidad y basan su arquitectura en cuatro principios básicos citados por Barros [Barros 2000]:

- Construcción conjunta de la solución a un problema siguiendo algún método de colaboración.
- Coordinación de los miembros del grupo para organizar el trabajo.
- Semiestructuración de los mecanismos que soportan la discusión argumentativa y la consecución de acuerdos entre los participantes.
- Interés tanto en el proceso de aprendizaje como en el resultado del trabajo en grupo, y por tanto, representación explícita de los procesos de producción e integración.

A partir de estos principios, se ha desarrollado un sistema CSCL, enfocado al desarrollo de proyectos software en grupo, que intenta solucionar las barreras con las que se encuentran los estudiantes de programación o desarrolladores en general, en este tipo de tareas. Es el Sistema para la Colaboración en el Desarrollo de Aplicaciones, cuyas siglas del inglés *Collaborative Development* son COLLDEV.

Con este sistema se ha tratado de resolver los inconvenientes que surgen a la hora de trabajar en grupo de forma presencial para desarrollar un proyecto software, como es la

distancia física que puede separar a los desarrolladores, y la incompatibilidad entre los horarios de trabajo.

8.1.1 Facilitar la comunicación y la colaboración en el desarrollo de software

El principal objetivo es permitir el desarrollo de software en colaboración por parte de un grupo de usuarios, siendo, por tanto, un objetivo muy importante a cumplir por este sistema, facilitar la comunicación y el trabajo colaborativo entre desarrolladores, y de esta forma permitir que los miembros del grupo colaboren en el objetivo común para hacer más eficiente el trabajo de todos. Este objetivo tiene relación con el primer y segundo principios de los sistemas CSCL citados anteriormente.

Para ello, se ha creado una aplicación que, de forma fácil, permitiese a los desarrolladores compartir los archivos fuente, permitiendo el trabajo en paralelo sobre los archivos del proyecto, a través de un espacio de trabajo compartido, y facilitar la comunicación, tanto entre ellos como con su supervisor, que será el encargado de dirigir el proyecto, mediante el envío de mensajes asíncronos, sin tener que recurrir a otros sistemas de comunicación más genéricos y menos eficientes. Esta comunicación es particularmente interesante en el planteamiento de tareas, la resolución de dudas y la revisión de código.

El objetivo del trabajo en paralelo sobre los archivos del proyecto consiste en que dos o más desarrolladores puedan modificar a la vez archivos pertenecientes al mismo proyecto, incluso el mismo archivo y que sea el propio sistema el que se encargue de juntar automáticamente y de forma asíncrona las modificaciones. Esto es viable y relativamente sencillo cuando los cambios no coinciden en la misma localización del archivo; cuando existe coincidencia habrá que habilitar un sistema de resolución de conflictos de código donde serán los desarrolladores los que manualmente hagan la mezcla de las modificaciones que entran en conflicto.

8.1.2 Seguimiento continuo del proceso de construcción de software

El segundo objetivo a cumplir por este sistema, y no menos importante, es permitir al supervisor un seguimiento continuo del proyecto, basado en la "historia activa", es decir, en las distintas versiones que se van generando de los archivos fuente del proyecto cuando los desarrolladores realizan las distintas tareas que se le plantean. La "historia activa" también debe facilitar la realización de comentarios sobre el trabajo realizado por los desarrolladores. Esto permite centrar el interés tanto en el proceso de desarrollo como en el resultado final del trabajo en grupo, cuarto principio de los sistemas CSCL.

Además de los comentarios realizados sobre los trabajos de los desarrolladores, los *supervisores* podrán responder, a las dudas planteadas por los desarrolladores e incluso colaborar en el desarrollo mediante el envío de algún documento o archivo.

8.1.3 Ayuda a la toma de decisiones en grupo

Un tercer objetivo perseguido en la realización de esta aplicación, es ayudar a los desarrolladores en la consecución de acuerdos, por lo que se pone a su disposición un sistema de encuestas que los propios desarrolladores irán creando cuando necesiten tomar alguna decisión, del tipo que sea, cumpliendo así el tercer principio de los sistemas CSCL.

8.2 Ámbito del prototipo y relación con el resto de prototipos

El sistema COLLDEV proporciona toda la funcionalidad relativa a la gestión de grupos de trabajo y almacenamiento compartido de los proyectos. Permite asignar uno o más proyectos a un grupo, refleja que miembros han trabajado sobre los ficheros de un proyecto y permite seguir la evolución de este proyecto a través de sus distintas versiones. Delega en IDEWeb para realizar la creación, edición y modificación de nuevos ficheros de un proyecto y para gestionar los errores y su corrección. Por otra parte, este sistema tampoco se encarga de la compilación y análisis de errores de los ficheros fuente del proyecto que realizará el subsistema PBA, sólo permite la gestión de las distintas versiones del proyecto.

8.3 Requisitos del prototipo

8.3.1 Gestión de roles, grupos y tareas

El prototipo distinguirá tres tipos usuarios.

- **Desarrolladores**, los cuales realizarán las tareas de desarrollo de forma individual o en grupos; y podrán editar, y por lo tanto, modificar, los archivos del proyecto almacenados en el repositorio de su grupo y revisar las versiones anteriores, así como comunicarse con los demás usuarios del sistema y leer los comentarios realizados por supervisores y otros desarrolladores en las revisiones del código fuente.
- **Supervisores**, su principal función (aunque no exclusiva, ya que pueden realizarla también desarrolladores) es revisar el código fuente realizado por sus desarrolladores, teniendo la posibilidad de realizar comentarios sobre el mismo, así como de responder a las dudas expuestas por los desarrolladores y proporcionar la ayuda que sea necesaria.
- **Administrador**, que realiza las tareas de un supervisor y además, hará las veces de administrador de la aplicación, aunque uno de los objetivos de esta aplicación es reducir las tareas que tiene que llevar a cabo este para no centralizar el trabajo en un usuario, siendo la mayoría de ellas automatizadas.

Usuarios

La forma de acceder a la aplicación será la misma para todos los tipos de usuarios, es decir, en cualquier caso, deberán introducir un nombre de usuario y una contraseña secreta que los identifique de forma única, para poder realizar esto es condición necesaria que el usuario se haya registrado previamente en el sistema, posteriormente el sistema muestra la información correspondiente al usuario que ha entrado en sesión.

El alta de un usuario desarrollador en el sistema se realiza de la siguiente forma: él mismo registra sus datos en el sistema, para lo cual deberá rellenar un formulario público con los datos necesarios (nombre, apellidos,…); esta acción es notificada al usuario administrador, que autoriza o deniega la incorporación del usuario al sistema.

Cada usuario supervisor podrá acceder a los archivos de sus desarrolladores para revisarlos y realizar los comentarios que sean oportunos, así como proveer a los desarrolladores de las especificaciones de las tareas a realizar, pudiendo aportar archivos, que serán expuestos en un espacio de trabajo compartido de los desarrolladores correspondientes. El usuario

administrador deberá llevar a cabo también las tareas propias de un administrador en lo que concierne a la gestión de usuarios (autorización de usuarios nuevos, creación de reglas para los grupos de trabajo, creación de grupos de trabajo, etc.).

Grupos de trabajo

Para poder trabajar en el sistema, un desarrollador deberá de elegir un grupo de trabajo al que incorporarse de entre los existentes en el sistema. Esta inscripción se verificará automáticamente mediante reglas. Una vez que el desarrollador se inscriba en un grupo se notificará al administrador, para que este tenga notificación de los usuarios inscritos en los grupos. La notificación sólo se produce cuando se ha incorporado a un grupo al cumplir las reglas preestablecidas, de otra forma el usuario es notificado en el momento que no puede darse de alta al incumplir una de las reglas.

El sistema permite la definición de reglas para la verificación automática de la correcta creación de nuevos grupos de trabajo. Estas son gestionadas por el administrador. Todo grupo existente en la aplicación deberá cumplir estas reglas. Ejemplos de estas reglas son: cada grupo de trabajo debe tener asignado algún supervisor, cada grupo de trabajo tendrá un número mínimo y máximo de desarrolladores inscritos a él, cada desarrollador sólo podrá aparecer inscrito en un grupo de trabajo.

Los grupos de trabajo serán creados a partir de las peticiones de los desarrolladores a la hora de formarlos. El sistema también permite la incorporación de un desarrollador a un grupo de trabajo o el cambio de un desarrollador de un grupo de trabajo a otro. Para realizar todas estas operaciones se sigue el mismo mecanismo: son los propios usuarios los que desencadenan la acción que se verifica automáticamente mediante las reglas de verificación definidas a priori y el administrador recibirá la notificación de la operación realizada.

Estos grupos poseerán un espacio de trabajo colaborativo (identificado por el nombre del grupo) en el cual los desarrolladores irán escribiendo y probando los archivos fuente en relación con las tareas que tienen asignadas; estos archivos quedarán disponibles para el resto del grupo, así como para el supervisor que tengan asignado, cuando se vayan finalizando las tareas se realizará un nuevo reparto de tareas en el grupo.

El proceso de inscripción de los supervisores a los grupos es el mismo que el de los desarrolladores, el supervisor hace la petición que tiene que cumplir las reglas establecidas y se notificará al administrador.

8.3.2 Espacio compartido de colaboración - espacio personal

Cada grupo tiene un espacio de trabajo colaborativo asociado. En este espacio se mantendrán los archivos correspondientes a los proyectos que el grupo está desarrollando. Se pueden crear nuevos proyectos en el espacio de trabajo colaborativo para que todo el grupo pueda trabajar sobre ellos. Cada vez que un usuario quiera trabajar sobre un archivo almacenado en el espacio compartido de su grupo, primero se realizará una copia del mismo en el espacio de trabajo personal, sobre esta copia se realizará el trabajo correspondiente y una vez finalizado, el archivo se enviará de nuevo al espacio de trabajo colaborativo combinando los cambios realizados.

Cada usuario tendrá a su disposición un espacio de trabajo personal, que será creado al entrar en el sistema como usuario registrado. A este espacio personal sólo tendrá acceso el propio usuario. Este espacio de trabajo será identificado por el nombre del usuario y se utilizará para el trabajo individual que desarrolle el usuario.

Tanto el espacio compartido de colaboración, como el espacio de trabajo personal estarán situados en el servidor, con lo que estarán siempre disponibles independientemente del lugar desde el que se acceda. Los usuarios podrán intercambiar archivos entre su espacio de trabajo personal (en el servidor) y el sistema local.

Cada desarrollador podrá editar los archivos existentes correspondientes a un determinado proyecto y guardar de nuevo los cambios realizados. Para poder realizar un seguimiento se guardan distintas versiones del archivo a medida que los usuarios lo van modificando.

8.3.3 Navegación a través de las versiones

La historia activa permite una navegación entre las diferentes versiones que se han ido generando de un trabajo, artículo, archivo, etc. a lo largo de su proceso de creación, permitiendo la introducción de comentarios y sugerencias en cualquiera de las versiones.

En nuestro caso, la historia activa hará referencia a las distintas versiones que los desarrolladores han ido generando de los archivos para la realización del proyecto, las cuales ayudarán a los supervisores a llevar un seguimiento de la evolución del trabajo de sus desarrolladores.

Se podrán realizar revisiones sobre los archivos, y todas sus versiones, existentes en su espacio de trabajo colaborativo del grupo al que pertenece y anexar comentarios a los mismos. Cada usuario podrá revisar: qué miembro del grupo hizo determinados cambios y cuáles fueron estos; todo esto en cada una de las versiones de un archivo; de esta forma, podrá ponerse en contacto con él para hacer comentarios o pedir aclaraciones sobre un fragmento de código.

8.3.4 Comunicación, coordinación y toma de decisiones conjunta por parte de los usuarios

En lo que se refiere a la comunicación asíncrona entre desarrolladores y supervisores o entre múltiples desarrolladores, se distinguen tres tipos de mensajes: "preguntas y respuestas", comentarios y tareas.

- Mensajes de tipo preguntas y respuestas. Este tipo de mensajes, como su propio nombre indica, pueden contener preguntas sobre cualquier tipo de duda, o su respuesta. Pueden ser enviados y recibidos tanto por supervisores como desarrolladores y su finalidad es establecer una comunicación para facilitar la resolución de las dudas que puedan surgir. Ante este tipo de mensaje, el receptor podrá enviar otro al emisor del mismo, como respuesta a sus dudas o planteando otra pregunta relacionada con la anterior.

- Mensajes de tipo comentarios. Estos mensajes contienen comentarios (sugerencias, errores, etc.) realizados, bien por supervisores, bien por desarrolladores, que surgen tras la revisión de algún archivo de otro desarrollador. Además, deberán contener también, el nombre del archivo al que hacen referencia. En el caso de los desarrolladores, estos sólo podrán realizar comentarios sobre los archivos que se encuentran en el espacio de trabajo colaborativo de su grupo, ya que al resto no tiene acceso. Este tipo de mensajes también puede ser enviados de forma automática por el sistema, indicando el registro de un nuevo usuario, la creación de un nuevo grupo, etc. Ante este tipo de mensaje el receptor puede responder con mensajes, matizando los comentarios, o dar solución al error o problema notificado.

- Mensajes de tipo tarea. En estos mensajes unos usuarios indican a otros las tareas que han de realizar, y en el caso de los supervisores podrán ir acompañados de archivos que han depositado en el espacio de trabajo de sus desarrolladores. Ante estos mensajes el receptor podría responder con la aceptación o el rechazo de una determinada tarea.

Todos los mensajes deberán ser accesibles cuando el usuario receptor de los mismos esté trabajando con la aplicación, visualizándose cuando son recibidos en un espacio de la aplicación dedicado especialmente para ello y donde se ubicarán los mensajes más recientes. De esta forma la transmisión de mensajes entre usuarios es casi instantanea. También se podrá acceder al resto de los mensajes recibidos con anterioridad a petición del usuario. Los mensajes leídos deberán ser distinguidos de aquellos que el usuario aun no ha leído o marcado como tal.

Otra herramienta del sistema COLLDEV pensada para la coordinación es la realización de votaciones o encuestas que ayudará a los desarrolladores a ponerse de acuerdo en determinadas decisiones. Estas siempre se llevarán a cabo entre los miembros integrantes de un grupo. Cualquier miembro que desee someter a consulta una decisión puede crear una encuesta, proponiendo una pregunta y una serie abierta de alternativas. Cuando se crea una encuesta, el resto de desarrolladores del grupo recibirán un mensaje de tipo Tarea. Todos los miembros del grupo pueden votar en la encuesta, en caso de no estar de acuerdo con las alternativas propuestas en la encuesta, puede aportar su propia alternativa, a la cual le dará su voto. Una encuesta será evaluada cuando todos los miembros del grupo hayan votado. Se pueden consultar los resultados obtenidos tras las encuestas, ya que quedarán almacenados.

8.4 Descripción de Actores y casos de uso

En este apartado describimos cada uno de los actores del sistema, además de la definición, se explica, las operaciones que puede realizar cada uno. Y los casos de uso dando una breve descripción de lo que comprenden.

8.4.1 Actores

Supervisor. Usuario encargado del reparto de tareas en el grupo y la revisión del trabajo realizado por los desarrolladores. Puede enviar cualquier tipo de mensajes.

Administrador. Encargado de realizar las tareas del administrador de un sistema. Recibe la notificación del registro de nuevos usuarios. Es el único que puede modificar cualquier tipo dato de cualquier usuario. Es el encargado de crear las reglas a cumplir por los grupos de trabajo. Recibe la notificación de la creación de nuevos grupos. Recibe la notificación de la incorporación de usuarios a los grupos. Indica a los desarrolladores si han de cambiarse a otro grupo, si se elimina el actual.

Desarrollador. Usuario que interacciona con la aplicación para realizar el desarrollo de la aplicación colaborando con otros. Puede realizar cualquier operación sobre sus archivos. Recibe y envía mensajes de preguntas y respuestas e, incluso de tareas. Recibe mensajes de comentarios. Tiene acceso a los archivos de su grupo. No tendrá acceso a la información (cuentas y archivos) de otros usuarios del sistema.

8.4.2 Casos de uso

Gestión de usuarios y grupos. Es una función exclusiva del administrador. Autorización de registro de un usuario. Modificación y consulta de usuarios. Creación de reglas para la creación de grupos, creación y eliminación de grupos de trabajo.

Administración de proyectos y espacios de trabajo. Acceso a los archivos de los proyectos de todos los grupos, configuración de los repositorios de versiones.

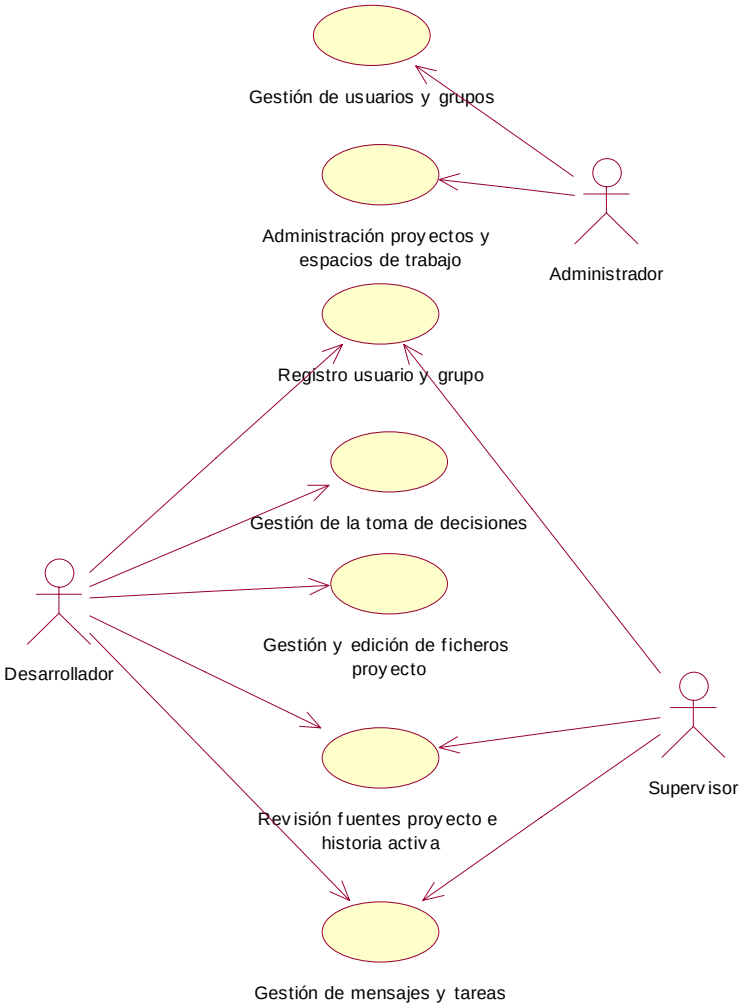


Figura 49. Diagrama de casos de uso del prototipo de colaboración

Registro de usuario y grupo. Tanto desarrolladores como supervisores pueden acceder a este caso de uso. Registro pro-activo (por parte del propio usuario) de un usuario en la aplicación. Incorporación de usuarios a los mismos siguiendo un modelo pro-activo.

Gestión y edición de archivos del proyecto. Creación, edición y borrado de archivos de un proyecto. Gestión del intercambio de archivos entre los dos espacios y con el exterior, generación de versiones para permitir un seguimiento mediante la historia activa.

Revisión de fuentes del proyecto e historia activa. Visualización, revisión y realización de comentarios sobre los archivos fuente. Acceso a versiones anteriores de los archivos permitiendo seguir el proceso de construcción del proyecto.

Gestión de mensajes y tareas. Envío y recepción de mensajes entre usuarios. Generación automática de mensajes asociados a determinadas tareas.

Gestión de toma de decisiones. Creación y gestión de encuestas para ayudar a la toma de decisiones.

8.5 Diseño

8.5.1 Diseño del espacio de trabajo compartido basado en un sistema concurrente de versiones (CVS)

Uno de los puntos clave del sistema, como se ha dicho en el análisis de requisitos, es el establecimiento de un espacio compartido para facilitar la colaboración de los desarrolladores, en la escritura de código, en la revisión, en la corrección de errores o en la mejora del diseño.

Para obtener un espacio de trabajo colaborativo entre un grupo de desarrolladores se ha utilizado un sistema concurrente de versiones (CVS) [Cederqvist 1993], que, como su propio nombre indica, es un sistema de control de versiones automático para facilitar el trabajo en grupos desarrollo.

CVS centra su misión en llevar a cabo el control de versiones de los archivos de un proyecto, que es realizado por un grupo de desarrolladores, facilitando el control de cambios realizados en el mismo por los distintos componentes del grupo de desarrollo. No tiene ninguna faceta de comunicación con lo que no aporta características para el envío de mensajes o la comunicación directa entre usuarios.

Una de las características más destacadas que aporta CVS es que no bloquea los archivos que están siendo modificados. El usuario que desea acceder a un archivo debe realizar una copia local del archivo original, que se encuentra en el repositorio, que es "la copia maestra en la que CVS guarda el historial de revisiones al completo efectuadas en un proyecto". Sobre esta copia local el desarrollador realizará las modificaciones oportunas, sin entorpecer el trabajo de los demás, ya que cualquier otro desarrollador, de forma simultánea, también puede hacer una copia del archivo original y ponerse a modificarlo. Una vez hechos los cambios, esta copia, será enviada de nuevo al repositorio, dando lugar a una nueva versión del archivo, que CVS se encargará de combinar con la copia maestra de forma automática.

Al hacer esta actualización de la copia maestra pueden suceder tres casos distintos:

1. Que la actualización se realice sin ningún tipo de problemas, porque la copia maestra no haya cambiado.
2. Que mientras un desarrollador ha estado realizando modificaciones otro haya enviado sus cambios al repositorio. Y los cambios hayan sido realizados en fragmentos distintos del archivo, con lo que CVS será capaz de combinar ambos cambios de forma automática.
3. Que mientras un desarrollador ha estado realizando modificaciones otro haya enviado sus cambios al repositorio. Y que estos cambios hayan sido realizados en el mismo fragmento del archivo por los dos desarrolladores. CVS detectará un conflicto, que deberán resolver los desarrolladores manualmente.

Antes de comenzar a trabajar con la copia local, es importante actualizarla, por si se han producido modificaciones en el repositorio que nuestra copia actual aun no recogía. De

este modo, los usuarios podrán realizar su trabajo reduciendo el riesgo de aparición de conflictos.

Uno de los inconvenientes de CVS es que no es posible crear archivo en el repositorio a partir de la nada. Para crear un nuevo archivo en el repositorio, este deberá ser importado de algún directorio local.

CVS permite realizar todas estas operaciones mediante instrucciones para ejecutar en línea de comandos. Las instrucciones más utilizadas son:

- **Update.** Realiza una actualización de la copia de trabajo a partir del repositorio. Es la primera operación que debe realizar un desarrollador a la hora de ponerse a modificar un archivo del proyecto; para actualizar la versión que tiene del mismo con los cambios realizados por el resto de los desarrolladores.
- **Dic.** Compara las copias de trabajo con sus homónimos en el repositorio. Y muestra gráficamente los cambios entre un archivo y otro.
- **Commit**: Envía las modificaciones al repositorio. Cuando el desarrollador ha finalizado los cambios y ha comprobado que todo funciona correctamente, realiza esta operación para que el resto del grupo pueda acceder a sus cambios.
- **Import**: Importa un nuevo proyecto al repositorio. Hay que asegurarse de que ese directorio sólo está lo que realmente que realmente queremos copiar.
- **Checkout**: Obtiene una copia de trabajo del archivo o proyecto indicado, para poder comenzar a trabajar.

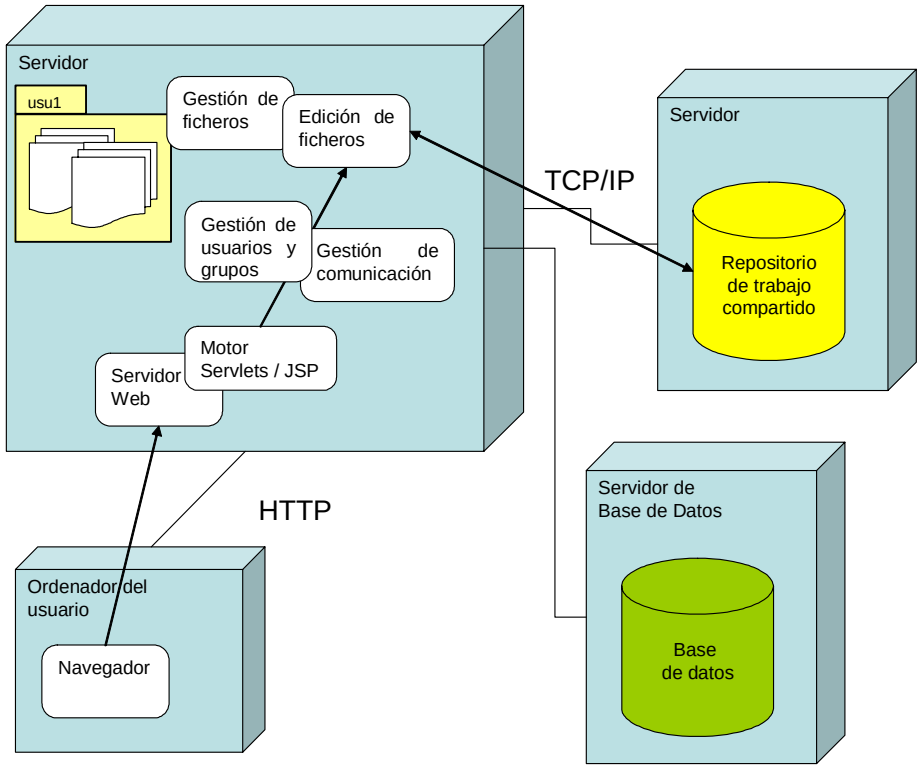


Figura 50. Arquitectura del sistema

Estos comandos tienen diferentes opciones que permiten flexibilizar su funcionalidad.

Para gestionar el espacio de trabajo colaborativo de COLLDEV utilizamos CVS, ya que ofrece un control de versiones automático, y esto nos facilitará la creación de la historia activa necesaria para realizar un seguimiento del progreso de los desarrolladores dentro un grupo de trabajo. Sobre este CVS de base añadimos una funcionalidad extra que automatice algunas de las tareas que tendrían que hacer los desarrolladores a mano; además también de forma automática agrupamos las versiones de diferentes ficheros de forma automática para conseguir versiones del proyecto completo para poder examinar todos los ficheros en conjunto.

8.5.2 Diseño de la arquitectura de la aplicación

COLLDEV es una aplicación Web basada en una arquitectura en la que toda la carga de la aplicación recae sobre el servidor, y el ordenador cliente únicamente debe ejecutar un navegador estándar que hará de interfaz con el desarrollador (ver Figura 50).

El espacio de trabajo compartido se encuentra en un servidor que gestiona el acceso concurrente a los archivos, al cual se accede mediante un cliente CVS, y en él se almacena el conjunto de archivos de los proyectos correspondientes a cada grupo de trabajo al que sólo pueden acceder los desarrolladores que lo componen y los supervisores que lo supervisan.

El espacio de trabajo personal contiene un directorio, personal y restringido, por cada usuario registrado en el sistema, por lo que cada usuario sólo tendrá acceso a su directorio, y nunca al del resto se sus compañeros.

Por otro lado, en la base de datos se encuentra toda la información relativa a usuarios, grupos y mensajes, que la aplicación necesita para funcionar correctamente.

8.5.3 Diseño de la interfaz

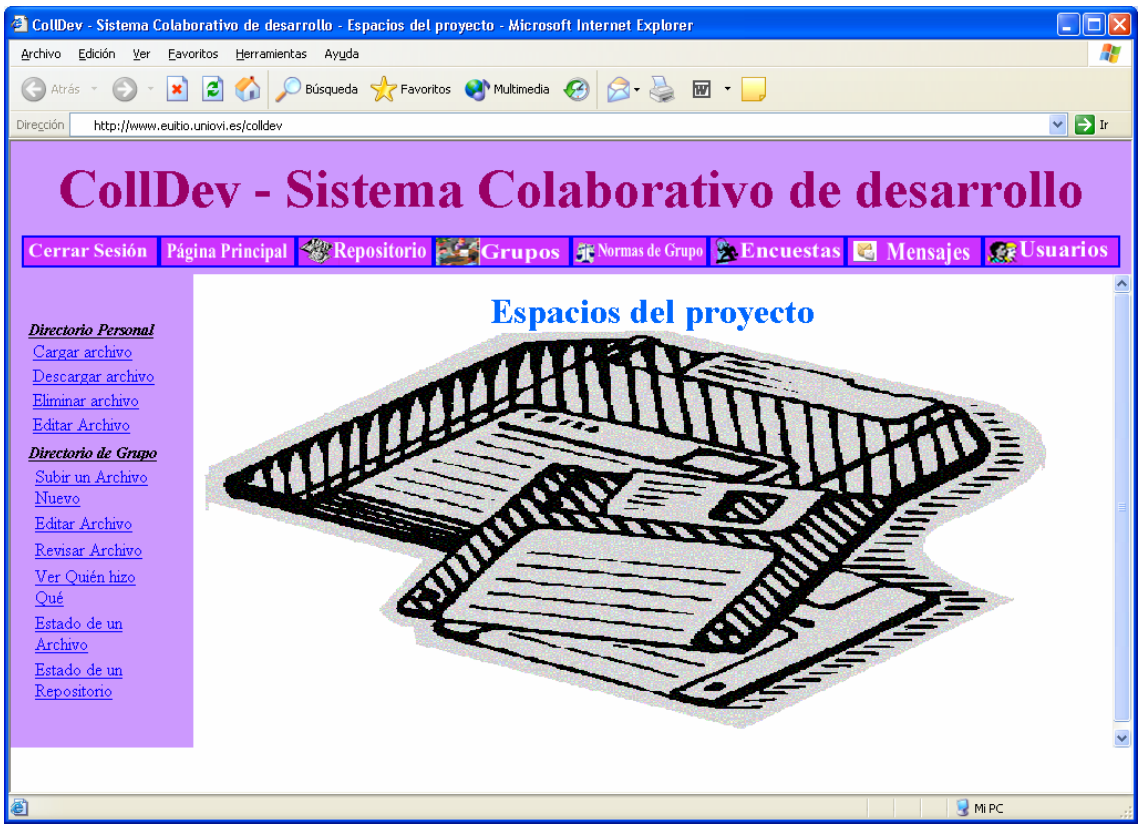


Figura 51. Página de espacios del proyecto del sistema colaborativo de desarrollo

En la Figura 51 se puede ver la página que permite acceder a la funcionalidad de espacios del proyecto, tanto el personal como el compartido. Todas las páginas de la aplicación siguen la misma estructura, analizaremos esta estructura general de la interfaz del sistema sobre esta página.

Los elementos que componen una página en el sistema COLLDEV son:

- Cabecera con el nombre del sistema, identifica claramente la aplicación.
- Menú horizontal (Figura 52), siempre está visible. Permite acceder en todo momento a toda la funcionalidad del sistema.
- Menú de la opción, menú vertical en el que se recoge todas las funciones de la opción elegida, en este caso "Espacios del proyecto".
- Área principal de interacción, se encuentra a la derecha del menú de la opción y es el área más amplia de la página, permite presentar al usuario información relacionada con la opción que ha seleccionado; también permite mostrar los formularios para recoger datos adicionales.
- Área de avisos del sistema, la parte inferior se reserva a avisos del sistema al usuario, informándole de operaciones erróneas o cualquier otra información útil en la interacción.

Figura 52. Barra horizontal con el menú principal de la aplicación

8.5.4 Modelado de datos

La base de datos es utilizada para recoger información de forma persistente sobre los distintos usuarios, grupos, mensajes y encuestas que tienen lugar dentro del sistema, así como de las relaciones creadas entre ellos. Además, la base de datos permite compartir información con los otros dos subsistemas: PBA e IDEWeb, con los cuales se formará el sistema SICODE que, entre otras cosas, facilita la realización de proyectos software en grupo.

Una parte común a todos los subsistemas es la entidad Usuario, la cual representa a todos los usuarios, reales o potenciales, del sistema y permite autenticar a los usuarios que quieren iniciar una sesión en el sistema. La entidad Estadistica, encargada de almacenar datos para la obtención de estadísticas de errores de los usuarios, es el punto de unión con el subsistema PBA (Capítulo 7) que se encarga de esta funcionalidad.

Usuarios y grupos

Entidad Usuario. Almacena toda la información que el sistema necesita en relación a los usuarios, en general. Permite utilizar estos datos en el proceso de autenticación.

Entidades Desarrollador, Supervisor, Administrador. Son distintos tipos de usuario. Almacenan la información asociada a cada uno de estos tipos.

Entidad Grupo. Almacena información relevante a cerca de los grupos de trabajo formados por los desarrolladores y autorizados por supervisores.

Entidad NormaGrupo. Almacena información a cerca de las normas que han de cumplir los grupos de trabajo en su composición. Esta información se utiliza para que el sistema verifique automáticamente la formación de grupos.

Relación Pertenece Grupo. Representa la relación existente entre un desarrollador y un grupo de trabajo. Un grupo podrá tener varios desarrolladores, pero un desarrollador únicamente podrá pertenecer a un grupo.

Relación Tutoriza Grupo. Representa la relación existente entre un grupo y un supervisor. Un supervisor podrá tutorizar muchos grupos, y un grupo podrá ser autorizado por varios supervisores.

Relación CumpleNormas. Representa la relación existente entre un grupo y la norma que cumple. Un grupo podrá cumplir muchas normas, y una norma podrá ser cumplida por varios grupos.

Relación Crea. Representa la relación existente entre una norma y el administrador que la crea. Debido a que en el sistema sólo existe un único administrador, esta relación carece de representación en forma de tabla, tratándose únicamente de una característica funcional.

Mensajes

Entidad Mensaje. Almacena información relativa a los mensajes enviados entre los diferentes usuario, incluido por el sistema.

Relación Envia. Representa la relación de envío de mensajes existente entre un usuario y el mensaje que envía. Un usuario podrá enviar muchos mensajes, pero un mensaje únicamente podrá ser enviado por un usuario.

Relación Recibe. Representa la relación de recepción de mensajes existente entre un usuario y el mensaje que recibe. Un usuario podrá recibir muchos mensajes, y un mensaje podrá ser recibido por varios usuarios.

Encuestas

Entidad Encuesta. Almacena datos de las encuestas que los desarrolladores realizan para poder llegar a acuerdos.

Entidad Idea. Almacena datos de las sugerencias que han ido surgiendo en las encuestas. Tiene una relación de dependencia por existencia con la entidad encuesta.

Relación CreaEncuesta. Representa la relación existente entre un grupo y la encuesta creada por uno de sus miembros. Un grupo podrá crear muchas encuestas, pero una encuesta sólo podrá ser creada por un grupo.

Relación Voto. Representa la relación existente entre un desarrollador y la encuesta en la cual ha participado con su voto. Un desarrollador podrá votar en muchas encuestas, y una encuesta podrá ser votada por muchos desarrolladores.

Relación Contiene Idea. Representa la relación existente entre una encuesta y sus ideas o sugerencias. Una encuesta podrá contener muchas ideas, pero una idea sólo podrá pertenecer a una encuesta.

Versiones de archivos y estadísticas

Entidad VersionArchivo. Almacena datos referentes a una versión de un archivo del espacio de trabajo colaborativo.

Relación CreaVersion. Representa la relación existente entre una versión de un archivo y el desarrollador que la creo. Un desarrollador podrá crear muchas versiones de un archivo, pero una versión sólo podrá ser creada por un desarrollador.

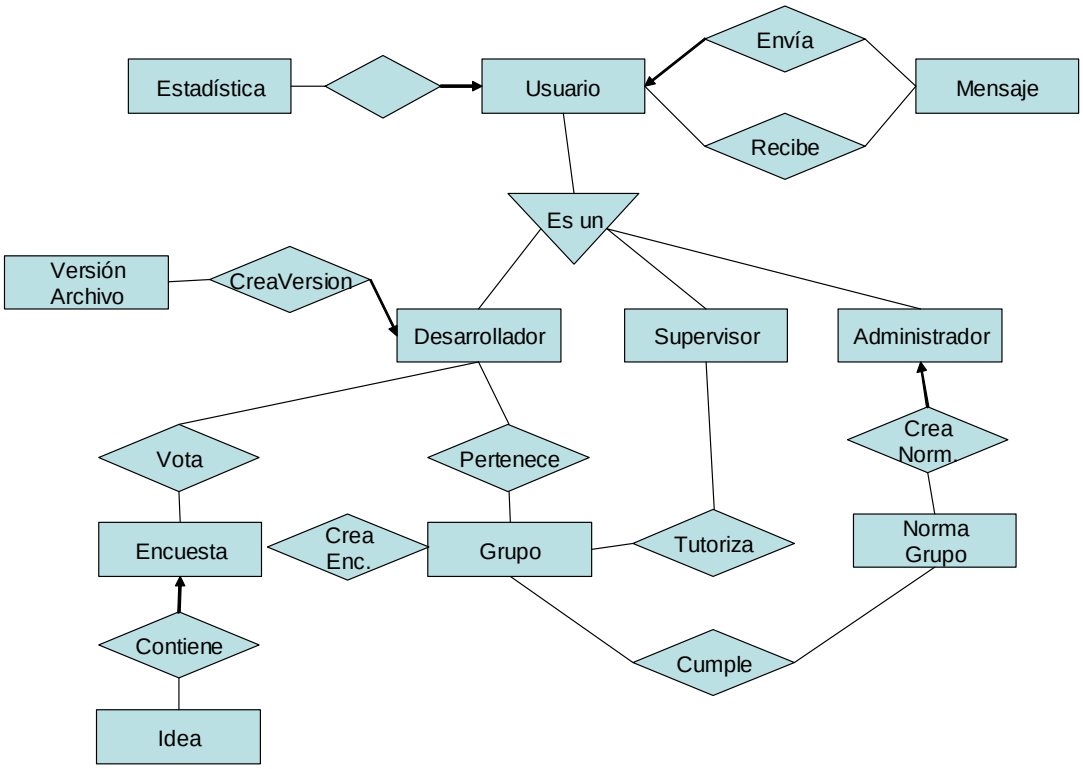


Figura 53. Diagrama entidad - relación del sistema

Entidad Estadística. Almacena datos que pueden ser de utilidad a la hora de hacer estadísticas sobre la actuación del usuario, sirviendo de enlace entre este proyecto y otros. Tiene una relación de dependencia por existencia con la entidad Usuario. En este diagrama (Figura 53) toda la información estadística está representada por una entidad; sin embargo, realmente hay un conjunto de entidades que permiten almacenar esta información; para una información más detallada ir al apartado correspondiente del subsistema PBA (apartado 7.8.5. Modelado de datos del Capítulo 7).

8.6 Limitaciones del prototipo

El prototipo implementa una historia activa de los proyectos en desarrollo permitiendo la revisión de cualquier versión de un archivo y compararlo con otra versiones para comprobar cuales han sido los cambios; sin embargo la navegación a través de la versiones del proyecto y se carece de una visión general de la evolución del proyecto, por lo que un supervisor tendrá que invertir bastante tiempo para tener una idea clara de la evolución de un proyecto.

Actualmente el sistema permite el intercambio de distintos tipos de mensajes entre los usuarios; sin embargo la construcción de los mensajes está poco automatiza, por ejemplo en los comentarios sobre un error debemos escribir el comentario y luego adjuntar el fichero que contiene el error; el sistema debería crear automáticamente el mensaje cuando el usuario marcara un error en área de edición del archivo fuente.

8.7 Conclusiones

En la introducción de este documento se señalaban tres objetivos clave. El resultado final, tras la realización del sistema COLLDEV, ha sido el siguiente:

- **Permitir la comunicación y el trabajo colaborativo entre desarrolladores y supervisores, para facilitar el desarrollo colaborativo de software.** El sistema permite la comunicación mediante mensajes especializados según su función. Permite compartir y colaborar en la escritura de los archivos del proyecto a través de un espacio de trabajo colaborativo basado en un sistema de control de versiones.
- **Permitir al supervisor un seguimiento continuo de sus desarrolladores basado en la "historia activa".** Gestión automática de las versiones de los archivos del proyecto en el espacio de trabajo colaborativo y navegación a través de esas versiones facilitando la revisión y el seguimiento del proceso de desarrollo de una aplicación.
- **Ayudar a los desarrolladores en la toma de decisiones.** Creación y gestión de encuestas que permiten recoger planteamientos y elegir uno de ellos por mayoría.

De esta manera, se consigue disminuir los obstáculos a los que se enfrentan los desarrolladores a la hora de realizar sus proyectos software en grupo permitiendo una obtención de aplicaciones con una mayor calidad de código.

En el siguiente capítulo veremos el prototipo para el último subsistema de los que componen el sistema SICODE.

Capítulo 9. Entorno de desarrollo integrado en Web y base de conocimientos colaborativa: IDEWeb

En este capítulo abordamos el subsistema de SICODE que constituye el entorno de desarrollo integrado en Web (IDEWeb). Junto con los subsistemas PBA y COLLDEV (analizados en los capítulos anteriores) constituyen el sistema completo SICODE. Este entorno permitirá la edición de código y permitirá la invocación al compilador y las herramientas de análisis estático (subsistema PBA). Por otra parte, permite trabajar con los mensajes de error de varias formas: muestra los errores y facilita su corrección mostrando el archivo fuente donde se localizó, muestra avisos en base a las estadísticas realizadas por PBA para prevenir la repetición de errores y sobre todo proporciona enlaces a la base de conocimientos colaborativa que permitirá obtener mayor información sobre el error para solucionarlo y aprender a evitarlo.

9.1 Planteamiento general

El prototipo IDEWeb (Integrated Development Environment on Web) trata de conseguir un entorno de desarrollo que funcione a través de Web. El objetivo es mejorar las destrezas de los desarrolladores en la comprensión y solución de los errores y el aprendizaje para evitarlos en futuras ocasiones, con ello se consigue la mejora de la calidad del código fuente.

Cuando un programador tiene poca experiencia en un lenguaje de programación comete errores que aunque simples, no todos son de obvia solución, ya que sin esa experiencia es difícil relacionar el mensaje de un error con su causa. Algunas veces, en estos casos, una persona con conocimientos le proporciona algún patrón para solucionar el error; o bien el propio programador invierte tiempo hasta que encuentra la causa. Ese mismo error puede sucederle a otros desarrolladores. La experiencia acumulada por una persona que ha programado mucho en un lenguaje actualmente está desaprovechada, solo sirve para esa misma persona. Si conseguimos que el entorno en el que trabaja pueda guardar esas experiencias y compartirlas con el resto de desarrolladores, se aprovecharán las experiencias individuales de cada uno de ellos, resultando una adquisición de experiencia acelerada para todos los usuarios y una mejora en la calidad del código más rápida que si cada programador trabajase por su cuenta.

El entorno debe permitir la edición de código, y proporcionar ayudas en la edición por ejemplo el resaltado de sintaxis

Hemos decidido realizar el entorno de desarrollo para que funcionase en la Web por varios motivos:

- Al funcionar sobre un navegador Web estándar, el usuario final puede ponerse a programar sin necesidad de instalar nada, simplemente debe arrancar su navegador preferido y conectarse a la URL de IDEWeb, con esto evitamos el paso previo de instalación y configuración del entorno que siempre requiere emplear un tiempo y a veces conlleva también complicaciones.
- Una arquitectura Web nos permite centralizar, de forma sencilla, la información que se recoge en la monitorización del proceso de desarrollo de los distintos usuarios.
- En el campo del aprendizaje, un entorno Web puede ser integrado de forma relativamente fácil y natural con un sistema de e-learning que contenga la parte contenidos sobre la programación.
- Por último, el hecho de que funcione a través de Web, posibilita que los distintos usuarios trabajen a su ritmo sin la necesidad de que haya presencia física coincidente en lugar y tiempo de desarrolladores y supervisores, proporcionando mucha más flexibilidad de horarios.

Consideramos la base de conocimientos como una pieza clave de este sistema. Muchos errores son difíciles de corregir por una mala comprensión del mensaje de error por parte del desarrollador. Además, muchas veces es más importante obtener una ayuda con ejemplos concretos y casos en los que se haya detectado y corregido el error que una definición teórica de en que consiste el error. En estos momentos, existen distintos sistemas para gestionar conocimiento que son muy populares; pero la característica que deberíamos establecer para este módulo es la posibilidad de incluir contenido de forma muy sencilla para los usuarios y de forma colaborativa, es decir que todos los usuarios puedan dejar información en la base de conocimientos de una forma muy sencilla y sin la necesidad de herramientas especializadas. Además, otro factor importante es que el propio sistema extraiga conocimiento de forma automática de las acciones de los usuarios.

9.2 Objetivos del prototipo

La creación de este prototipo tiene dos objetivos fundamentales:

- Desarrollo un **entorno de desarrollo integrado con una interfaz Web**,
    - El entorno debe ser lo más abierto y flexible posible para que pueda ser utilizado con distintos lenguajes y compiladores.
    - La instalación de elementos en la parte cliente debe ser mínima, idealmente, solo un navegador Web.
    - Debe proporcionar herramientas de desarrollo tales como un editor de código, diversos compiladores y un gestor de archivos y directorios del proyecto, para poder afrontar proyectos software de tamaño medio.
- El entorno debe disponer de una **base de conocimientos colaborativa**, esta base de conocimiento será creada a partir de la experiencia de todos los usuarios
    - Debe permitir a todos los usuarios consultar información semántica a cerca de los errores detectados en el programa.
    - Cualquier usuario debe poder introducir, de forma sencilla y rápida información sobre los errores detectados o solucionados.
    - Así mismo, el sistema debe enlazar automáticamente esta información con el sistema de compilación para facilitar la consulta del conocimiento

asociado a cada error, proporcionando ayudas a la programación y a la resolución de errores.

Entre los objetivos también incluimos:

- La **sencillez de uso**. Es importante que sea sencillo de utilizar, instalar y mantener.
- Que el entorno pueda ser ejecutado y sea accesible desde un **gran número de plataformas**.

9.3 Ámbito del prototipo y relación con el resto de prototipos

Este subsistema cumple la función de interfaz con el usuario y permite controlar todas las herramientas necesarias en el proceso de escritura y depuración de código desde una única pantalla. El trabajo con este subsistema siempre se realiza sobre los archivos en el espacio personal del usuario y delega en el sistema COLLDEV el intercambio entre este espacio personal y el espacio compartido colaborativo disponible para el grupo que trabaja sobre un proyecto. El módulo IDEWeb tiene una relación muy estrecha con el subsistema PBA, ya que desde IDEWeb se invoca la compilación y el análisis estático que realiza PBA y se recoge la información generada por este tras realizar el análisis de los datos almacenados en la base de datos.

9.4 Requisitos del prototipo

A continuación, se enumerar los requisitos de este prototipo.

9.4.1 Gestión de usuarios (identificación y autenticación)

Para abrir una nueva sesión en el entorno de desarrollo los usuarios deben de estar registrados. El entorno permitirá autenticar a los usuarios para poder acceder a los archivos del proyecto que están desarrollando, a los servicios que ofrece y personalizar el acceso a la base de conocimientos.

9.4.2 Gestión del desarrollo de proyectos

Cada proyecto tiene reservado un espacio de trabajo en el servidor. Una vez que el usuario abre una nueva sesión de trabajo, se le presenta su espacio de trabajo, con una lista de los archivos que le pertenecen, y opciones para la gestión de los mismos. Dentro del espacio del servidor, así mismo, se dispondrá de utilidades para intercambiar archivos entre este espacio en el servidor y los sistemas locales de usuario.

Al elegir la opción de edición de un archivo, se cargará el archivo seleccionado en el área de edición, en caso de seleccionar nuevo archivo se creará en blanco para su edición. El entorno dispondrá de utilidades de ayuda a la edición, como el resaltado de sintaxis dependiendo del lenguaje seleccionado.

El sistema permitirá compilar los archivos en el espacio del servidor y recoger los binarios resultantes.

9.4.3 Gestión de errores

Los errores de compilación se mostrarán al usuario inmediatamente; pero además, serán clasificados por tipos y almacenados en la base de datos de la aplicación junto al enlace a la

página correspondiente de la base de conocimientos que explica la solución al mismo. Estas páginas de la base de conocimientos son creadas y enlazadas automáticamente por el sistema, pero los propios usuarios añaden más información y las modifican para enriquecerlas con su experiencia.

9.4.4 Base de conocimientos colaborativa

Se pretende aprovechar los conocimientos adquiridos por todos los usuarios del sistema para crear una base de conocimientos acerca del lenguaje de programación.

Un mismo error puede tener varias soluciones, dependiendo de las causas que lo provoquen, y a veces resulta frustrante, para el alumno, comprobar que la "ayuda" del compilador no aporta nada para solucionar el problema.

Pretendemos que en esta base de conocimiento, cada usuario del sistema vaya incluyendo ejemplos reales que permitan resolver distintos problemas, ya que los ejemplos de código es la parte de la ayuda que aporta más luz a los programadores. Con este sistema la experiencia de los usuarios van quedando reflejada en esos archivos de ayuda, las cuales con el tiempo van formando una excelente "enciclopedia" de programación.

La base de conocimientos colaborativa dispone de páginas HTML en las que se recoge información sobre un error de compilación concreto, que cualquier usuario puede crear y modificar para puntualizar alguna parte del contenido, para corregir algún error en el contenido, añadir más datos, etc. Pretendemos que todos los usuarios colaboren entre si para crear páginas que recojan contenidos amplios, correctos y de calidad; más que cada usuario aporte su solución de forma independiente.

Todo este sistema estará conectado tanto con el sistema de captura de errores que permitirá acceder fácilmente a este sistema para profundizar en la información sobre el error capturado y también con el entorno de desarrollo en el cual los mensajes visualizados serán extraídos de esta base de conocimientos.

9.4.5 Arquitectura portable y multiplataforma

Aplicación que funciona según el modelo de aplicación Web. La función del cliente es la de proporcionar una interfaz al usuario, dejando toda la parte del proceso en el lado servidor. La comunicación entre ambos será a través del protocolo HTTP, usándose en el cliente cualquier navegador Web estándar.

El sistema está pensado para ser portable entre plataformas, por lo cual las rutas a archivos, llamadas a ejecutables, llamadas a bases de datos, etc., deben situarse en un archivo de configuración esto proporcionará mucha flexibilidad para adaptar al sistema a nuevas situaciones. Por la misma razón, la salida HTML que proporcione deberá ser estrictamente compatible con los estándares (W3C, World Wide Web Consortium) para que pueda visualizarse en cualquier navegador.

9.5 Actores y casos de uso

Administrador. Lleva el control de la aplicación, configura nuevos compiladores, gestiona usuarios, modifica índices de la base de conocimiento.

Desarrollador. Sólo tiene acceso a su espacio personal, en él puede importar archivos, crearlos, compilarlos, eliminarlos. Puede trabajar con la base de conocimientos colaborativa creando nuevas entradas o añadiendo y modificando contenidos a entradas ya existentes.

Supervisor. Puede realizar todas las operaciones del actor desarrollador y además, invocar

análisis sobre cualquier proyecto, de los que supervisa, bien finalizado o en desarrollo.

Gestión de usuarios. Creación, modificación y eliminación de usuarios.

Validación de usuario. Si el usuario es correcto y la clave de usuario es correcta, entonces se inicia la sesión del usuario y se le muestra el menú adecuado según el rol: desarrollador, supervisor, administrador.

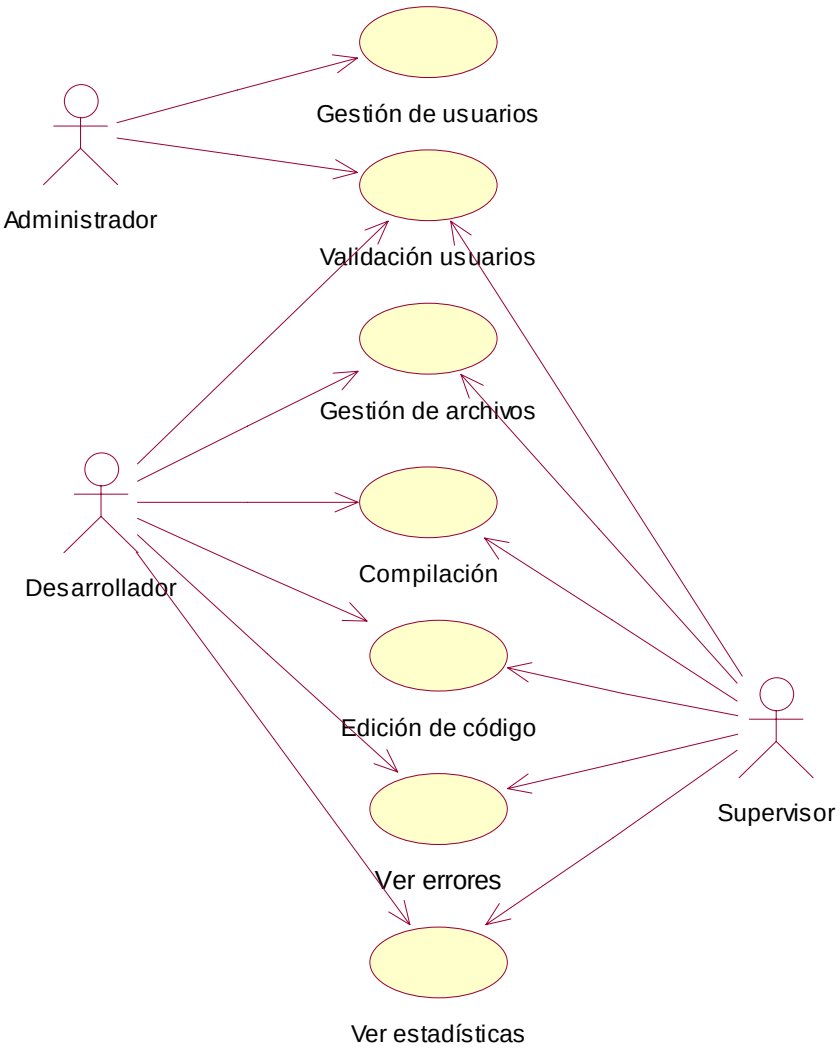


Figura 54. Diagrama de casos de uso para el prototipo de entorno integrado

Gestión de archivos. Nuevo archivo, existen dos formas de crear un nuevo archivo en espacio de trabajo: importándolo del ordenador cliente o editándolo en el sistema. En ambos casos se guardará en el directorio de trabajo del usuario, o en un subdirectorio. Intercambio de archivos entre servidor y sistema local, el usuario puede seleccionar un archivo existente en su espacio de trabajo, y este se transfiere a la máquina cliente y viceversa.

Edición de código. Para editar código IDEWeb proporcionará dos posibilidades, un área de texto HTML, y un editor Java con más opciones, que se desplegará en forma de applet.

Compilación. Compilar un archivo: Se selecciona un archivo del espacio de trabajo del usuario, un lenguaje y un compilador de entre todos los disponibles. La salida se mostrará en un área del entorno incluyendo todos los mensajes con errores de compilación y otros posibles errores.

Gestión de errores. El sistema localizará automáticamente cada error en el archivo fuente correspondiente. Además, por cada error se creará un enlace a la base de conocimientos según el compilador y lenguaje utilizados. El usuario podrá consultar la información asociada a cada error y revisar los enlaces relacionados. Si no existe la entrada en la base de conocimientos se creará automáticamente con la información básica.

Gestión de estadísticas. Muestra las estadísticas sobre frecuencia y evolución de errores dentro del entorno integrado para ayudar al usuario a no repetirlos. La elaboración de estas estadísticas la realiza el subsistema PBA (Capítulo 7).

9.6 Diseño

9.6.1 Diseño del subsistema de edición y compilación

Diagrama de clases

Para desarrollar este entorno como aplicación Web se ha utilizado el framework Struts®. En este framework la funcionalidad se descompone en "acciones" muy simples que están implementadas por clases que heredan de la clase "Action". Por otra parte, los formularios que permiten la invocación de las acciones desde la "vista" de la aplicación heredan de la clase "ActionForm".

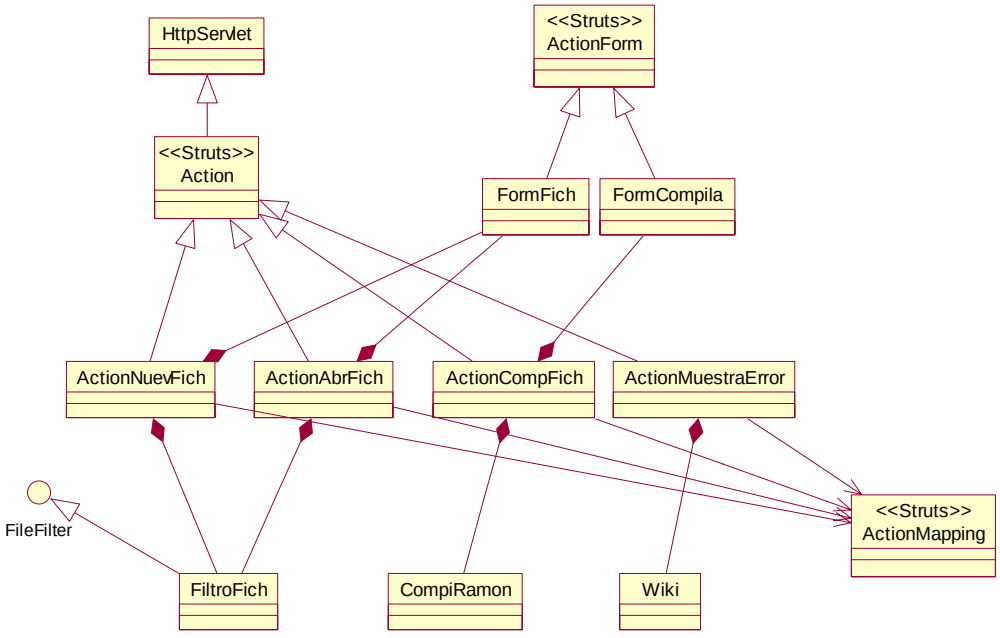


Figura 55. Vista parcial del diagrama de clases del sistema

Diagrama de secuencia de edición Web de un nuevo archivo

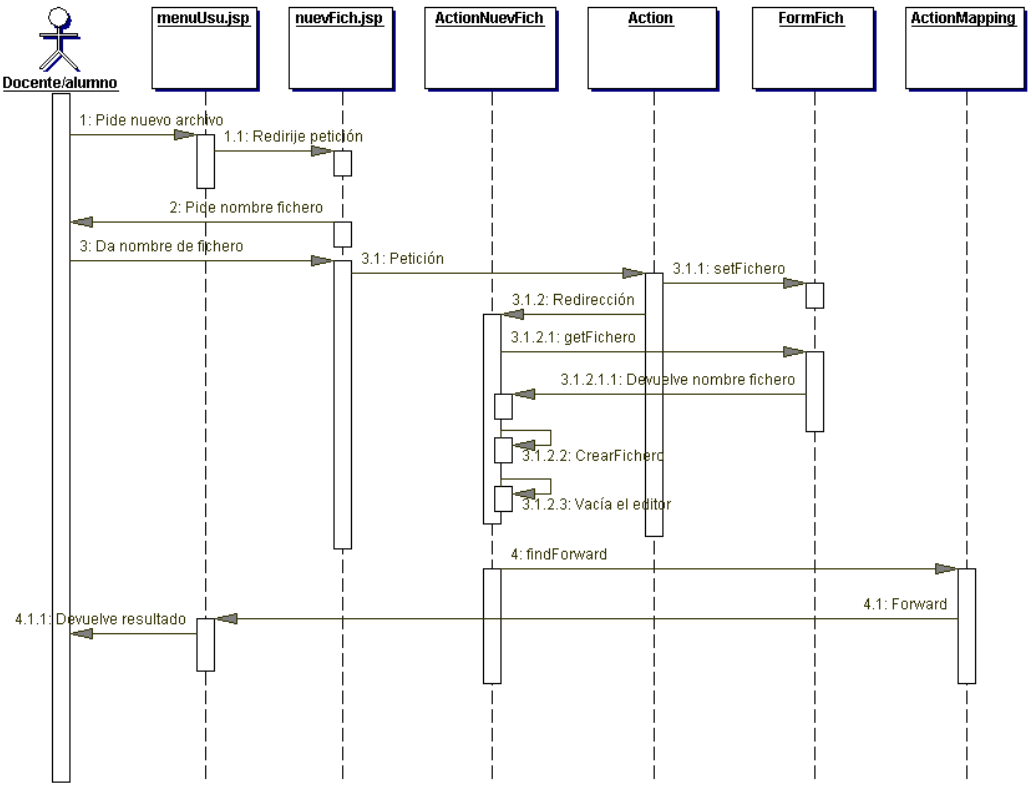


Figura 56. Diagrama de secuencia de Gestión de archivos, edición de un nuevo archivo

Diagrama de secuencia de la compilación de un archivo

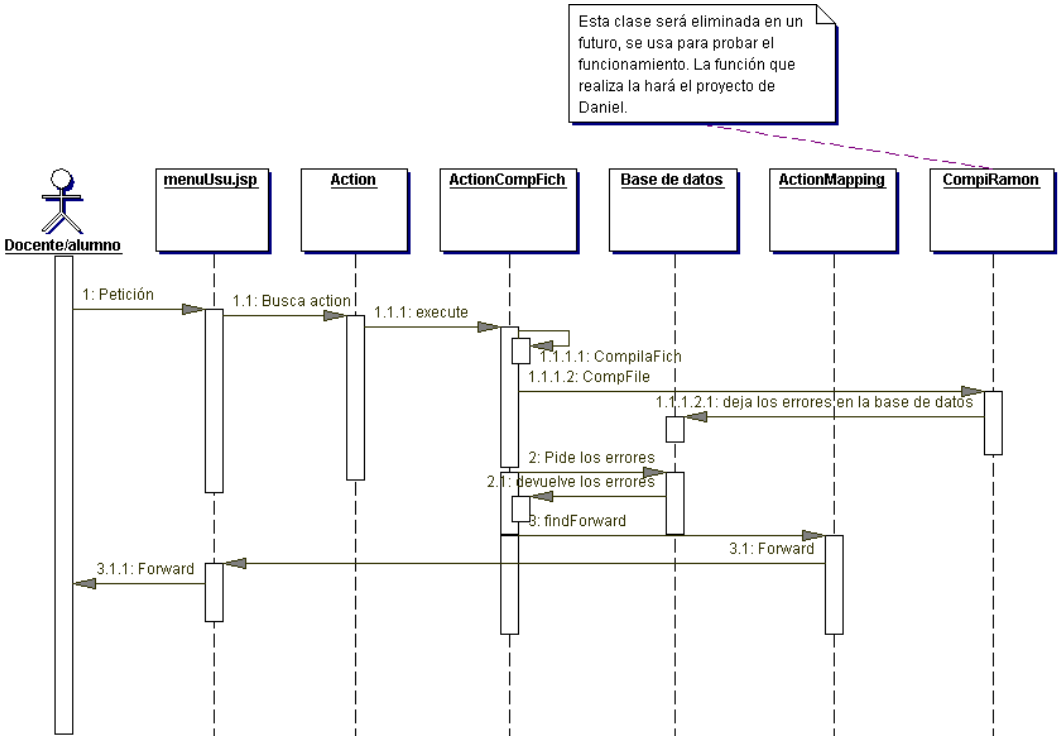


Figura 57. Diagrama de secuencia de Gestión de las compilaciones

Diagrama de secuencia de la creación de una nueva página asociada a un error

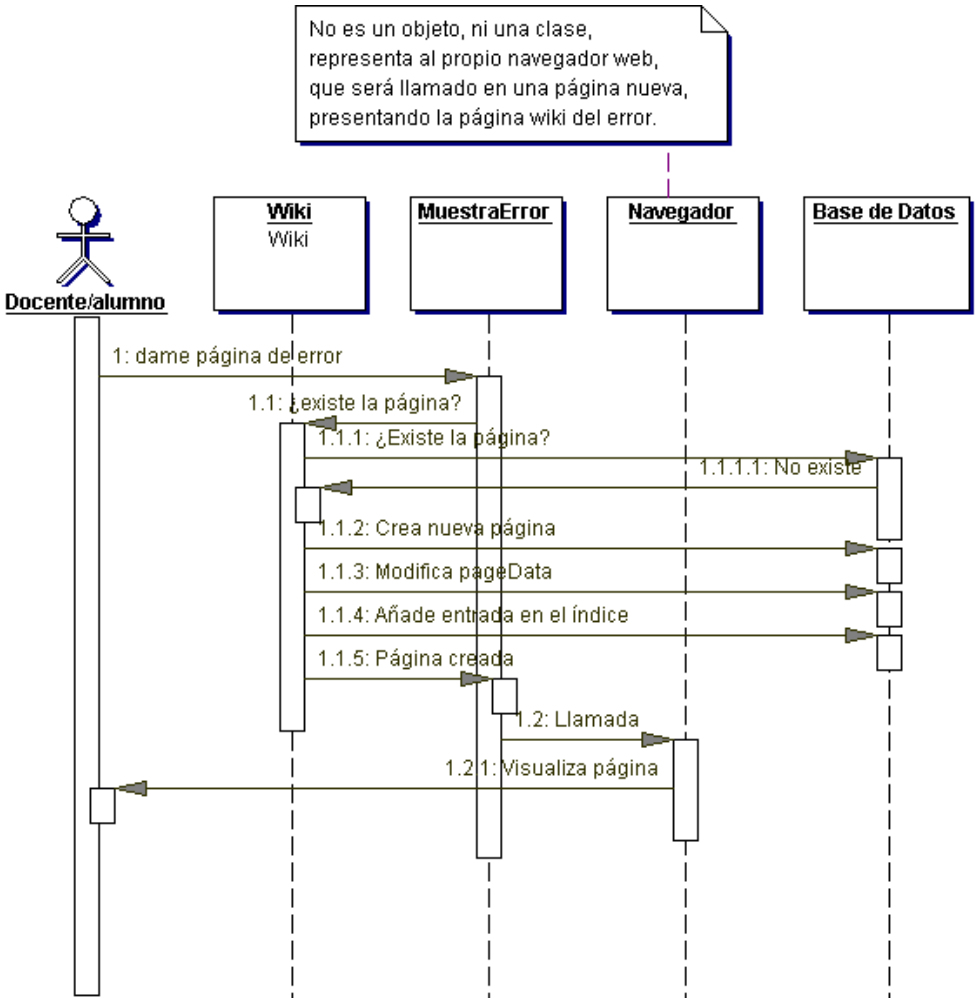


Figura 58. . Diagrama de secuencia de Gestión de errores, creando una nueva entrada de error en la base de conocimiento

9.6.2 Diseño subsistema de la base de conocimientos colaborativa

Una parte fundamental del sistema IDEWeb es la base de conocimientos en la que el usuario del sistema podrá obtener información y ayuda en el manejo del lenguaje, del compilador y de los errores de programación que vaya cometiendo. Se pretende que sean los propios usuarios quienes compartan sus conocimientos, ayuden a resolver problemas, informen de cómo consiguieron arreglar un determinado error para implementar esta base de conocimientos se han estudiado varias posibilidades y la que se ha elegido ha sido el sistema Wiki.

¿Qué es Wiki?

El término WikiWiki ("wiki wiki" significa "rápido" en la lengua hawaiana, "wee kee wee kee") se utiliza para referirse a un programa que permite tanto leer documentos hipertexto como publicarlos en la Web, este sistema es típicamente colaborativo ya que distintos autores describen un tema con ayuda de enlaces a otros temas relacionados creados por otros autores, además en una determinada página pueden intervenir varios autores a la vez. A las páginas publicadas por este sistema se las denomina Wiki Wiki Web, pero suele abreviarse como “wiki”.

Wiki es una herramienta colaborativa que permite a los usuarios publicar en Web sin tener conocimientos de HTML ni permisos de acceso a las carpetas que contienen los archivos de la página. El tiempo necesario para aprender a manejar esta tecnología es muy corto. Fue creado en 1.995 por Ward Cunningham quien creó uno de los primeros sitios con esta tecnología[6], y además ha escrito un libro al respecto [Leuf 2001]. Wiki no es un nuevo protocolo de Internet, las páginas generadas por un wiki deberían de seguir los estándares HTML. Para hacer un sitio wiki es necesaria la generación de páginas dinámicas, con cualquier tecnología que nos permita hacer esto y una base de datos que guarde dichas páginas.

Su funcionamiento es simple. Cada página tiene un botón de edición, al pulsarlo sale una nueva ventana con un área de edición de texto que nos permite modificar la página, escribiéndola en un lenguaje de marcado estándar para los wiki mucho más sencillo que HTML, con solo añadir enlaces a páginas que aún no existen, podemos crear nuevas páginas wiki. Wiki nos permite ver las diferencias entre versiones y realiza automáticamente un registro histórico de todos los cambios.

Cómo se utiliza un Wiki

El modo de empleo del wiki es muy sencillo, para consultar una entrada, no hay que hacer nada en especial, se consulta como cualquier otra página HTML, en cualquier navegador: introduciendo su URL o bien utilizando los enlaces hasta llegar a la entrada buscada. Para editar una página wiki sólo hace falta pulsar en el botón (o enlace) "editar" que hay en cada página, saldrá un área de edición de texto con el contenido de la página, en lenguaje wiki. Este lenguaje es simple y permite un marcado sencillo. Para crear una nueva página sólo hay que hacer un enlace a ella y cuando se pulsa sobre él, si no existía la página, se crea una nueva, si la página ya existía se edita.

Para darnos cuenta de la ventaja que supone esto sobre la edición convencional de página HTML, pensemos en como se edita una página Web típica:

- Se obtiene copia de las fuentes de la página a modificar, mediante ftp, o directamente al leerla en un navegador.
- Se edita el archivo HTML con todas sus etiquetas, para lo cual se requiere unos conocimientos sobre las etiquetas y la estructura del HTML. Normalmente se utilizan herramientas de edición especializadas para realizar esta tarea.
- Se sube este archivo al servidor que contiene la página Web, normalmente mediante el servicio ftp, necesitando para ello un acceso de lectura-escritura en el directorio.

Todos estos pasos se simplifican en Wiki a pulsar el botón de edición, escribir sobre el navegador y pulsar el botón de grabar. Además, debemos de pensar que wiki está pensado para que muchos usuarios colaboren, evita la obtención de las fuentes habría que asegurarse siempre que es la última versión, la sustitución de la página antigua por la nueva con el riesgo de sobrescribir cambios, el uso de ftp, y la necesidad de tener cuentas de usuario con permisos de lectura-escritura en el directorio para todos los usuarios que colaboren.

Casi todos los wikis tienen botones (o enlaces) diff e info. Diff indica los cambios realizados entre las distintas versiones de la página, e Info da información sobre la página,

[6] Wiki mantenido por Ward Cunningham, el creador del Wiki, con muchos enlaces a motores wiki, sintaxis… Fue el primer wiki de la historia y sigue siendo uno de los más visitados: http://c2.com/cgi/wiki

versión, última modificación, etc. Esto es muy importante ya que al ser unas páginas abiertas a la libre modificación conviene llevar un control de esos cambios realizados, para poder dejar la página en un estado previo si la modificación no fue conveniente, ver quien realizó las modificaciones y llevar estadísticas de su uso.

Funcionamiento interno de un wiki

Un wiki puede programarse en cualquier lenguaje, a través de CGI, aunque abundan los realizados en php y java, dada su mayor orientación a la red. Los más sofisticados usan bases de datos para almacenar las páginas e incluyen más opciones de sintaxis, posibilidad de añadir etiquetas HTML, si bien es importante la estandarización en la mayoría de las funciones. Un wiki tiene una página principal, generada automáticamente, a partir de la cual se construirá el resto del sitio, con el sistema de crear enlaces vacíos e irlos rellenando. El programa va generando las nuevas páginas y las guarda en archivos, o base de datos, guardando además información de los cambios, fechas, históricos, quien realizó la modificación, y demás datos necesarios.

Seguridad en wiki

En principio cualquier persona puede modificar las páginas de un Wiki, esto puede parecer un fallo de seguridad muy grande, pero en la práctica funciona; así lo demuestran proyectos como Wikipedia[7], una enciclopedia basada en este sistema. Hay que situar a wiki en su contexto, wiki nace como herramienta de colaboración y para ello debe estar abierta a todos. Se basa en el hecho de que si alguien se molesta en editar una página, lo va a hacer para aportar algo al wiki y no para destruirlo. Evidentemente, cualquiera puede hacerlo, no supone un reto para nadie, precisamente por ello los wikis no suelen ser objetivos de hackers. Otra cuestión es que se introduzcan datos incorrectos en el wiki ya sea de forma accidental o intencionada, eso puede ocurrir en un wiki o en cualquier página HTML convencional, y en un wiki tenemos la ventaja de que algún visitante de la página puede percatarse del error y corregirlo de forma inmediata.

Aplicación de wiki a la base de conocimientos colaborativa de IDEWeb

Una vez vistas las características de los sistemas wiki y su forma de trabajar, comprobamos que encaja perfectamente con los requisitos establecidos para la base de conocimientos colaborativa de IDEWeb, ya que:

- Facilita la consulta de la información ya que podemos acceder a ella mediante un hiperenlace, de la misma forma podemos acceder también a información relacionada.
- Permite la construcción y edición de páginas Web completas, con el formato deseado y con muy pocas restricciones, todo ello de una forma muy cómoda, sin necesidad de conocer HTML en profundidad.
- La única herramienta para realizar todo esto es un navegador Web por lo que evitamos la instalación de herramientas más sofisticadas.
- Está preparado para agilizar la colaboración de diferentes personas en la incorporación de nuevo contenido y refinamiento del existente.

[7] Wikipedia, es un proyecto que trata de construir una enciclopedia en múltiples idiomas con la colaboración de todo el mundo que quiera aportar algo, su dirección es: www.wikipedia.org.

La concepción del wiki que permite que todo el mundo introduzca contenido en el sistema es un cambio en la concepción respecto a muchos sistemas en los que únicamente un experto o un grupo de expertos reducido puede aportar conocimiento al sistema y todos los demás únicamente pueden consultar. En este nuevo planteamiento todos pueden aportar con lo que la base de conocimientos se enriquece más rápidamente, y aunque pueda haber ciertos errores en el contenido estos pueden ser reparados rápidamente.

Para integrar este sistema dentro de IDEWeb hemos trabajado fundamentalmente a nivel de la base de datos del wiki. A través, de la base de datos disponemos del índice del wiki y la URL de cada una de sus páginas, por lo que podemos crear enlaces dentro del entorno de desarrollo a la entrada del wiki correspondiente. Por ejemplo, en cada mensaje de error, cuando se visualizan en el entorno después de realizar una compilación, incluimos un enlace a la información correspondiente a ese error en el Wiki. Además, si no existe entrada para ese error en el Wiki el sistema crea una nueva entrada en la base de datos e introduce en la página la información básica; para que posteriormente los usuarios vayas añadiendo sus experiencias.

Como se vio en el apartado anterior, el problema de la seguridad puede solucionarse evitando que modifiquen páginas albergadas en el wiki. Los usuarios que no se hayan identificado y abierto una sesión en el sistema, con lo cual se sabrá siempre quien ha modificado una página.

9.6.3 Diseño de la arquitectura de la aplicación

La aplicación se divide en tres bloques claramente diferenciados (Figura 59):

- El entorno sigue la arquitectura de una aplicación Web modelo 2 clásica. Donde todo el procesamiento se lleva a cabo en el lado servidor. El núcleo principal del proyecto, está constituido por el entorno que integra una amplia funcionalidad: edición, compilación, gestión de archivos, gestión de la base de conocimientos, etc.
- Un applet que se ejecuta en la parte cliente y cuyo único objetivo es editar texto de la forma más cómoda posible y que permite el coloreado de sintaxis.
- Por último la base de conocimientos colaborativa, que almacena las páginas de ayuda, es totalmente independiente del entorno y simplemente necesitan una conexión TCP/IP para comunicarse, por tanto podrán estar en máquinas distintas para mejorar el rendimiento general del sistema.

Entorno: Se seguirá el patrón arquitectónico MVC (Modelo-Vista-Controlador). Este patrón promueve la separación de las funciones:

- Modelo: Lógica del negocio, son las operaciones propias de la aplicación, en nuestro caso compilación de archivos, gestión de usuarios, gestión de archivos, gestión de errores, manejo de páginas de ayuda. También se incluyen en este nivel las clases que permiten el acceso a la base de datos.
- Vista: Es el interfaz con el usuario, la entrada-salida de la aplicación, en IDEWeb son las páginas Web presentadas en el navegador y los formularios de entrada de datos.
- Controlador: Son las funciones de control, recogen los datos de entrada del usuario, se los envían al "Modelo", y redirigen la salida adecuada al usuario llamando a la "Vista".

Trabajando con servlets/JSP, el patrón MVC también se denomina Modelo2. El tamaño del presente proyecto es de la suficiente envergadura para que el Modelo2 comience a

mostrar todas sus ventajas. Se ha utilizado el framework Struts [Cavaness 2004] como base para implementar el sistema. Struts implementa el patrón MVC mediante un servlet como controlador y páginas JSP que conforman la vista. El modelo estará formado por diversas clases Java, algunas de ellas implementadas como Beans para permitir un acceso "limpio" desde los JSP.

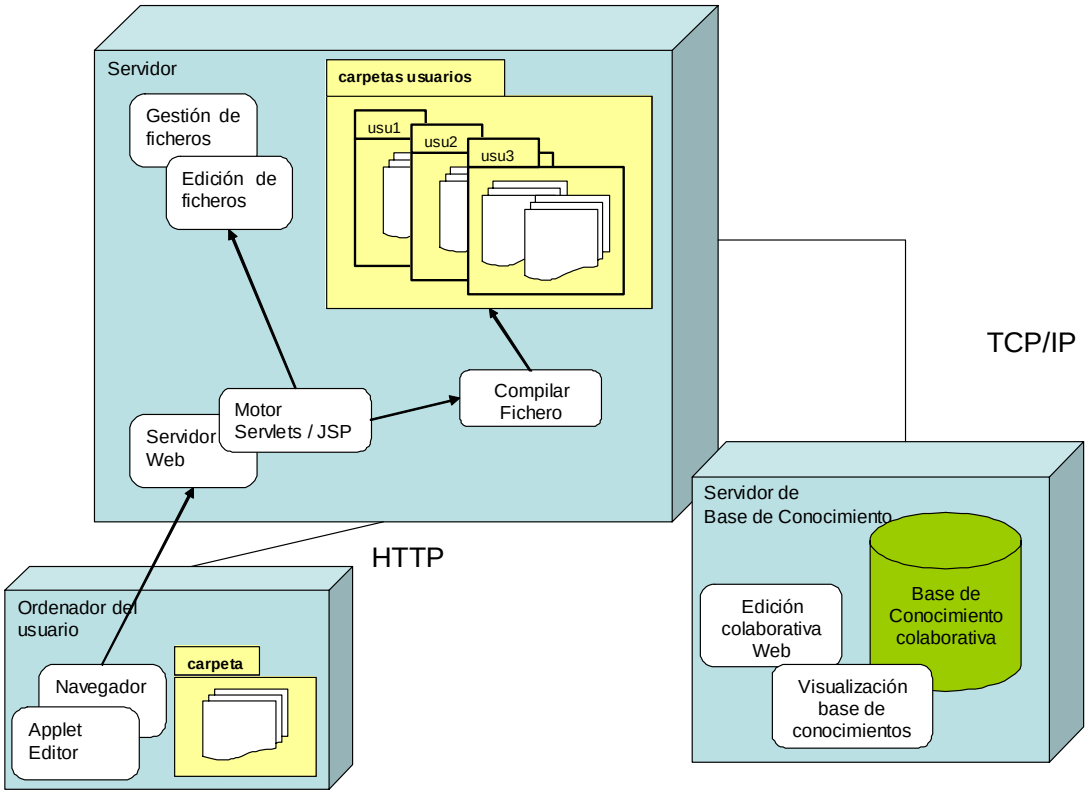


Figura 59. Arquitectura del sistema

Base de conocimientos colaborativa: Es independiente del entorno. Para su interconexión el entorno hace consultas sobre la base de datos que soporta a la base de conocimiento, de esta forma puede recuperar la dirección de cada una de las entradas de la base de conocimientos para insertar enlaces a estas entradas y crear nuevas páginas modificando directamente la base de datos, añadiendo un enlace a la nueva página en la página índice, y creando la nueva página, con una plantilla que servirá como base para que el usuario edite el error más fácilmente.

Editor: El editor es un programa Java implementado como un applet para que pueda ser ejecutado desde el navegador Web. Con la tecnología del plug-in de Sun podemos utilizar las últimas versiones del runtime de Java con lo que hemos decidido utilizar componentes Swing para la interfaz del editor. Este editor se comunica con el navegador para intercambiar el contenido del área de edición en el applet y en el navegador.

9.6.4 Diseño de la navegación

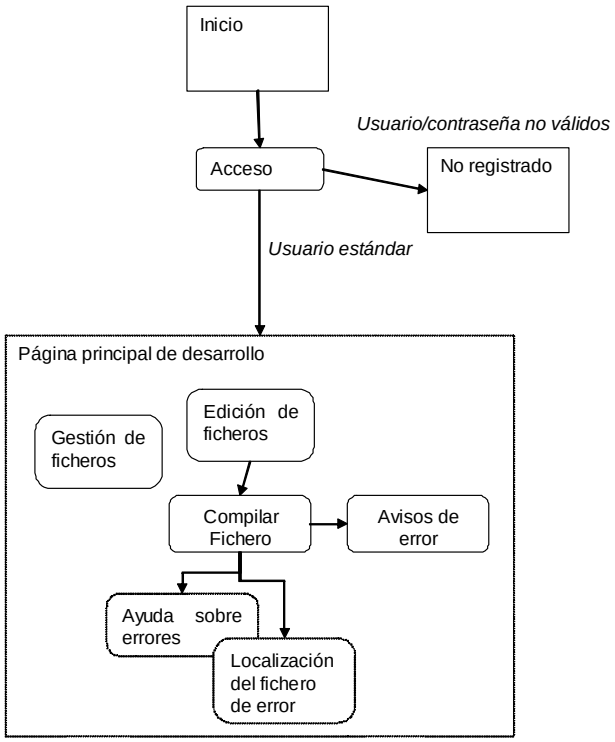


Figura 60. Esquema del modelo navegacional de la aplicación

9.6.5 Diseño de la interfaz

Primer prototipo de interfaz

Se ha realizado una aproximación iterativa para el diseño de la interfaz realizando varios prototipos sobre el papel y en HTML. En la primera aproximación al diseño considerado óptimo: una amplia zona de edición en la zona central, y los controles necesarios en los laterales. Se intentó minimizar el número de páginas necesarias, idealmente se manejarán todas las opciones principales desde una única página. Se busca una apariencia similar a un entorno de desarrollo tradicional. Por supuesto, el código HTML debe ajustarse a los estándares del W3C para evitar incompatibilidades entre navegadores. Se prima el espacio dedicado al área de texto que servirá para editar el código fuente, esto es importante ya que va a ser lo más utilizado y debe poder leerse con la mayor comodidad posible.

Se han utilizado técnicas de internacionalización para que el idioma de la interfaz pueda adaptarse a la configuración del navegador, con lo que el usuario dispondrá del idioma de su preferencia. IDEWeb estará disponible en inglés y castellano de forma inicial, añadir nuevos idiomas es extremadamente sencillo y no requiere cambios en el código de la aplicación ni su recompilación.

Las utilidades de archivo, en la primera aproximación, se sitúan en la parte derecha de la pantalla, en una columna. Los errores obtenidos en la compilación se situarían debajo de la ventana del texto, y se dispone de una barra de avisos donde van apareciendo mensajes según las operaciones que realice el usuario. En la parte superior se indica la fecha, y se deja espacio para enlaces que puedan ser interesantes.

Editor de código avanzado

Este es el elemento que más flexibilidad tiene a la hora del diseño, ya que al ser un applet, puede usarse toda la potencia de los componentes swing de Java en interfaces de usuario, y no estamos limitados al código HTML como en el resto de la aplicación. No obstante se opta por un diseño sencillo, teniendo en cuenta que aunque no estemos limitados por la potencia del lenguaje, si lo estamos por la velocidad de transmisión a través de la red, el editor se despliega en forma de applet, por lo que debe ser descargado de la máquina servidora a la máquina cliente a través de la red. Pese a que el enfoque de la aplicación es su uso en Intranets, es perfectamente posible su uso en Internet, que suelen ser bastante limitadas en cuanto a ancho de banda de comunicación.

Versión final de la interfaz

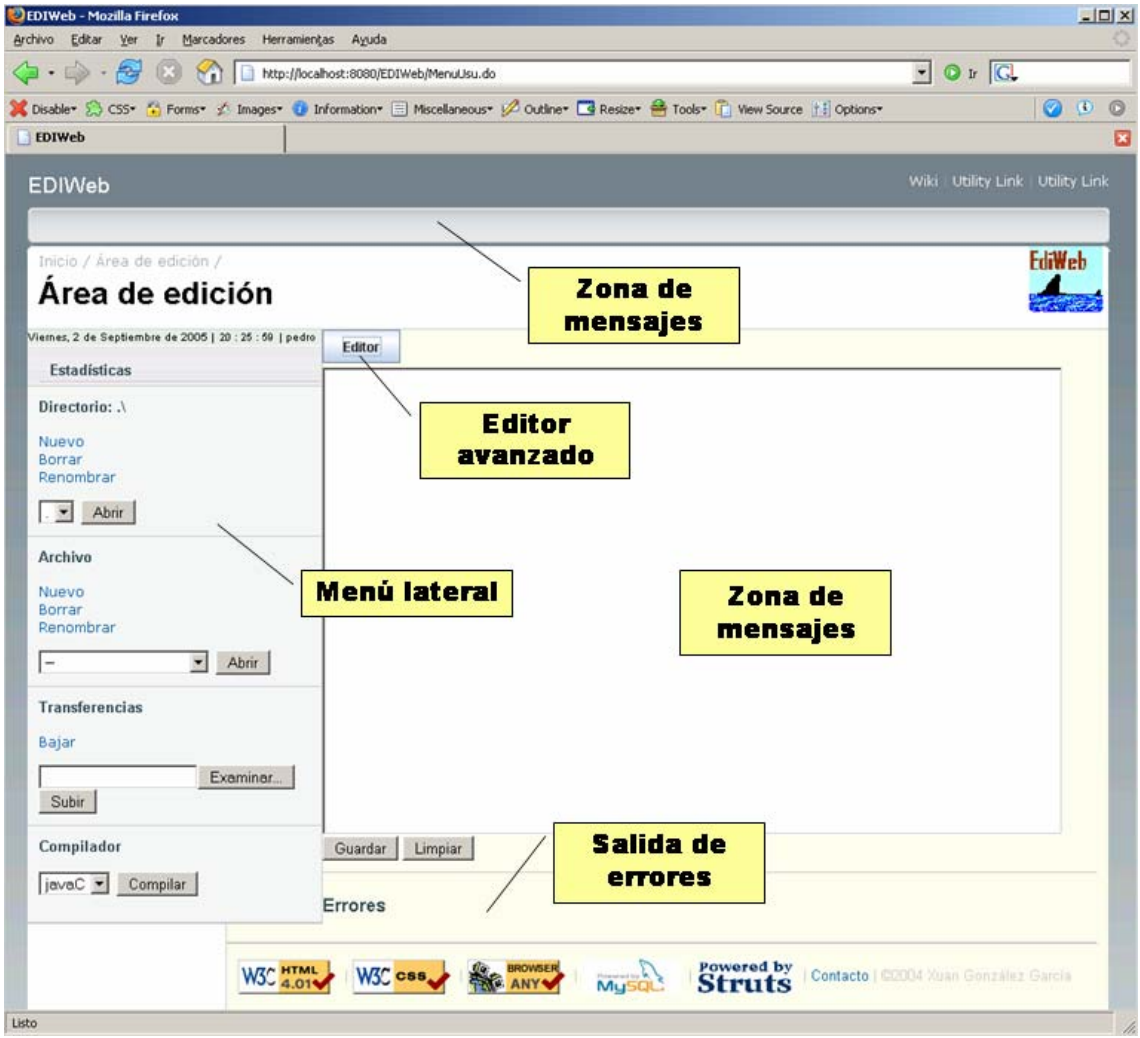


Figura 61. Partes de la interfaz de usuario

Con la primera aproximación se obtuvieron buenos resultados, un interfaz sencillo y fácilmente comprensible. Para la versión final (Figura 61) se mantuvo la estructura básica, pero se profesionalizó el diseño para darle una apariencia más seria. Se cambiaron los controles de archivos a la parte izquierda de la pantalla, dado que también los entornos de

desarrollo (Netbeans, Eclipse…), y los exploradores de archivos más conocidos (Internet Explorer de Windows, Nautilus, Xffm…) también lo sitúan ahí.

La barra de mensajes que en el prototipo se situó en la parte inferior, ahora se ha colocado en la parte superior. Las herramientas la sitúan normalmente en la parte inferior, pero al tratarse de una aplicación Web, no contener un marco de aplicación definido, y poder funcionar en ordenadores con muy distinta resolución de pantalla (aunque se recomienda 1024x768 como mínimo para trabajar cómodamente), esta barra podría quedar oculta en la parte inferior y ser molesto el tener que bajar la barra de scroll para verla. Por ello aún estando en un sitio no habitual, se ha preferido colocar en la parte superior.

Se ha comprobado su correcto funcionamiento con los dos navegadores más populares, el Mozilla Firefox (Figura 61), y el Microsoft Internet Explorer (Figura 62). El motor del Mozilla (Gecko) es usado a su vez en muchos otros navegadores, con lo que se supone igual apariencia en todos ellos. También se probó de forma satisfactoria en el navegador Konqueror, que dispone de un motor de renderizado propio, aunque en este navegador mostraba un pequeño error, que afecta estéticamente; pero no funcionalmente, y es que el tamaño del elemento de selección de archivo, es bastante superior a los otros navegadores, y se "sale" un poco de la cuadrícula del menú. Como se dijo, esto no afecta a la funcionalidad, solo en la apariencia. Se conseguido mantener un diseño sin frames, de acuerdo a los principios de usabilidad propuestos por Jakob Nielsen [Nielsen 1999].

Además del correcto funcionamiento, se ha intentado que la apariencia en ambos sea lo más parecida posible, esto no resulta sencillo ya que tienen un tratamiento distinto de CSS, distintos tamaños de elementos y distintas políticas de colocación.

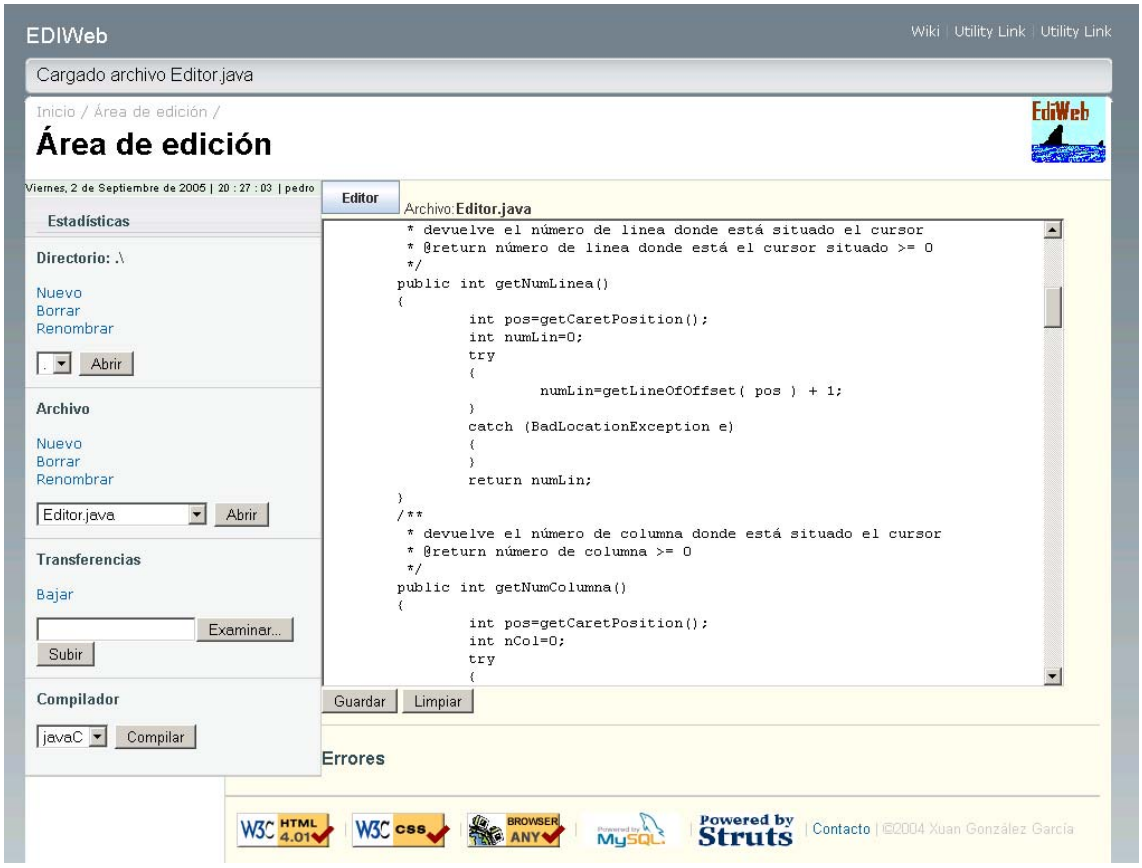


Figura 62: Vista del área de trabajo de usuario con el navegador Explorer 6 en castellano.

Se ha mantenido la misma estructura en todas las pantallas de la aplicación, solo variando el contenido del área de trabajo, el contenido del menú lateral, y la barra de navegación (esta no estaba presente en el prototipo).

Al editor avanzado (Figura 63), desplegado en forma de applet Java, se le han añadido funcionalidades de posicionamiento de cursor, indicando en todo momento en qué línea, columna, y número de carácter se encuentra, y cual es el total, esto ayuda a la hora de buscar un error, ya que el compilador informa del número de línea donde se encuentra. También se cambiaron las barras de desplazamiento, del tipo AWT, al tipo swing, esto permite desplazar el texto con los cursores, que antes no era posible, solo se podía mover la barra de desplazamiento con el ratón, lo cual hacía su uso muy molesto. Esto ha cambiado la apariencia de dicha barra, que se muestra con el aspecto del resto de componentes swing, en lugar de adoptar el estilo del sistema operativo. La apariencia es correcta en ambos casos, por lo que estéticamente no se considera que se haya mejorado o empeorado, pero funcionalmente si se ha mejorado su comportamiento.

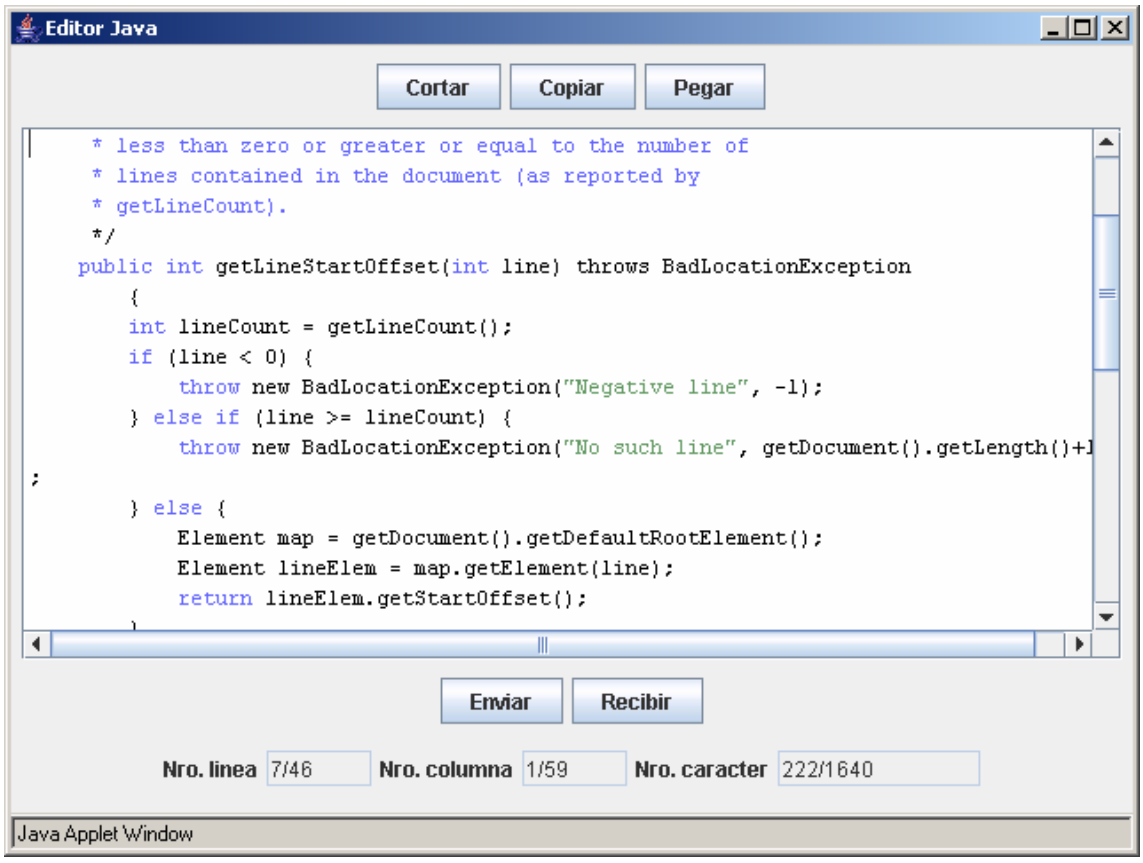


Figura 63: Versión final del applet Editor.

9.7 Evaluación de la base de conocimientos colaborativa

Para evaluar la idoneidad de la base de conocimientos colaborativa en el cometido de ayuda a la programación, se ha efectuado una prueba real en un entorno académico, con los

alumnos de la asignatura Estructura de Datos y de la Información de la Escuela de Ingeniería Técnica Informática de Oviedo[8].

Esta prueba ha consistido en probar la parte colaborativa en la construcción de la base de conocimiento, el prototipo que se utilizó se basa en un sistema Wiki, para ello se instaló el phpwiki [phpwiki], y se usó como herramienta de ayuda en los grupos de prácticas de la asignatura (Figura 64). El Wiki se administró de forma totalmente manual, el objetivo de la prueba era demostrar la idoneidad de una herramienta colaborativa para gestionar los conocimientos relacionados con el desarrollo de proyectos de programación de tamaño medio.

Figura 64. Pantalla principal del Wiki de pruebas.

En esas prácticas se les pide a los alumnos que realicen una aplicación Java que resuelva un determinado problema. Las prácticas duraron 3 meses aproximadamente, el número de alumnos implicado fue de unas 20 personas por grupo, 5 grupos.

Para resolverlo, pueden usar distintas estructuras de datos, y se pretende que el Wiki sirva como tablón de exposición de las distintas opciones, todas ellas aportadas, ampliadas y corregidas por los usuarios.

9.7.1 Encuesta

Para evaluar la utilidad del Wiki se pasaron 2 encuestas, una de respuestas libres en el propio Wiki, para conocer las preferencias de configuración y la forma concreta de utilizar

[8] http://petra.euitio.uniovi.es/asignaturas/edi/ediweb/

la herramienta, y otra en papel con preguntas tipo test sobre la actitud ante el Wiki que se evaluó según la escala de Likert (Figura 65). La encuesta consta de 2 bloques, uno de actitud, que mide la predisposición del usuario a las tecnologías en la educación, y otro de utilidad que trata de medir la utilidad real de la herramienta en el experimento.

Test EDIWiki

Edad……

Actitud general:

Valore el uso de tecnologías web en la enseñanza (1 nada positivo, 5 muy positivo)

1	2	3	4	5

Valore la tecnología wiki (1 nada positiva, 5 muy positiva)

1	2	3	4	5

Valore el presente wiki de 1 a 5

1	2	3	4	5

Aprendizaje con el wiki

Valore la organización del wiki (1 mal organizado, 5 bien organizado)

1	2	3	4	5

¿Le ha sido de utilidad en estas prácticas? (1 nada útil, 5 muy útil)

1	2	3	4	5

¿Resulta sencillo de usar? (1 muy difícil de usar, 5 muy fácil de usar)

1	2	3	4	5

¿Más o menos válido, para el uso que se le ha dado, que una lista de correo? (1 mucho menos válido, 5 mucho más válido)

1	2	3	4	5

¿Qué mejoraría? (Respuesta libre)

Figura 65. Encuesta

Si se puede demostrar que los resultados de ambos bloques de la encuesta se ajustan a la curva normal, con un intervalo de confianza de al menos un 95 % podríamos asumir que individuos con la misma predisposición hacia las tecnologías en la educación también tendrían una opinión positiva del uso del Wiki en la educación. Para ello primero se comprueba que ambas variables estén relacionadas, usando la prueba de Levene para

evaluar la homogeneidad de ambas varianzas. Los resultados de la encuesta se han transcrito de forma numérica y se ha hecho la media de puntuaciones de cada grupo (actitud y utilidad), por cada individuo del experimento ().

9.7.2 Resultados encuesta

Tabla 9. Resultados de la encuesta Test EDIWiki

Preguntas **Edad**	**1**	**2**	**3**	**4**	**promedio**	**5**	**6**	**7**	**8**	**promedio**
20	4	4	3	2	3,25	4	5	4	3	4
20	4	4	5	2	3,75	3	2	3	5	3,25
24	3	3	3	2	2,75	3	4	2	3	3
21	4	4	3	2	3,25	3	1	4	3	2,75
19	5	5	4	2	4	4	2	4	4	3,5
25	5	5	5	2	4,25	4	4	4	3	3,75
21	5	4	4	2	3,75	3	4	2	3	3
21	2	2	1	1	1,5	3	1	2	3	2,25
	4	4	4	3	3,75	3	3	3	4	3,25
20	5	5	5	2	4,25	4	2	5	4	3,75
24	4	4	4	4	4	3	4	5	4	4
24	4	4	4	4	4	3	3	4	3	3,25
24	5	3	3	3	3,5	4	4	4	3	3,75
	5	4	4	4	4,25	3	5	5	4	4,25
23	4	5	5	3	4,25	4	5	4	4	4,25
23	4	3	3	3	3,25	4	4	5	4	4,25
24	5	4	5	3	4,25	4	4	5	5	4,5
	3	3	3	2	2,75	3	3	4	3	3,25
22	5	4	3	2	3,5	3	3	4	3	3,25
21	5	4	3	3	3,75	4	4	4	4	4
24	5	4	3	2	3,5	2	1	2	3	2
	4	4	4	2	3,5	4	4	5	5	4,5
25	5	5	4	4	4,5	4	4	4	5	4,25
21	5	5	5	4	4,75	5	5	4	5	4,75
21	4	4	3	3	3,5	3	4	3	4	3,5
23	4	3	3	5	3,75	3	2	5	3	3,25
22	5	4	4	3	4	3	3	2	2	2,5
22	4	4	3	3	3,5	2	4	3	4	3,25
21	4	5	3	3	3,75	4	5	2	3	3,5
19	4	5	3	2	3,5	3	5	3	4	3,75
19	5	4	4	2	3,75	3	4	5	4	4
19	5	4	4	3	4	3	3	4	4	3,5
20	4	4	4	3	3,75	3	4	4	3	3,5
22	5	5	4	4	4,5	4	4	3	4	3,75

Una encuesta se eliminó de forma preliminar (casillas sombreadas), al errar en la pregunta de control, en la que se preguntaba si había utilizado el Wiki (la cuarta del bloque de

actitud). No había utilizado el Wiki por lo tanto sus respuestas no son válidas. El resto de las encuestas quedaron todas dentro del estudio. Los datos se trataron con el paquete estadístico SPSS® versión 11.5.1.

Tabla 10

	BLOQUE	Casos					
		Válidos		Perdidos		Total	
		N	Porcentaje	N	Porcentaje	N	Porcentaje
PROMEDIO	Actitud	33	100,0%	0	,0%	33	100,0%
	Utilidad	33	100,0%	0	,0%	33	100,0%

Prueba de homogeneidad de la varianza

		Estadístico de Levene	gl1	gl2	Sig.
PROMEDIO	Basándose en la media	2,159	1	64	,147
	Basándose en la mediana.	2,079	1	64	,154
	Basándose en la mediana y con gl corregido	2,079	1	60,745	,154
	Basándose en la media recortada	2,127	1	64	,150

Para que Levene demuestre la homogeneidad de las varianzas, el resultado Sigma debe ser >0,05. El resultado de la prueba de Levene es positivo.

Para demostrar que ambos bloques se ajustan a la normal, empleamos el test de Shapiro-Wilk, al igual que la prueba de Levene debe de tener una sigma >0,05 para que quede demostrada la hipótesis con un nivel de significancia del 95%.

Pruebas de normalidad

	BLOQUE	Kolmogorov-Smirnov(a)			Shapiro-Wilk		
		Estadístico	gl	Sig.	Estadístico	gl	Sig.
PROMEDIO	Actitud	,132	33	,153	,963	33	,307
	Utilidad	,125	33	,200(*)	,970	33	,489

* Este es un límite inferior de la significación verdadera.

Corrección de la significación de Lilliefors

Vemos que ambas Sigmas están bastante por encima del 0,05, con lo que demostramos la hipótesis.

9.8 Limitaciones del prototipo

IDEWeb resuelve muchos de los problemas que afectaban al Compilador Web SICODE, pero aún es claramente mejorable.

La primera mejora consistiría en integrar el entorno IDEWeb con el Sistema para la colaboración en el desarrollo de aplicaciones: COLLDEV, así como con el Sistema de análisis de errores en programas: PBA.

El applet editor puede ampliarse para incluir más funcionalidades que ayuden en la edición, al estilo de los modernos editores de código (autocompletado de código, detección de errores de sintaxis, detección avanzada de errores…). Las ampliaciones a IDEWeb son sencillas de hacer, dada la descomposición modular de que goza su código.

Son importantes todas las mejoras que se puedan hacer en el interfaz, este debería incorporar las facilidades de los entornos de programación tradicionales (Netbeans, Eclipse…), permitir abrir varios archivos, usando pestañas, e integrar el mayor número de herramientas posible.

Haría falta aumentar el número de lenguajes soportados por el editor avanzado, permitiendo resaltar la sintaxis a otros lenguajes, actualmente solo se soporta el lenguaje Java.

También se puede aumentar el número de estadísticas ofrecidas al usuario, incluyendo gráficos de las mismas.

9.9 Conclusiones

Hemos creado un entorno de desarrollo sobre Web que permite utilizar desde un navegador Web las herramientas esenciales en el proceso de desarrollo: editor, gestor de archivos, compilador y analizador de código y gestor de errores.

El entorno Web es interesante dada su universalidad, permite un interfaz común sin importar la tecnología que haya detrás, en la parte servidor. El hecho de poder acceder a aplicaciones desde un simple navegador Web es una ventaja que permite a los usuarios ser independientes del puesto de conexión que utilicen.

Una de las limitaciones del entorno es la capacidad limitada en la edición; para superar esta limitación se ha desarrollado un editor que se implementa como un applet para que el entorno se pueda seguir utilizando con un simple navegador.

La base de conocimientos colaborativa es una aportación esencial de este sistema. Su necesidad está justificada porque la interpretación de los mensajes de error de los compiladores se basa muchas veces, en la experiencia del desarrollador. Para acelerar la adquisición de esta experiencia buscamos que todos los desarrolladores aporten la suya en una base de conocimientos que está centrada en la comprensión y corrección de errores a través de ejemplos.

Para implementar esto hemos utilizado la tecnología Wiki, que ha demostrado su valor en otros campos. La concepción del Wiki permite que todo el mundo introduzca contenido en el sistema y supone un cambio en la concepción respecto a muchos sistemas en los que únicamente un experto o un grupo de expertos reducido puede aportar conocimiento al sistema y todos los demás únicamente pueden consultar. En este nuevo planteamiento todos pueden aportar con lo que la base de conocimientos se enriquece más rápidamente, y aunque pueda haber ciertos errores en el contenido estos pueden ser reparados rápidamente.

Wiki además permite añadir contenido desde el propio navegador sin herramientas especiales. Además, en nuestro sistema los usuarios no crean las páginas desde cero sino que el propio sistema introduce la información básica y a partir de ella los usuarios introducen sus experiencias concretas.

Capítulo 10. Clasificación de usuarios basada en la detección de errores

En este capítulo abordamos la realización de un estudio inédito hasta el momento, en el que analizamos errores de usuarios teniendo en cuenta su nivel de experiencia. Para ello utilizamos los avisos recogidos por de los proyectos desarrollados por los alumnos de distintos cursos de la titulación de Ingeniería Técnica en Informática tanto de Gestión como de Sistemas de Escuela Universitaria de Ingeniería Técnica en Informática[9] y alumnos del segundo ciclo de Ingeniería en Informática de la Escuela Politécnica Superior[10] de la Universidad de Oviedo. El objetivo del estudio es obtener datos que permitirán construir las bases para un modelo de usuario basado en el análisis de errores y realizar una clasificación de usuarios. De esta forma, el sistema podría permitir una adaptación a cada usuario dependiendo de su perfil de experiencia [González 2002][González 2004].

10.1 Precedentes en este tipo de estudios

Como se describía en el 3.6 *Gestores de prácticas avanzados* (Capítulo 3) existen actualmente gestores de prácticas que permiten recoger automáticamente proyectos realizados por estudiantes y someterlos a distintos test: compilarlos, realizar pruebas de ejecución, comprobar la coherencia de la documentación, etc. Utilizando estas facilidades algunos autores han realizado estudios en la línea en la que planteamos el nuestro.

Por ejemplo Huizinga [Huizinga 2001] realizan un estudio sobre los problemas encontrados en una práctica de una asignatura de programación. Define una serie de objetivos de cada proyecto relacionados jerárquicamente con los objetivos de la materia a enseñar y realiza una evaluación del cumplimiento de estos objetivos. Para ello 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. Elabora una serie de datos estadísticos sobre el comportamiento del conjunto de los alumnos al presentar una práctica. Huizinga plantea una asociación, basada en la intuición, entre cada dato estadístico y objetivo de aprendizaje. Por ejemplo: 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;

[9] La Web de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo, donde se puede encontrar el plan de estudios completo y el programa de las asignaturas citadas es: http://www.euitio.uniovi.es

[10] La Web de la Escuela Politécnica Superior, donde se puede encontrar el plan de estudios de las asignaturas de segundo ciclo: http://www.epsig.uniovi.es/

porcentaje de envíos finales con errores de ejecución puede reflejar una incapacidad para manipular las estructuras de datos.

Satratzemi [Satratzemi 2001] con ayuda de su herramienta AnimPascal (ver apartado RC.3.2.2) estudia como los alumnos van resolviendo los errores que surgen en una práctica planteada y pensada de antemano por los profesores. AnimPascal permite guardar las acciones de los estudiantes. Se consideran estas acciones como pasos en el proceso de solución del problema. Haciendo un seguimiento del camino recorrido por el estudiante para resolver cada problema, los profesores pueden descubrir "lagunas" de conocimiento y problemas de aprendizaje de sus estudiantes. Estudiando estos problemas de aprendizaje podrán ayudar a cada alumno particular.

La pretensión de nuestro trabajo es, como en los anteriores, estudiar los errores con los que se encuentran los estudiantes cuando realizan un proyecto software. Sin embargo, nuestro trabajo tiene varias diferencias con los descritos anteriormente:

- En nuestro trabajo no nos limitaremos a una práctica de una única asignatura, sino que trabajaremos con cuatro asignaturas más el proyecto fin de carrera nos interesa no sólo detectar errores puntuales en una práctica sino descubrir tendencias y la evolución de los errores según la tendencia.
- Utilizaremos nuestro sistema SICODE para detectar errores en el código fuente. El sistema, además del compilador, utiliza herramientas de análisis estático. Esto permitirá realizar un análisis más profundo que sólo con el compilador.
- En este experimento no estudiamos una evolución de los errores en el desarrollo de un proyecto sino que analizamos siempre proyectos ya terminados.
- Para descubrir problemas de aprendizaje nos basamos exclusivamente en errores en el código fuente, no tenemos en cuenta cuestiones relacionadas con la entrega, ni con la ejecución del proyecto.
- Los proyectos analizados son todos relativamente grandes. No se hace un análisis detallado sobre posibles errores lógicos en una práctica concreta; sino que pretendemos realizar un análisis estático de todos los errores que pueden cometer los desarrolladores.

10.2 Descripción del trabajo

El trabajo realizado se ha dividido en las siguientes etapas:

1. Recopilación de proyectos de distintos niveles (cursos) a partir de los proyectos entregados por los alumnos en cada asignatura.
2. Análisis de los mismos utilizando nuestro sistema SICODE y más concretamente el Sistema de Análisis de Errores en Programas (PBA) descrito en el Capítulo 7 *Sistema de análisis de errores de programas: PBA*.
3. Comparación de los datos de los proyectos en distintas etapas de aprendizaje de la programación, en concreto los tres primeros cursos de la titulación de ingeniería técnica en informática y en cuarto del segundo ciclo de Ingeniería Informática.
4. Caracterización de los errores en relación a los niveles de experiencia de los desarrolladores.

10.3 Elección de las muestras

Para realizar el estudio hemos utilizado los proyectos correspondientes a cuatro asignaturas: tres de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo y una del segundo ciclo de Ingeniería en Informática de la Universidad de Oviedo. Además del proyecto fin de carrera.

Las asignaturas seleccionadas son: Metodología de la Programación, Estructura de Datos y de la Información, Bases de Datos y Procesadores de Lenguaje. En todas se trabaja sobre conceptos del diseño y la programación orientada a objeto aplicados a diferentes aspectos de la programación.

La situación en el plan de estudios de las asignaturas es la descrita en la Tabla 11.

Tabla 11. Grupos de proyectos que se utilizarán en el experimento y descripción de las asignaturas a las que pertenecen

Asignatura	Curso	Duración	Tipo	Año de las muestras
Metodología de la programación	1º	2º cuatrimestre	Troncal	Curso 2002-2003
Estructura de datos y de la información	2º	Anual	Troncal	Curso 2003-2004
				Curso 2004-2005
Bases de Datos	3º	Anual	Troncal	Curso 2004-2005
Procesadores de Lenguaje	4º	Anual	Troncal	Curso 2004-2005
Proyecto Fin de Carrera	5º		Obliga	Distintos años.

10.3.1 Descripción de las prácticas de las asignaturas

En las tres primeras asignaturas: Metodología de la Programación, Estructuras de Datos y de la Información y Bases de Datos se realizan prácticas en las que los alumnos utilizan, de forma obligatoria, el lenguaje de programación Java para implementar el problema pedido; además de realizar el diseño, expresarlo mediante diagramas UML y realizar una documentación adjunta a la práctica. Para programar en lenguaje Java los alumnos utilizan un entorno de desarrollo integrado: el entorno JBuilder de Borland (ver descripción en el apartado 3.3.2 JBuilder).

Las prácticas están divididas en las tres asignaturas en dos módulos, con la diferencia que:

- Prácticas en Metodología de la Programación (primer curso). El tiempo de realización de cada módulo es la mitad del cuatrimestre y los dos módulos son independientes.
- Prácticas en Estructura de Datos y de la Información (segundo curso). El desarrollo de cada uno de los módulos ocupa un cuatrimestre completo y el segundo módulo es continuación del primero, suele conllevar cambios y ampliaciones; pero se reutiliza gran parte del código del primero.
- Prácticas en Bases de Datos (tercer curso). Como en los casos anteriores también se desarrollan dos módulos; pero el primero se desarrolla exclusivamente con SQL así que sólo consideramos el segundo módulo en el que se accede a la base de datos desde Java.

En la asignatura de Procesadores de Lenguaje (cuarto curso) los alumnos eligen el lenguaje de implementación entre C++ y Java y el entorno de desarrollo también es a elección del alumno, los alumnos que trabajan en Java suelen utilizar los entornos de desarrollo JBuilder

o Eclipse. Para nuestro estudio se han seleccionado exclusivamente las prácticas implementadas en Java.

Los alumnos realizan las prácticas de forma individual; excepto en el módulo de Bases de Datos. En clase de prácticas se trabaja sobre la comprensión del problema y sus requisitos y un diseño preliminar del mismo. Cada alumno trabaja personalmente en el diseño detallado y la implementación, buscando la mejor forma de realizar esta implementación y resolviendo los problemas que vayan surgiendo. El profesor resuelve dudas concretas en esta parte y orienta en el diseño detallado del proyecto.

10.3.2 Evaluación de las prácticas en las asignaturas

Para los proyectos de todas las asignaturas los criterios de evaluación incluyen el correcto funcionamiento de la aplicación cumpliendo los requisitos establecidos; pero también otras facetas como:

- Realización del diseño de la práctica utilizando diagramas UML, fundamentalmente de clases y de interacción, que ilustren las facetas más representativas del problema.
- Documentación del código mediante comentarios y utilización de Javadoc para la documentación de clases.
- Realización de una documentación complementaria adecuada.
- Escritura de código que permita un buen mantenimiento; para lo cual se establecen normas de escritura de código.

10.3.3 Entorno de desarrollo utilizado por los estudiantes

En todos los módulos, excepto en Procesadores de Lenguajes, los alumnos han utilizado como herramienta de desarrollo Borland JBuilder, en el caso de los cuatro primeros grupos se trataba de JBuilder 7.0 que utilizaba como compilador la versión 1.3.1 de la plataforma Java Software Develpment Kit (JSDK) de Sun Microsystems. En el caso de los dos últimos grupos que pertenecen al curso 2004-2005 han utilizado la versión JBuilder 9.0 que utiliza como compilador el de la plataforma JSDK 1.4.1_02 de Sun Microsystems. En la asignatura de procesadores de lenguaje se permite la libre elección del entorno de desarrollo por parte del alumno y un porcentaje alto elige el entorno Eclipse.

10.3.4 Descripción de los proyectos utilizados en el trabajo

Los proyectos evaluados son exclusivamente proyectos completos que, se supone, cumplen con la funcionalidad pedida en el problema. Por tanto, el número varía debido al número de alumnos matriculados en la asignatura ese año y al número de alumnos que entregan su proyecto. Se procesan todos los proyectos entregados independientemente de la evaluación posterior por parte del profesor; es decir se incluyen también los proyectos de los alumnos suspensos.

En la Tabla 12 se pueden ver los proyectos procesados para cada uno de los subgrupos. Se puede apreciar la mayor complejidad de las prácticas de Estructura de Datos y de la Información en relación al número de clases por proyecto y al tamaño de cada una de las clases.

Tabla 12. Descripción de los grupos de proyectos seleccionados para el experimento

Asignatura	Curso	Núm. proyectos	Num. Fich. *.java	Media fich/proy	Tamaño fich .java (bytes)	Media tam/fich (bytes)
Metodología de la programación	1º	316	4.473	14,16	14.029.045	3.136,38
Estructura de datos y de la información	2º	337	8.319	24,69	45.192.259	5.432,41
Bases de Datos	3º	22	492	22,36	3.427.435	6.966,33
Procesadores de Lenguaje	4º	5	398	79,60	1.692.506	4.252,53
Proyectos Fin de Carrera		9	695	77,22	5.509.856	7.927,85

10.4 Proceso previo de los datos

Los proyectos utilizados para la realización de este trabajo son prácticas realizadas por los estudiantes de las asignaturas enunciadas anteriormente. Los proyectos fueron realizados en los laboratorios de prácticas con la presencia del profesor de la asignatura y otro tiempo de trabajo individual por parte del estudiante. No se impuso ningún requisito extra a los alumnos, en relación con este estudio, simplemente los ya establecidos en las especificaciones de prácticas. Debido en parte a que los alumnos han utilizado herramientas externas a nuestro sistema como entornos integrados de desarrollo (en concreto JBuilder de Borland), fue necesario un proceso previo para adaptar los proyectos al tratamiento que queríamos realizar con nuestro sistema.

JBuilder genera dos tipos de archivos que afectan a nuestro sistema:

- Genera una serie de archivos de copias de seguridad *.java~<num>~
- Tenemos los .class generados en un directorio aparte de los .java

Para adaptar los proyectos a nuestro sistema debemos realizar una serie de operaciones que no son necesarias si hubiésemos utilizado SICODE para el desarrollo:

- Quitar el atributo sólo lectura, debido a que hay proyectos que proceden de un CD y por tanto los directorios impiden escribir archivos resultado de la compilación necesarios para el proceso que realizamos.
- Eliminar todos los archivos que cumplen el patrón "*.java~*"
- Eliminar todos los archivos que cumplen el patrón "*.class"
- Verificar que los nombres de los directorios no sean muy largos para no hacer paths largos.
- Comprobar que la extensión de los archivos es .java en minúsculas ya que sino el parser de Ant no los encuentra.

Por otra parte, también se han eliminado determinados errores que aparecían en el análisis; pero que venían provocados por determinadas configuraciones de las herramientas y por tanto no son reales:

- En los entornos integrados se pueden declarar paquetes externos comunes que luego los alumnos no incluyen a la hora de hacer la entrega del proyecto.
- En mpmod1 y mpmod2 existe un archivo que se proporcionó a los alumnos que tenía código que nunca se llegaba a ejecutar; esto aparecía como un aviso en la versión 1.3 del compilador javac (la que utilizaron los alumnos), con lo cual no ocasionaba problemas; sin embargo, en la versión 1.4 de javac (la que utilizamos en el análisis) esto es notificado como un error.

10.5 Descripción del proceso al que se someten los proyectos

Una vez recopilados los proyectos de las distintas asignaturas y realizadas las operaciones previas utilizamos el prototipo *Sistema de análisis de errores de programas: PBA* descrito en el Capítulo 7 para realizar un análisis exhaustivo del código fuente escrito por los estudiantes y obtener así que errores suelen cometer.

10.5.1 Herramientas utilizadas y configuración de cada herramienta en el análisis

En este apartado enumeramos cada una de las herramientas que utiliza el prototipo *Sistema de análisis de errores de programas: PBA* para realizar el análisis. Estas herramientas se han descrito en el Capítulo 7. Algunas de ellas permiten configurar los tipos de errores que detectan, esto ha permitido especificar los errores concretos que queremos buscar con estas herramientas para potenciar su efectividad.

Javac

Utilizamos el compilador estándar de Java javac que está incluido en la plataforma sdk versión 1.4.2 de Sun Microsystems. La configuración es la que tiene por defecto ya que no se pasa ninguna opción de configuración.

Antic

La única opción que se le pasa al hacer la llamada es '-java' que indica que va a procesar un programa escrito en lenguaje Java, en vez de C que es la opción por defecto.

Findbugs

Herramienta findbugs versión 0.7.3.

Opciones de configuración que se utilizan: no se comprueban los errores de concurrencia entre hilos.

Jlint

Opciones que se utilizan al llamar a este programa desde la unidad de compilación realizada para el experimento: no se comprueban los errores de sincronización entre hilos.

PMD

Herramienta PMD versión 1.6. Esta herramienta permite definir reglas para verificar la corrección de los programas analizados. Se ha configurado esta herramienta para que detecte los errores recogidos en la Tabla 13 y la Tabla 14.

Tabla 13. Tipos de errores que hemos activado en PMD para su detección (1)

Sentencias vacias	Simplicidad del código	Convenciones de código y de nombres
NonCaseLabelInSwitchStatement	DontImportJavaLang	ForLoopsMustUseBracesRule
EmptyWhileStmt	DuplicateImports	IfElseStmtsMustUseBracesRule
EmptyFinalizer	SimplifyBooleanExpressions	WhileLoopsMustUseBracesRule
EmptySwitchStatements	UnnecessaryReturn	IfStmtsMustUseBraces
EmptyFinallyBlock	ForLoopShouldBeWhileLoop	SwitchStmtsShouldHaveDefault
EmptyStaticInitializer	SimplifyBooleanReturnsRule	VariableNamingConventionsRule

Sentencias vacias	Simplicidad del código	Convenciones de código y de nombres
EmptyTryBlock	UnconditionalIfStatement	ClassNamingConventionsRule
EmptyStatementNotInLoop	UnnecessaryConversionTemporaryRule	MethodNamingConventions
EmptyIfStmt	StringInstantiation	AbstractNamingRule
EmptySynchronizedBlock	StringToString	MethodWithSameNameAsEnclosingClass
	ImportFromSamePackage	AvoidDollarSigns

Tabla 14. Tipos de errores que hemos activado en PMD para su detección (2)

Código no utilizado	Mejoras de diseño	Errores potenciales
UnusedLocalVariable	UseSingletonRule	ExplicitCallToFinalize
UnusedImports	LooseCouplingRule	JumbledIncrementer
UnusedFormalParameter	SwitchDensity	OverrideBothEqualsAndHashcodeRule
UnusedPrivateField	NonStaticInitializer	DoubleCheckedLockingRule
	FinalFieldCouldBeStatic	AvoidReassigningParametersRule
	ProperCloneImplementationRule	CloseConnectionRule
	ConstructorCallsOverridableMethodRule	BooleanInstantiation
	FinalizeShouldBeProtected	FinalizeOnlyCallsSuperFinalize
	AvoidDuplicateLiterals	FinalizeDoesNotCallSuperFinalize
	DefaultLabelNotLastInSwitchStmt	AvoidCatchingThrowable
	SignatureDeclareThrowsException	SuspiciousHashcodeMethodName
	ExceptionTypeChecking (AvoidInstanceofChecksInCatchClause)	FinalizeOverloaded
		ReturnFromFinallyBlock

La forma de indicar al *Sistema de análisis de errores de programas: PBA* las herramientas que queremos utilizar, la configuración de cada una de ellas y el orden de ejecución se realiza a través de lo que llamamos *unidad de compilación*. En el Capítulo 7 se describe el diseño de estos archivos, así como la forma de uso.

10.5.2 Tipos de problemas buscados

La combinación de las herramientas enumeradas en el apartado anterior permiten a nuestro sistema detectar gran cantidad de errores, proporcionando una efectividad mucho mayor que el análisis realizado por el compilador del lenguaje.

A partir de los errores que detectan cada una de las herramientas con la configuración planteada hemos realizado una clasificación en una serie de tipos de error que permiten orientar el análisis.

En la Tabla 15 se incluyen los tipos de errores junto con su descripción que resumen el ámbito de errores buscados en este trabajo.

Tabla 15. Descripción de los distintos tipos de problemas buscados

Tipo error	Descripción
Aviso	Aviso de posible error en una aplicación.
Aviso de código ineficiente	Aviso sobre la posible ineficiencia de zonas del código.
Aviso de código innecesario	Aviso sobre zonas del código que son innecesarias o redundantes.
Aviso de código no usado	Aviso sobre zonas de código no usadas en el programa.
Aviso de comparación	Aviso de posibles problemas a la hora de llevar acabo una comparación.
Aviso de diseño	Aviso de un posible fallo a la hora de diseñar las aplicaciones, puede ser interesante pensar en una Refactorización del código.
Aviso de dominios	Aviso de posible fallo en el dominio de una variable, en el rango de una matriz, en la valor de retorno de una función, etc.
Aviso de estilo	Aviso de una violación de los convenios de estilo de codificación del Lenguaje. Estas normas pueden ayudar a evitar errores en algunas ocasiones.
Aviso de estructura de control	Aviso de un posible error en una estructura de control de flujo del programa.
Aviso de excepciones	Aviso de zonas de código que pueden ser propensas a lanzar excepciones, o bien de zonas donde se está haciendo un mal tratamiento de estas.
Aviso de finalize()	Aviso sobre posibles problemas en el uso del método finalize().
Aviso de inicialización	Aviso sobre un posible problema en las inicializaciones o en los constructores.
Aviso de JUnit	Aviso sobre un posible problema en el uso de la herramienta de prueba JUnit.
Aviso de literales	Aviso sobre un posible error en la escritura de un literal.
Aviso de nombre	Aviso sobre un problema potencial por no seguir las normas estándares de estilo para los nombres o por una mala elección de los nombres de campos, métodos o clases que pueden llevar a confusiones.
Aviso de ocultación	Aviso provocado por una posible ocultación de campos o métodos en una clase.
Aviso de operadores	Aviso sobre posibles errores en el uso de los operadores.
Aviso de sincronización	Aviso sobre posibles errores en la sincronización en aplicaciones multihilo.
Aviso de vulnerabilidad	Aviso sobre posibles vulnerabilidades de las clases ante código mal intencionado.
Error de compilación	Error encontrado por el compilador al analizar el código fuente.

10.6 Resultados del estudio

10.6.1 Información incluida en las tablas de resultados

En los siguientes apartados mostramos los resultados obtenidos tras el tratamiento de los proyectos correspondientes a diferentes cursos de la titulación de Ingeniería Informática (en la Universidad de Oviedo, la titulación se encuentra dividida entre primer y segundo ciclo).

Tabla 16. Tabla de avisos de incumplimiento de convenciones descartados en el análisis.

Código error	Descripción del error	Tipo de error
13019	Avoid using "if...else" statements without curly braces	Convenciones de código
13017	Avoid using "if" statements without curly braces	Convenciones de código
13071	Method name does not begin with a lower case character.	Convenciones de nombres
13020	Avoid using "for" statements without curly braces	Convenciones de código
13089	Method names should not contain underscores	Convenciones de nombres
13088	Variables should start with a lowercase character	Convenciones de nombres
13087	Variables that are not final should not contain underscores.	Convenciones de nombres
13091	Class names should not contain underscores	Convenciones de nombres
13073	Abstract classes should be named AbstractXXX"	Convenciones de nombres

Aunque la configuración de las herramientas incluida en la *unidad de compilación* utilizada para este trabajo detecta y recoge avisos correspondientes al incumplimiento de convenios de nombres y de código, estos convenios se han eliminado a la hora de elaborar las tablas (ver Tabla 16), ya que lo que pretendemos estudiar son avisos que indiquen la presencia de errores reales.

Los resultados del estudio se muestran en forma de tabla y aparecen ordenados en orden decreciente por el número absolutos de avisos que han aparecido para cada grupo de proyectos. En cada tabla de datos se incluye la siguiente información para cada uno de los avisos:

- Código del error. Es el código asignado al aviso correspondiente. Es un identificador único para cada aviso asignado por el Sistema de Análisis de Errores en Programas e independiente de la herramienta utilizada.
- Descripción del error. Descripción del aviso detectado. Se incluye en inglés basándonos en la descripción original que hace cada herramienta del error.
- Número de errores. Número total de avisos de este tipo en el grupo de proyectos analizado.
- Núm. proyectos con errores. Número de proyectos que han generado este tipo de aviso.
- Media de errores por proyecto. Es el número medio de avisos de un tipo determinado en los proyectos que contienen este tipo de error.

$$Media\_errores\_por\_proyecto = \frac{Número\_errores}{Núm\_proyectos\_con\_errores}$$

- Porcentaje de proyectos con errores. Es el porcentaje de proyectos que contienen este tipo de avisos respecto al total de proyectos analizados para este grupo.

$$Porcentaje\_proy\_errores = \frac{Núm\_proyectos\_con\_errores}{Total\_proyec} \cdot 100$$

- Total proyectos. Es el total de proyectos analizados para este grupo.
- En las gráficas el valor de la frecuencia de error para cada uno de los errores, se refiere a:

$$Frecuencia\_error = \frac{Número\_errores}{Total\_proyec}$$

10.6.2 Resultados del análisis de los proyectos correspondientes a primer curso

En este apartado describimos los resultados del análisis de los proyectos correspondientes a la asignatura de primer curso: Metodología de la programación. En primer lugar hacemos una mínima descripción de la asignatura y las prácticas planteadas en ella. Incluimos una tabla resumen de los errores más frecuentes con la información descrita en el apartado 10.6.1 y una gráfica con la misma información generada por el prototipo PBA, por último hacemos un análisis de los resultados para este curso.

Descripción de la asignatura: Metodología de la programación

- Es una asignatura donde se profundiza en los conceptos de orientación a objetos y técnicas de programación de algoritmos: recursividad y algoritmos de ordenación.
- Situación dentro del plan de estudios. Esta asignatura se imparte en primer curso de la carrera en su segundo cuatrimestre.
- Asignaturas de programación precedentes. Los alumnos que realizan esta asignatura han visto los fundamentos de la programación orientada a objeto en otra asignatura

del primer cuatrimestre denominada Introducción a la Programación. En esta asignatura también se realizan prácticas utilizando Java como lenguaje de implementación y el entorno JBuilder con lo que los alumnos ya están familiarizados con las funciones básicas del entorno y los fundamentos del lenguaje.

Prácticas que se proponen en la asignatura para su resolución

- Módulo 1 de Metodología de la programación (identificador: mpmod1). Consiste en la simulación de un juego de rol muy sencillo en el que se crea un personaje con determinadas características, unos escenarios con objetos o adversarios y una forma de luchar con los adversarios y recorrer los escenarios. En este proyecto el alumno debe crear estructuras dinámicas secuenciales, esencialmente listas y llevar a cabo una interacción con el usuario para decidir las acciones del personaje. No existe interacción con archivos.
- Módulo 2 de Metodología de la programación (identificador: mpmod2). Gestión de una biblioteca con autores y libros. Estos se organizan en memoria mediante listas dinámicas y la información se lee desde archivos almacenados en disco.

Tabla 17. Avisos más frecuentes en la asignatura de primer curso: Metodología de la Programación

Código error	Descripcion	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13044	Avoid calls to overridable methods during construction	2456	260	9,45	82,28%
12001	Component in this class shadows one in base class.	1873	118	15,87	37,34%
13039	Avoid unnecessary comparisons in boolean expressions	1484	183	8,11	57,91%
13080	The same String literal appears several times in this file.	1213	268	4,53	84,81%
13084	Avoid unused local variables.	1181	275	4,29	87,03%
12002	Local variable shadows component of class.	1033	176	5,87	55,70%
12014	Compare strings as object references.	769	119	6,46	37,66%
13040	Switch statements should have a default label	520	191	2,72	60,44%
13000	Avoid empty catch blocks	416	237	1,76	75,00%
11004	Comparison of String objects using "==" or "!="	296	117	2,53	37,03%
13016	An empty statement (semicolon) not part of a loop	289	229	1,26	72,47%
	Número total de proyectos	316			

Análisis de los avisos detectados con mayor frecuencia en los proyectos de este curso

Aparecen cuatro tipos de avisos que se dan en más del 75% de los proyectos:

- ***Avoid calls to overridable methods during construction***. Este aviso denota un proceso de construcción del objeto con muchos pasos y que por tanto, requiere la llamada a métodos auxiliares; sin embargo, estos métodos auxiliares no son exclusivos de la clase que pretendemos instanciar sino que redefinen método que ya existen en la superclase. Esto conlleva un error conceptual en el modelo de constructor que pretende hacer demasiadas cosas. Además, esto podría provocar errores difíciles de encontrar al invocar a métodos sin la seguridad de que todos los datos miembro de los objetos están correctamente inicializados.
- ***The same String literal appears several times in this file***. Este aviso simplemente afecta a la claridad del código ya que se podría haber creado una

constante y utilizarla en los distintos puntos de código; sin embargo, esto unido a otras cuestiones puede hacer más complicado la modificación y el mantenimiento del código fuente.

- ***Avoid unused local variables***. El aviso indica que después de las sucesivas modificaciones del código fuente han quedado variables locales que no se utilizan. Esto fundamentalmente afecta a la comprensión del código fuente ya que el desarrollador que lee el código va a analizar cada una de las variables que se declaran que sin embargo no van a ser utilizadas posteriormente.

Figura 66. Avisos más frecuentes para el primer curso

- ***Avoid empty catch blocks***. Este aviso denota que no hay código que maneje la excepción capturada para avisar al usuario de un problema irresoluble de forma automática o hacer una recuperación del error. Esto fundamentalmente es debido a que Java obliga a capturar las posibles excepciones y por eso el alumno utiliza la cláusula *catch*; pero no sabe lo que puede hacer con ella una vez capturada. Esto puede provocar un "retorno silencioso" es decir que salte una excepción y como es capturada en el catch, continúe la ejecución como si nada hubiese pasado y más tarde empiecen a ocurrir errores en la ejecución. Con lo cual el usuario que ejecuta la aplicación no se da cuenta que ha ocurrido un problema, ya que la excepción es capturada; pero hay problemas de funcionamiento cuya causa no está clara.

Otro aviso que no aparece en tantos proyectos; pero que es significativo ya que denota problemas en el modelo mental del alumno es: "Comparison of String objects using "==" or "!=""". Aquí se intentan comparar cadenas que en Java son objetos como si fuesen tipos primitivos, con el operador ==, con lo que en realidad se están comparando las referencias, en vez de utilizar el método equals.

10.6.3 Resultados del análisis de los proyectos correspondientes a segundo curso

En este apartado describimos los resultados del análisis de los proyectos correspondientes a la asignatura de primer curso: Estructura de datos y de la información. En primer lugar hacemos una mínima descripción de la asignatura y las prácticas planteadas en ella. Incluimos una tabla resumen de los errores más frecuentes con la información descrita en el apartado 10.6.1 y una gráfica con la misma información generada por el prototipo PBA, por último hacemos un análisis de los resultados para este curso.

Descripción de la asignatura: Estructura de Datos y de la Información

- Se estudian las estructuras de datos básicas: listas y sus variantes, árboles, tablas hash, grafos.
- Situación dentro del plan de estudios. Esta asignatura se imparte en el segundo curso de la carrera y tiene una duración anual.
- Asignaturas de programación precedentes. Previamente han visto como asignaturas de programación las citadas anteriormente de Introducción a la Programación, Metodología de la Programación y Comunicación Persona Máquina (segundo cuatrimestre de primer curso). En esta última asignatura también se realizan prácticas que conllevan la implementación de prototipos aunque haciendo hincapié en la interfaz de usuario. Sin embargo, también se utiliza el mismo entorno de desarrollo con lo que sirve a los alumnos para adquirir una mayor experiencia con él.

Tabla 18. Avisos más frecuentes para la asignatura de segundo: Estructuras de Datos y de la Información

Código error	Descripcion	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13046	This final field could be made static	4161	224	18,58	66,47%
11021	Unread field: should this field be static?	3792	211	17,97	62,61%
13080	The same String literal appears several times in this file.	2249	287	7,84	85,16%
13084	Avoid unused local variables.	1993	292	6,83	86,65%
13000	Avoid empty catch blocks	1764	207	8,52	61,42%
12002	Local variable shadows component of class.	1684	262	6,43	77,74%
13078	A method should'nt have Exception in throws declaration.	1358	98	13,86	29,08%
13042	Avoid reassigning parameters.	1342	267	5,03	79,23%
13040	Switch statements should have a default label	1279	254	5,04	75,37%
13039	Avoid unnecessary comparisons in boolean expressions	1121	135	8,30	40,06%
10013	May be wrong assumption about ELSE branch association	1020	166	6,14	49,26%

337

Prácticas que se proponen en la asignatura para su resolución

- Módulo 1 de Estructura de Datos y de la Información (identificador: edi3mod1). Aplicación para una empresa de autocares que tiene líneas regulares que comunican distintas poblaciones. La aplicación debe permitir solucionar a la empresa todas las necesidades de información pre-venta de los billetes a los viajeros que preguntan sobre itinerarios, horarios, tipo de autocar, servicios que se ofrecen durante el viaje y precio del billete. Toda la información sobre itinerarios, horarios, tipo de autocar y servicios estará almacenada en un archivo del tipo BML un lenguaje creado a

partir de XML. La aplicación deberá permitir cargar el archivo sobre estructuras de datos en memoria para gestionar esta información. Así mismo, debe permitir volver a guardar la información de memoria en un archivo con la misma estructura. Los alumnos deben desarrollar a partir de los paquetes básicos la lectura de los archivos XML, no utilizarán ningún paquete específico para este propósito. El usuario dispondrá de un menú para elegir las acciones que quiere realizar.

- Módulo 2 de Estructura de Datos y de la Información (identificador: edi3mod2). Se trata de ampliar edi3mod1 con las siguientes características: un Arbol Binario de Búsqueda que permita almacenar y buscar poblaciones, este almacenamiento se realiza en un formato diferente al resto de los datos y para su lectura se emplea introspección mediante la cual se crean los objetos adecuados para el almacenamiento y el tratamiento de la información en memoria, un TAD Matriz Dinámica Genérica que permita almacenar horarios y desarrollar una opción que permita mostrar las posibles alternativas para viajar entre dos poblaciones pudiendo hacer transbordos.

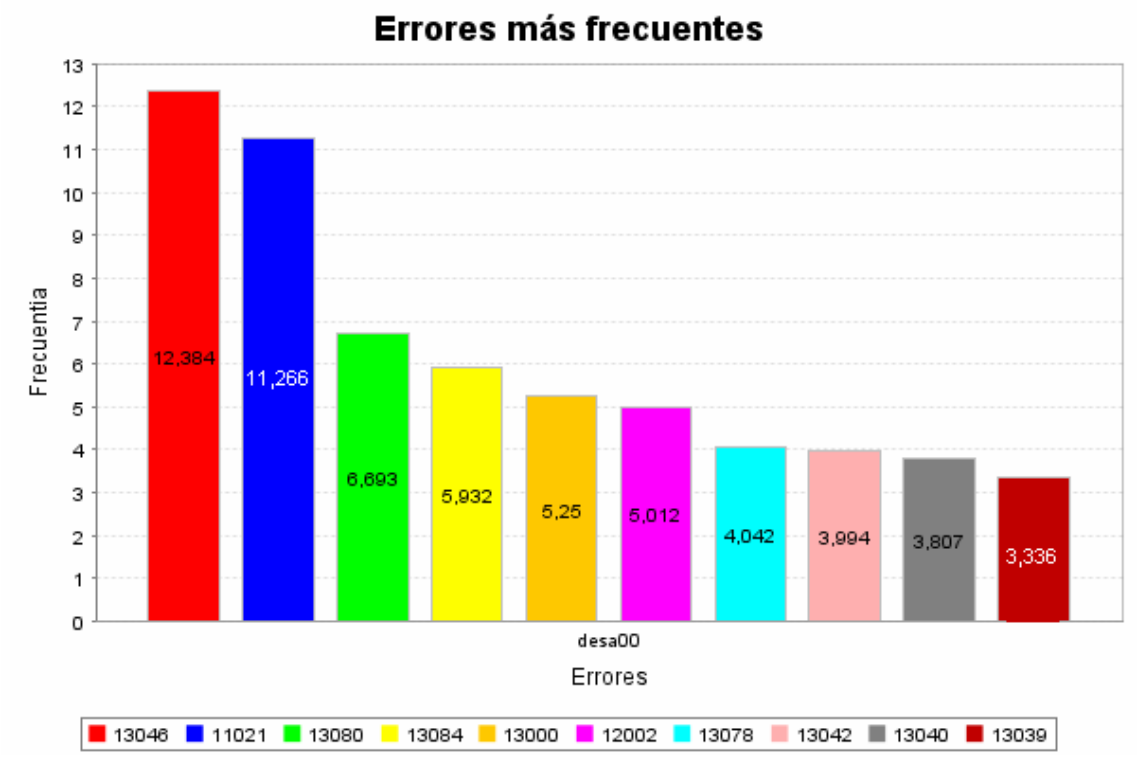


Figura 67. Avisos más frecuentes para el grupo de proyectos de segundo curso

- Módulo 1 de Estructura de Datos y de la Información (identificador: edi4mod1). Gestor de datos de una mini biblioteca en la cual sólo habrá libros. Toda la información del gestor estará almacenada en un archivo del tipo DML (un lenguaje creado a partir de XML). La aplicación deberá permitir cargar el archivo sobre estructuras de datos en memoria y gestionar esta información. Así mismo, debe permitir volver a guardar la información de memoria en un archivo con la misma estructura. La carga del archivo DML se realizará en la siguiente estructura de datos, un nodo con dos referencias: Lista de libros, es una lista simplemente encadenada y ordenada en la que tenemos el código del libro y el título del mismo. Índice, es un árbol de búsqueda que contiene los códigos de los libros así como una

referencia a un nodo de la lista de libros. Las acciones sobre el gestor se realizan mediante una interfaz en línea de comandos.

- Módulo 2 de Estructura de Datos y de la Información (identificador: edi4mod2). Se trata de ampliar edi4mod1 con las siguientes características: generalizar el gestor para que pueda leer archivos con distinto tipo de datos, se pide a los alumnos que realicen esto mediante introspección; realizar joins entre distintas tablas.

Análisis de los avisos detectados con mayor frecuencia en los proyectos de este curso

Existen cinco tipos de avisos que están muy extendidos (aparecen en más del 75% de los proyectos):

- ***The same String literal appears several times in this file***. Este aviso ya aparecía entre los más extendidos en los proyectos de primer curso y puede provocar problemas en el mantenimiento de las aplicaciones.
- ***Avoid unused local variables***. Este aviso también aparecía en el apartado anterior y provoca problemas en la lectura y por tanto en la comprensión del código.
- ***Local variable shadows component of class***. Este problema también dificulta la comprensión del código ya que al coincidir el nombre de la variable local y del componente hay que tener claro en primer lugar que hay dos elementos que se llaman igual y además, hay que saber a cuál nos estamos refiriendo en cada punto del código.
- ***Avoid reassigning parameters***. La reasignación de parámetros puede provocar errores debido a los efectos laterales y además dificulta la lectura del código.
- ***Switch statements should have a default label***. Java no obliga a que todas las sentencias switch tengan una etiqueta default sin embargo en estas condiciones puede ocurrir que la condición de la sentencia tome un valor inesperado y por tanto el flujo de ejecución no entre por ninguna de las etiquetas previstas provocando un mal funcionamiento cuya causa es difícil de determinar ya que ocurre lejos de esta.

10.6.4 Resultados del análisis de los proyectos correspondientes a tercer curso

En este apartado describimos los resultados del análisis de los proyectos correspondientes a la asignatura de primer curso: Bases de datos. En primer lugar hacemos una mínima descripción de la asignatura y las prácticas planteadas en ella. Incluimos una tabla resumen de los errores más frecuentes con la información descrita en el apartado 10.6.1 y una gráfica con la misma información generada por el prototipo PBA, por último hacemos un análisis de los resultados para este curso.

Descripción de la asignatura: Bases de Datos

- El objetivo de la asignatura es que el alumno sepa diseñar y crear el esquema de una base de datos y conozca distintas herramientas de acceso como la utilización de SQL como lenguaje de modificación y consulta; así como el acceso desde lenguajes de programación convencionales. En el caso del módulo que analizamos se trata de desarrollar un pequeño proyecto en el que se diseñe y cree una base de datos adecuada al problema, programación del servidor mediante procedimientos

almacenados en la base de datos, programación del cliente mediante lenguaje Java y un controlador que permita el acceso a la base de datos.

- Situación dentro del plan de estudios. Esta asignatura se estudia en tercero y tiene una duración anual.

Tabla 19. Avisos más frecuentes para la asignatura de tercer curso: Bases de Datos

Código error	Descripcion	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13078	A method shouldn't have Exception in throws declaration.	269	15	17,93	68,18%
13084	Avoid unused local variables.	113	13	8,69	59,09%
11023	This field is never read. Consider removing it from the class.	113	13	8,69	59,09%
13080	The same String literal appears several times in this file.	111	17	6,53	77,27%
13044	Avoid calls to overridable methods during construction	97	16	6,06	72,73%
13061	Avoid unused imports	58	13	4,46	59,09%
13059	Avoid duplicate imports	38	12	3,17	54,55%
11011	Redundant comparison of a reference value to null.	25	2	12,50	9,09%
13000	Avoid empty catch blocks	21	5	4,20	22,73%
13039	Avoid unnecessary comparisons in boolean expressions	15	5	3,00	22,73%
13083	Avoid unused private fields.	15	5	3,00	22,73%

Número de proyectos	22

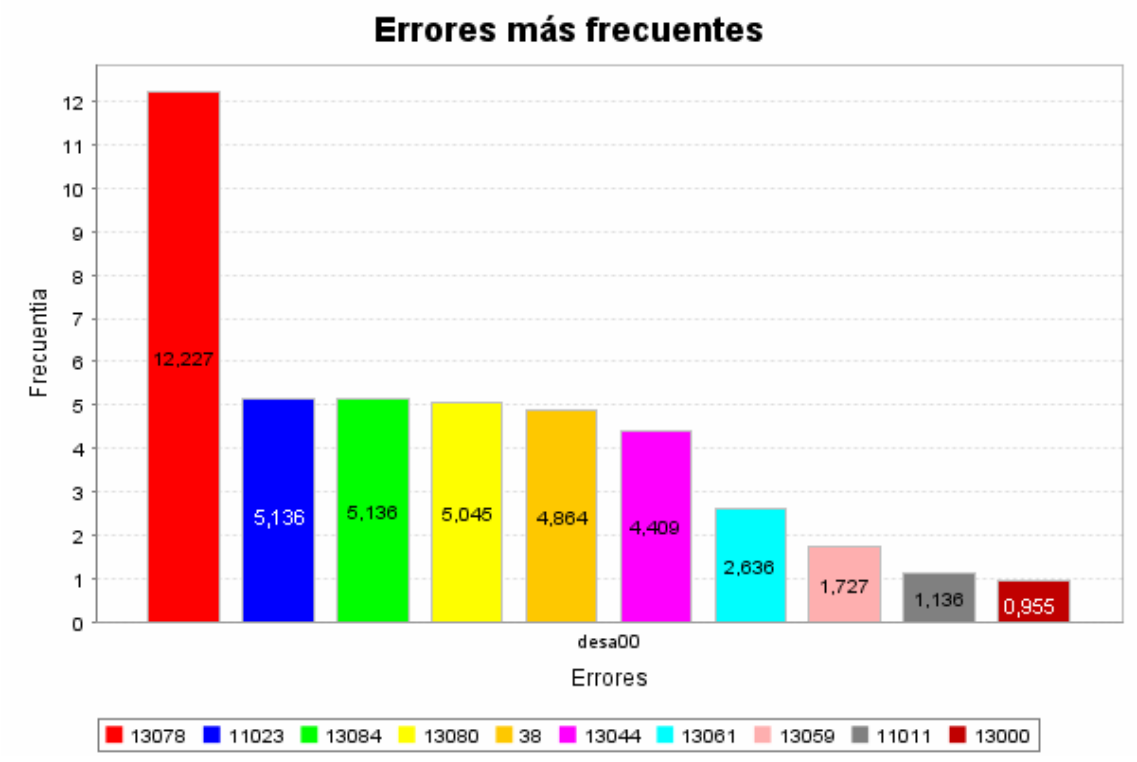


Figura 68. Avisos más frecuentes para el grupo de proyectos de tercer curso

- Asignaturas de programación precedentes. Estructuras de Datos y de la Información (segundo curso), la parte de las estructuras de datos en memoria secundaria, como archivos indexados, árboles B, etc. Introducción a la Programación, Metodología de la Programación y Tecnología de la Programación (primer y segundo curso), en cuanto a conocimientos generales de desarrollo de

proyectos de software, el modelo orientado a objetos y lenguajes de programación, y programación en general.

Prácticas que se proponen en la asignatura para su resolución

- Bases de datos. Se trata de desarrollar proyecto amplio de gestión de un campeonato de Formula 1. Con un módulo en el servidor basado en procesos almacenados en la base de datos, en este caso Oracle, programados en PL-SQL y un módulo en el cliente que hace interfaz con el usuario y realiza varios tipos de consultas y modificaciones sobre la base de datos, este módulo programado con lenguaje Java.

Análisis de los avisos detectados con mayor frecuencia en los proyectos de este curso

Sólo hay un aviso que aparece en más del 75% de los proyectos: ***The same String literal appears several times in this file***. Este es un error que aparece reiteradamente en todos los cursos, ya que al no tener gran importancia no se trata sobre él en clase y por tanto no se corrige.

Por otra parte, el error que aparece mayor número de veces aunque en menos proyectos es: ***A method shouldn't have Exception in throws declaration***. Este también es un problema de claridad de lectura del código, ya que el programador que vaya a utilizar un método no sabrá exactamente que excepción puede lanzar. También denota un problema conceptual ya que normalmente es debido a que el programador lanza excepciones de distintos tipos en un método, lo cual indica que seguramente deberíamos de tener varios métodos para hacer ese trabajo o bien que el programador no tiene claro que excepciones va a lanzar el método y deja la declaración más genérica posible.

10.6.5 Resultados del análisis de los proyectos correspondientes a cuarto curso

En este apartado describimos los resultados del análisis de los proyectos correspondientes a la asignatura de primer curso: Procesadores de lenguaje. En primer lugar hacemos una mínima descripción de la asignatura y las prácticas planteadas en ella. Incluimos una tabla resumen de los errores más frecuentes con la información descrita en el apartado 10.6.1 y una gráfica con la misma información generada por el prototipo PBA, por último hacemos un análisis de los resultados para este curso.

Descripción de la asignatura: Procesadores de Lenguaje

- El objetivo principal de esta asignatura es enseñar al alumno los problemas y técnicas que se plantean en la construcción de procesadores, traductores, compiladores e intérpretes de los distintos lenguajes de programación. El estudio de estas técnicas permite al alumno una visión más amplia de los lenguajes de programación, habitualmente estudiados desde el punto de vista del programador y no del constructor de compiladores o intérpretes.

- Situación dentro del plan de estudios. Esta asignatura se imparte en cuarto curso del segundo ciclo de Ingeniería en Informática y tiene una duración anual.

- Asignaturas de programación precedentes. Los alumnos cuando llegan a esta asignatura tienen un conocimiento amplio del lenguaje de programación empleado y de las técnicas de programación y depuración.

Tabla 20. Avisos más frecuentes para la asignatura de cuarto curso: Procesadores de Lenguaje

Código error	Descripcion	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13084	Avoid unused local variables.	64	3	21,33	60,00%
13080	The same String literal appears several times in this file.	50	4	12,50	80,00%
13081	Avoid instantiating String objects; this is usually unnecessary.	47	2	23,50	40,00%
13078	A method shouldn't have Exception in throws declaration.	34	3	11,33	60,00%
11003	Innecesary calls to methods.	33	2	16,50	40,00%
13039	Avoid unnecessary comparisons in boolean expressions	30	1	30,00	20,00%
12005	Value of referenced variable may be NULL.	23	1	23,00	20,00%
12001	Component in this class shadows one in base class.	21	3	7,00	60,00%
13000	Avoid empty catch blocks	20	2	10,00	40,00%
12014	Compare strings as object references.	18	2	9,00	40,00%

Número de proyectos	5

Prácticas que se proponen en la asignatura para su resolución

- Procesadores de Lenguaje. El proyecto consiste en desarrollar un compilador de un lenguaje sencillo que compile a un lenguaje intermedio. Este compilador se realiza con la ayuda de varias herramientas que permiten generar determinadas partes del compilador: analizador léxico y analizador sintáctico ascendente.

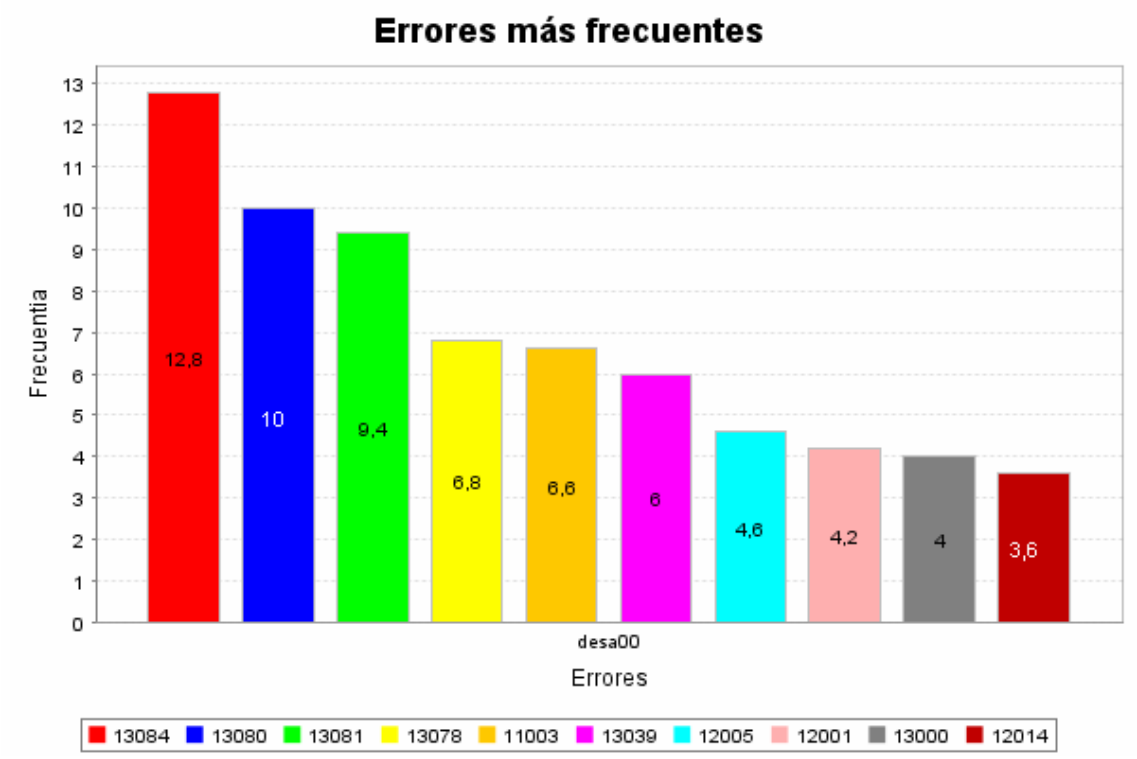


Figura 69. Avisos más frecuentes para el grupo de proyectos de la asignatura de cuarto curso.

Análisis de los avisos detectados con mayor frecuencia en los proyectos de este curso

Sólo hay un avisos que está presente en más del 75% de los proyectos: ***The same String literal appears several times in this file***. De nuevo nos encontramos con este problema que se hereda desde primer curso.

10.6.6 Resultados del análisis de los proyectos correspondientes al proyecto Fin de Carrera

En este apartado describimos los resultados del análisis de los proyectos correspondientes al proyecto fin de carrera. En primer lugar hacemos una mínima descripción de la asignatura y las prácticas planteadas en ella. Incluimos una tabla resumen de los errores más frecuentes con la información descrita en el apartado 10.6.1 y una gráfica con la misma información generada por el prototipo PBA, por último hacemos un análisis de los resultados para este curso.

Descripción del Proyecto Fin de Carrera

- Objetivo. Esta materia no consiste en una asignatura como tal, con clases teóricas y prácticas; sino que el estudiante tiene que realizar un proyecto autorizado e individual en el que aplique los conocimientos adquiridos en la carrera. Cada alumno tendrá su propio proyecto diferente al resto y se pretende que realice el ciclo completo en el proceso de desarrollo del software: análisis, diseño, implementación, pruebas, implantación.
- Situación dentro del plan de estudios. Este proyecto como su nombre indica se realiza al final de la carrera, se suele comenzar en el último cuatrimestre cuando el alumno ha visto la mayoría de las asignaturas de la carrera.
- Asignaturas de programación precedentes. El alumno ya ha visto todas las asignaturas relacionadas con la programación. Aunque debido a la diversidad de los proyectos puede tener que emplear técnicas o herramientas que no ha visto durante la carrera.

Objetivos que se proponen en los proyectos fin de carrera para su resolución

- Los trabajos que se proponen tienen una tipología muy variada; pero tienen en común que son proyectos reales o que podrían ser reales, y por tanto tienen un tamaño medio o grande donde el alumno tiene que trabajar desde la especificación de requisitos hasta el proceso de implantación en un entorno real.
- El lenguaje y entorno de desarrollo varían con el proyecto.

Para el estudio se han seleccionado proyectos con las siguientes características:

- La mayoría son aplicaciones Web con una interfaz de usuario realizada en HTML, JSP. Aunque una de ellas utiliza interfaz swing para realizar una aplicación de escritorio.
- Están realizados con el lenguaje Java, el entorno de desarrollo más utilizado es JBuilder, aunque alguno de ellos utiliza Eclipse o NetBeans.

Tabla 21. Avisos más frecuentes para el Proyecto Fin de Carrera

Código error	Descripcion	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13080	The same String literal appears several times in this file.	354	4	88,50	44,44%
13061	Avoid unused imports	220	1	220,00	11,11%
13000	Avoid empty catch blocks	128	2	64,00	22,22%
13084	Avoid unused local variables.	119	4	29,75	44,44%
13081	Avoid instantiating String objects; this is usually unnecessary.	116	4	29,00	44,44%
13044	Avoid calls to overridable methods during construction	110	2	55,00	22,22%
13011	Avoid returning from a finally block	98	3	32,67	33,33%
13039	Avoid unnecessary comparisons in boolean expressions	88	3	29,33	33,33%
13078	A method shouldn't have Exception in throws declaration.	65	1	65,00	11,11%
13042	Avoid reassigning parameters.	54	3	18,00	33,33%

Número total de proyectos	9

Análisis de los avisos detectados con mayor frecuencia en los proyectos de este curso

Ningún aviso alcanza ni el 50% de los proyectos, lo cual indica que los errores están muy repartidos entre los distintos proyectos.

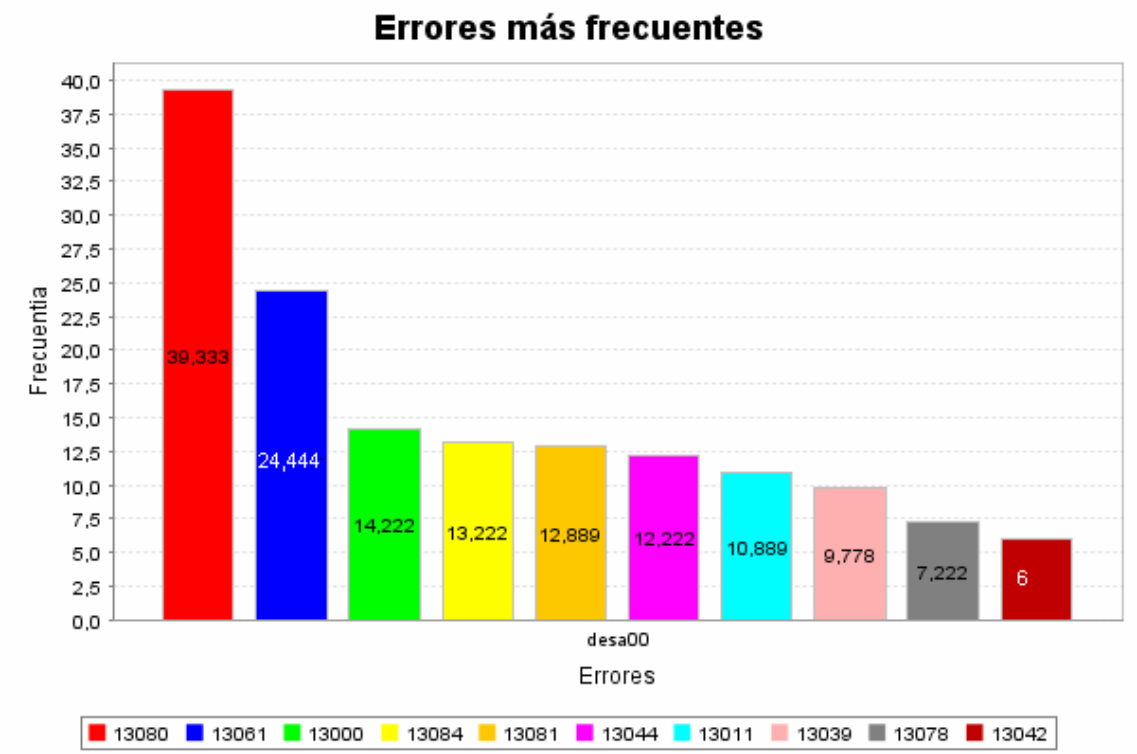


Figura 70. Avisos más frecuentes para el Proyecto Fin de Carrera

10.7 Comparación de los avisos y conclusiones

En la Tabla 22 listamos todos los avisos recogidos en las tablas previas de avisos frecuentes; pero ahora los ordenamos dependiendo del curso en el que aparecen para poder apreciar las diferencias. El número que aparece en las columnas de la derecha corresponde al número medio de errores en los proyectos, de ese curso, que contienen ese tipo de error. Analizando los datos de la tabla podemos considerar dos grandes clases de avisos:

- Los que aparecen en todos los cursos.
- Los que van evolucionando.

Tabla 22. Avisos encontrados en cada grupo de proyectos

Código error	Descripción del error	curso 1	curso 2	curso 3	curso 4	pfc
13080	The same String literal appears several times in this file.	4,53	7,84	6,53	12,50	88,50
13084	Avoid unused local variables.	4,29	6,83	8,69	21,33	29,75
13039	Avoid unnecessary comparisons in boolean expressions	8,11	8,30	3,00	16,50	29,33
13000	Avoid empty catch blocks	1,76	8,52	4,20	10,00	64,00
11004	Comparison of String objects using "==" or "!="	2,53				
13016	An empty statement (semicolon) not part of a loop	1,26				
13040	Switch statements should have a default label	2,72	5,04			
12002	Local variable shadows component of class.	5,87	6,43			
13044	Avoid calls to overridable methods during construction	9,45		6,06		55,00
12001	Component in this class shadows one in base class.	15,87			7,00	
12014	Compare strings as object references.	6,46			9,00	
11021	Unread field: should this field be static?		17,97			
13046	This final field could be made static		18,58			
13042	Avoid reassigning parameters.		5,03			18,00
10013	May be wrong assumption about ELSE branch association		6,14			
13078	A method should'nt have Exception in throws declaration.		13,86	17,93	11,25	65,00
13061	Avoid unused imports			4,46		220,00
13059	Avoid duplicate imports			3,17		
11011	Redundant comparison of a reference value to null.			12,50		
13083	Avoid unused private fields.			3,00		
11023	This field is never read. Consider removing it from the class.			8,69		
13081	Avoid instantiating String objects; this is usually unnecessary.				23,50	29,00
11003	Innecesary calls to methods.				11,33	
12005	Value of referenced variable may be NULL.				23,00	
13011	Avoid returning from a finally block					32,67
25,00	Total errores especificados	11,00	11,00	11,00	10,00	10,00

10.7.1 Errores que aparecen en todos los cursos

Existen una serie de errores que aparecen en todos los cursos analizados. Estos errores denotan dos tipos de problemas:

- Problemas a los que los docentes de las asignaturas no dan importancia, podría ser el caso de: ***The same String literal appears several times in this file***, ***Avoid***

unused local variables y ***Avoid unnecessary comparisons in boolean expressions***.

- Problemas que no se detectan nunca o en pocas ocasiones ante el poco tiempo para hacer una revisión del código de la práctica que existe actualmente. Es el caso de: ***Avoid empty catch blocks***. Además, este es un claro ejemplo de un problema en el que un análisis estático se muestra mucho más efectivo que la realización de pruebas.

10.7.2 Errores exclusivos de un curso

En la Tabla 22 aparecen algunos errores como exclusivos de un único curso. Los más significativos son:

- ***Comparison of String objects using "==" or "!="*** que aparece únicamente en primer curso. La media de errores por proyecto con errores es baja; sin embargo, este es un claro error conceptual a la hora de realizar el tratamiento de los objetos String. Parece claro que este problema se aborda por parte de los profesores de la asignatura, ya que en los siguientes cursos no aparece.
- ***This final field could be made static***, aparece únicamente en segundo. Este error parece evidenciar problemas en los conceptos de final y de static. En curso el concepto de static no se trata en profundidad es en segundo cuando aparece y se trata más ampliamente.
- ***Redundant comparison of a reference value to null***, aparece en tercer curso. Este error se detecta cuando hay una comparación entre referencias donde alguno de ellos es siempre null, normalmente porque se ha eliminado la referencia anteriormente. Por tanto, parece indicar que el programador ha cambiado código sin darse cuenta de los efectos sobre otra parte del código y por tanto seguramente habrá algún error lógico. Por otra parte, otros errores exclusivos de este curso están relacionados con campos no usados: ***Avoid unused private fields*** y ***This field is never read. Consider removing it from the class***. Esto tiene clara relación con la asignatura elegida en este tercer curso: base de datos. El tipo de proyectos que se realizan en esta asignatura, implica clases que se corresponden con tablas en la base de datos y por tanto habrá campos en cada clase por cada uno de los campos de la tabla, si posteriormente las operaciones que se piden para una tabla sólo implican operar con un campo los demás quedan sin utilizar.
- ***Value of referenced variable may be NULL***. Aparece en cuarto curso. Este error se produce cuando se utiliza una referencia que en determinadas condiciones del flujo de control se asigna a null. Por tanto, o hay determinadas condiciones del programa que nunca se ejecutan y por tanto se podrían eliminar o bien habría que comprobar si la referencia es null antes de utilizarla.

10.7.3 Errores que evolucionan con los cursos

Hay otro conjunto de errores que no aparecen en todos los cursos; pero tampoco son exclusivos de un único curso. Por ejemplo, ***Local variable shadows component of class***, aparece en primer y segundo cursos y luego desaparece.

10.7.4 Conclusiones finales

En este capítulo hemos abordado el estudio de los errores a partir de los análisis de los proyectos en distintos cursos de Ingeniería Informática.

Se ha realizado un estudio con un gran número de proyectos y además no se ha limitado a un único nivel sino que se han obtenido proyectos de cinco niveles de formación de los estudiantes.

En el análisis se han filtrado los mensajes relacionados con las convenciones de código y nombres que por un lado sólo se abordan directamente en alguna de las asignaturas y por otro lado no consideramos que puedan ser indicadores directos de errores.

Se ha utilizado el sistema propuesto en esta tesis: SICODE a través del subsistema PBA para obtener las tablas de los diez o doce errores que aparecen más frecuentemente en cada uno de los cursos y se proporcionan los datos de si los errores están concentrados en pocos proyectos o repartidos entre todos los proyectos y la media de errores de ese tipo en cada uno de los proyectos. Además, se incluye una gráfica de barras que permite comparar la frecuencia de cada uno de los errores por curso.

Por último, se ha elaborado una tabla global que permite comparar los errores en distintos cursos. En esta tabla hemos caracterizado errores dependiendo de los cursos en los que aparecen y su evolución: errores que aparecen en todos los cursos, errores que son exclusivos de un curso (en todos los cursos hay este tipo de errores) y errores que evolucionan con los cursos.

Este estudio proporciona datos que permitirán construir las bases para un modelo de usuario y de esta forma que el sistema se adapte a cada usuario dependiendo su perfil.

Capítulo 11. Conclusiones y trabajo futuro

En esta tesis se ha modelado y realizado un entorno de desarrollo Web que proporciona herramientas para la mejora de la calidad del código de los desarrolladores.

Para conseguir este objetivo se han analizado distintos sistemas tanto académicos como comerciales (Capítulo 3) revisando sus aportaciones a la mejora de la calidad del código. En dicha revisión, hay un gran número de aplicaciones académicas que se centran solamente en la descripción de los conceptos fundamentales de programación y además en el caso de entornos, están pensados también para un curso inicial. En la parte comercial las herramientas disponen de un gran número de opciones que les proporcionan gran potencia; pero está supeditada a la experiencia de los desarrolladores. En el caso concreto del tratamiento de errores, hay disponibles gran cantidad de herramientas que realizan análisis estático y que proporcionan una información mucho más profunda que el compilador sobre los errores del código fuente; sin embargo, la información que proporcionan prácticamente se limita al mensaje de error, si el desarrollador es la primera vez que trata este error tendrá difícil analizar sus causas y por tanto, corregir este error.

Además, se han establecido los requisitos generales que debería cumplir un sistema para la mejora de la calidad del código (Capítulo 4) y se han diseñado una serie de prototipos para refinar estos requisitos de forma iterativa (Capítulo 5 al Capítulo 9). Obteniendo un sistema que:

- Está centrado en el entorno de desarrollo que permite trabajar de forma colaborativa con proyectos de tamaño medio y grande.
- Se basa en técnicas de procesadores de lenguaje para realizar la captura y el análisis de errores sobre el código fuente de los proyectos que se están desarrollando.
- Proporciona una interfaz Web que permite la fácil integración con otros sistemas Web como sistemas de aprendizaje en el campo académico o sistemas Web de gestión de proyectos software; subsanando carencias actuales de estos sistemas: permitiendo el desarrollo de proyectos on-line; y un seguimiento continuo y personalizado de los desarrolladores mediante los datos de los errores capturados y analizados automáticamente.

Por último, se ha realizado un estudio empírico sobre la frecuencia de errores de los estudiantes de Ingeniería Informática en los distintos cursos de la titulación.

11.1 Sistema diseñado: SICODE

A continuación describimos las características más destacadas del sistema SICODE.

11.1.1 Entorno Web de desarrollo que integra los distintos subsistemas

El entorno de desarrollo (IDEWeb) es el centro del sistema. El entorno integra las distintas herramientas necesarias para realizar el desarrollo del software. Dispone de una arquitectura Web con lo que el desarrollador no necesita nada más que un navegador Web para poder programar, todo el procesamiento necesario y el almacenamiento de los archivos del proyecto se realiza en el servidor.

El entorno permite escribir y modificar código fuente, realizar la compilación y el análisis directamente. Muestra los errores resultantes de la compilación y facilita la corrección de los mismos al conectar directamente con la base de conocimientos. Además, recoge la información del Sistema de Análisis de Programas (PBA) y la muestra al usuario a través de avisos facilitando la repetición de errores a la hora de escribir el código fuente.

Para realizar la edición sobre el navegador Web se utiliza un campo de texto adicional. Para superar la limitación que supone esto, se ha desarrollado un editor que se implementa como un applet con características equiparables a un editor de un IDE. De esta forma obtenemos facilidades en la edición pero pudiendo usarlo desde el navegador Web.

11.1.2 Sistema de colaboración en el desarrollo de aplicaciones

El Sistema para la colaboración en el desarrollo de aplicaciones (COLLDEV) permite la comunicación y la colaboración entre varios desarrolladores y supervisores. Para ello permite la creación de grupos de trabajo que tienen asignados uno o más proyectos y la comunicación entre desarrolladores mediante mensajes asíncronos de distintos tipos.

Motiva a los desarrolladores a la toma conjunta de decisiones sobre el proyecto mediante sistemas de encuestas. Estas decisiones son tanto a nivel de código fuente, para corregir un error de programación; como a nivel de diseño para reparar errores conceptuales o hacer una refactorización que mejore el diseño para hacerlo más claro y mantenible.

11.1.3 Historia de trabajo de Proyectos compartidos

El sistema de colaboración también permite compartir los archivos del proyecto. Por un lado este sistema es imprescindible para construir un proyecto de mediana o gran entidad; además permite que todos los desarrolladores puedan leer el código fuente de cualquier clase del proyecto con lo que facilita la revisión de código y la corrección de errores de forma compartida.

A parte de compartir el proyecto, también se van guardando las sucesivas versiones de este para hacer posible el seguimiento del desarrollo del proyecto en etapas intermedias y no sólo en su etapa final; lo que permite explorar cuándo se realizaron determinadas modificaciones, cuándo se corrigieron determinados errores o cuándo se introdujeron errores nuevos.

11.1.4 Sistema de análisis de errores en programas

Sistema de análisis de errores de programas (PBA), se basa en varios procesadores de lenguaje que realizan un análisis estático sobre el código fuente de los proyectos en desarrollo. El sistema permite configurar de forma flexible la secuencia y las opciones de ejecución de estas herramientas mediante unidades de compilación escritas en XML y que se validan mediante XML-Schema.

El sistema crea una *historia de compilación* con el histórico de errores resultado de la captura de todos los análisis a lo largo del desarrollo de la aplicación. Esta historia de compilación

permite al sistema analizar no sólo la información de la compilación actual, sino también la información de anteriores compilaciones y de múltiples herramientas.

El sistema permite configurar los análisis para adaptarlos a lo que requiere cada usuario, descartando errores poco significativos o combinando los errores recogidos por varias herramientas. Además, permite realizar el análisis para un usuario individual o un grupo de usuarios. Y permite obtener información sobre errores frecuentes y la evolución de los errores en el desarrollo del proyecto. Por otra parte, presenta esta información al usuario de forma tanto numérica como gráfica. Esta información sirve además para seleccionar los avisos que le van a aparecer al usuario en la interfaz del entorno de desarrollo (IDEWeb).

11.1.5 Base de conocimientos colaborativa

Base de conocimientos que aporta información extra sobre los mensajes de error, a través de ejemplos y proporcionando información sobre las causas y posibles soluciones. Esto ayuda a los desarrolladores de dos formas: a entender y corregir el error y a aprender a como evitarlo para que no se vuelva a producir la próxima vez que escriban código.

Esta base de conocimientos está basada en un sistema Wiki que permite añadir contenido desde el propio navegador, sin herramientas especiales. Además, en el sistema los usuarios no crean las páginas desde cero sino que el propio sistema introduce la información básica y a partir de ella los usuarios introducen sus experiencias concretas.

La concepción del Wiki permite que todo el mundo introduzca contenido en el sistema y supone un cambio en la concepción respecto a muchos sistemas en los que únicamente un experto o un grupo de expertos reducido puede aportar conocimiento al sistema y todos los demás únicamente pueden consultar. En este nuevo planteamiento todos pueden aportar, con lo que la base de conocimientos se enriquece más rápidamente, y aunque pueda haber ciertos errores en el contenido estos pueden ser reparados rápidamente.

11.2 Clasificación de usuarios

Se ha llevado a cabo un estudio de los errores que suelen cometer los desarrolladores en el proceso de aprendizaje de la programación (Capítulo 10). El estudio ha sido amplio tanto por la cantidad de proyectos analizados (por ejemplo para segundo curso se han analizado 337 proyectos), como por el número de cursos (se han analizado proyectos de cuatro cursos más proyectos final de carrera); además no se han analizado clases aisladas o determinadas partes sino que se ha considerado el proyecto completo.

Se ha utilizado el sistema propuesto en esta tesis: SICODE a través del subsistema PBA para obtener las tablas de los diez o doce errores que aparecen más frecuentemente en cada uno de los cursos. Nos hemos centrado en los avisos que pueden incidir sobre la corrección del código fuente, su claridad, su facilidad de lectura y comprensión y la facilidad de mantenimiento.

Por último, se han caracterizado distintos tipos de errores dependiendo de los cursos en los que aparecen y su evolución: errores que aparecen en todos los cursos, errores que son exclusivos de un curso (en todos los cursos hay este tipo de errores) y errores que evolucionan con los cursos. Esto nos permitirá construir las bases para un modelo de usuario y que el sistema proporcione información adaptada dependiendo del tipo en el que se encuadre.

11.3 Principales aportaciones del trabajo

En esta tesis se ha conseguido implementar un sistema que cumple los requisitos establecidos en el Capítulo 4. De todas las características que se detallan en ese capítulo destacaremos cuatro.

11.3.1 Creación de un entorno de desarrollo integrado con una interfaz Web

Los sistemas que utilizan la Web para el aprendizaje de la programación raramente permiten al usuario interactuar con los programas, nuestro sistema no sólo permitirá interactuar al desarrollador para escribir y modificar código fuente sino que permitirá compilar y corregir errores.

Los entornos de desarrollo comerciales son locales, la única posibilidad de compartir información entre varios usuarios es la de conectarse a un repositorio donde se compartan los archivos del proyecto; pero los entornos no integran posibilidades de comunicación. El sistema SICODE permite compartir los archivos del proyecto, incluye un sistema de intercambio de mensajes y toma de decisiones, guarda un registro con todos los errores de compilación del sistema y además permite intercambiar conocimientos entre los desarrolladores mediante la base de conocimientos colaborativa.

Existen sistemas de gestión de proyectos por ejemplo SourceForge que están basados en la Web y a través de ella proporcionan múltiples servicios a los desarrolladores del proyecto; sin embargo no llegan tan lejos como SICODE, ya que la escritura de código siempre se realiza en un sistema local.

SICODE a través de su entorno Web permite una independencia total de la plataforma y evita la instalación y configuración del sistema integrando a la vez las herramientas esenciales para el desarrollo de la aplicación. Esta interfaz Web permite además al sistema integrarse con otros sistemas Web como sistemas de e-learning o de gestión de proyectos.

11.3.2 Diseño de una historia de compilación

Para asegurar la corrección de los programas en los procesos de desarrollo tradicionales se utilizan varias técnicas:

- Arreglar los errores que proporciona el compilador, muchas veces esto es insuficiente ya que la principal tarea del compilador es generar código objeto y no detectar posibles problemas en el código.
- Realizar un conjunto de pruebas lo más amplio posible y corregir los problemas encontrados, este proceso lleva bastante tiempo ya que tenemos que crear casos de prueba y ejecutarlos.
- Una técnica que siempre se recomienda es la inspección de código; pero de nuevo necesitamos bastante tiempo para llevarla a cabo.

El sistema SICODE permite compilar y hacer un análisis exhaustivo del código fuente del proyecto que está en desarrollo, para ello se basa en herramientas que realizan este análisis estático, se combinan sus resultados y se almacenan en un registro de errores que denominamos "Historia de compilación". De esta forma, el sistema no sólo proporciona información sobre la última compilación, sino que produce avisos en función de esta historia de compilación en la que se pueden buscar errores frecuentes y una evolución de los errores para proporcionar al desarrollador las herramientas para evitar nuevos errores cuando esté escribiendo más código.

11.3.3 Utilización del análisis activo de errores para obtener una mejora en el código fuente

Los entornos de desarrollo actuales no están pensados para que los desarrolladores mejoren su estilo de programación y la calidad del código que escriben. Sin embargo, constantemente están apareciendo nuevas versiones de herramientas, nuevas técnicas y nuevos métodos de trabajo que proporcionan ventajas sobre los anteriores y que es imprescindible no sólo conocer, sino aplicar de la forma más adecuada. Los desarrolladores tienen que establecer procesos de formación continua, que son paralelos al proceso de desarrollo, para permanecer al día.

Por otra parte, algunas aplicaciones tienen una serie de requisitos de aceptación en los que incluyen pruebas de aceptación para comprobar que las aplicaciones tienen la funcionalidad requerida; pero también para comprobar que el diseño y el código fuente siguen unas pautas previamente marcadas. Por otra parte, muchos equipos de desarrollo establecen una guía de estilo para la escritura de código donde se incluyen convenios y buenas prácticas en el diseño y la codificación de las aplicaciones. Sin embargo, el cumplimiento de estas normas es muchas veces difícil de llevar a cabo y de comprobar.

En nuestro sistema se propone integrar el desarrollo con el aprendizaje, basándonos en la comprobación automática del código que está escribiendo el desarrollador. La forma de realizar esto conlleva el trabajo conjunto de varios elementos del sistema:

- Análisis exhaustivo del código fuente mediante el subsistema PBA, permite detectar que el código fuente no cumple determinada norma de la guía de estilo, o no se está empleando correctamente una técnica recogida en el grupo de desarrollo o no se cumple un requisito recogido en las pruebas de aceptación.
- A partir de las comprobaciones del subsistema PBA se pueden generar mensajes de aviso y mostrárselos al desarrollador a través del entorno Web (IDEWeb). Estos avisos indicarán errores concretos en el código; pero también facetas más generales del código (estilo, nuevas técnicas) que debería mejorar el programador.
- Los mensajes de aviso aparecerán mientras el desarrollador está escribiendo código. El sistema es activo y no espera a la fase de compilación para notificar los problemas.
- A través de los avisos el usuario puede consultar la base de conocimientos colaborativa que recoge la experiencia de otros desarrolladores; a partir de esta experiencia el desarrollador puede "aprender" a hacer las cosas mejor.

De esta forma, el sistema va desde la notificación de un error concreto en el código fuente a la información más general sobre el estilo de programación y técnicas de desarrollo que podrán servir para que futuros desarrollos tengan una mayor calidad.

11.3.4 Diseño de un modelo que permite realimentarse con la experiencia de los desarrolladores

La base de conocimientos donde se guarda información sobre la experiencia de los desarrolladores para entender y solucionar errores, no sigue un modelo estático; sino que está diseñada para que los desarrolladores sigan introduciendo más información de forma colaborativa mientras trabajan en las aplicaciones. De esta forma la base de conocimientos cada vez se hace más completa y refinada gracias a las aportaciones de los usuarios.

11.3.5 Realización de un estudio de los errores de programación y su evolución a lo largo de los distintos cursos académicos

Como ya se ha comentado anteriormente en esta tesis se realiza un amplio estudio que ha permitido conocer los tipos de errores que aparecen en cada uno de los cuatro cursos más el proyecto fin de carrera de la titulación de Ingeniería en Informática. Pero sobre todo se ha estudiado la evolución de esos errores a lo largo de la carrera, obteniendo una relación entre los errores y la experiencia del estudiante. Hasta ahora los estudios publicados se centraban en una práctica concreta o, los más amplios, en una asignatura; pero no se había trabajado sobre varias asignaturas para estudiar la evolución de los usuarios.

11.4 Futuras líneas de investigación

El trabajo realizado en esta tesis abre nuevas líneas de investigación relacionadas con mejoras en el sistema o nuevas aplicaciones del mismo.

11.4.1 Reducción de la granularidad en la comprobación y generación de avisos de ayuda al desarrollador

Se pretende seguir trabajando en un sistema activo que permita ir guiando al usuario en la mejora del estilo y la calidad del código. Para ello pretendemos trabajar en dos líneas:

- Realizar las comprobaciones del código fuente y emitir los avisos derivados de estas en tiempo real.
- Añadir a los avisos un mecanismo que permita automatizar la corrección de errores.

Esto evitará que errores cometidos permanezcan mucho tiempo sin corregirse ya que el sistema podrá avisar y hacer propuestas de corrección para que el usuario las utilice inmediatamente.

11.4.2 Descentralización del sistema

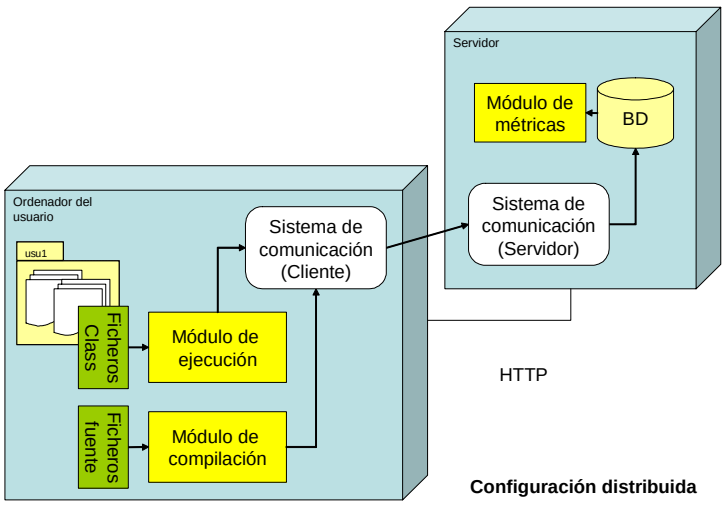


Figura 71. Arquitectura distribuida para el entorno de desarrollo y el sistema de análisis

La arquitectura planteada para el sistema ha sido una arquitectura Web centralizada en la que todo el procesamiento se realizaba en el servidor y los usuarios tienen una interfaz única a través del navegador. El planteamiento de esta línea de investigación consiste en

mantener la arquitectura Web; pero descentralizando el entorno de desarrollo y el sistema de análisis.

En este planteamiento el usuario utilizará un entorno de desarrollo instalado localmente; puede utilizar un entorno comercial en el que dispondrá de facilidades en la escritura del código. Los módulos de análisis se integrarán como plug-ins en el entorno para poder hacer su uso transparente al desarrollador. La base de datos y el módulo de generación de estadísticas permanecerán en un servidor centralizado y la comunicación se realizará mediante servicios Web sobre el protocolo HTTP.

Para realizar esto hay que abordar la creación de un vocabulario XML que permita expresar de forma genérica la información de análisis de errores y otro para poder expresar la información de las estadísticas en forma de avisos. De esta forma la información se transmitirá en dos direcciones: del cliente a la base de datos centralizada y del módulo de estadísticas en el servidor a cada cliente para presentar al desarrollador los avisos en su entorno.

11.4.3 Aplicación de técnicas de minería de datos para mejorar el análisis

El subsistema PBA ya dispone de ficheros de configuración que permiten realizar análisis filtrando determinados datos. Se podrían ampliar estos ficheros para utilizarlos en técnicas de minería de datos que se aplicarían sobre la historia de compilación; para obtener un mayor conocimiento sobre la calidad del código y dónde se debería mejorar.

11.4.4 Potenciación del análisis dinámico

En los prototipos implementados del sistema SICODE se realizaba fundamentalmente un análisis estático del código fuente para encontrar errores. Sin embargo, ya se dejaba abierta la posibilidad de ejecutar el código generado para realizar análisis dinámicos que complementen a los primeros (apartado 7.4.4).

Por tanto sería interesante, potenciar este módulo integrándolo con herramientas de pruebas como JUnit y que el sistema proporcionase ayuda al usuario para la creación de un conjunto de pruebas adecuado.

11.4.5 Potenciar el uso de la historia de trabajo

La historia de trabajo es una herramienta que puede proporcionar mucha información. Actualmente el sistema almacena la información de las versiones, proporciona una forma de navegación y permite realizar comparaciones. Sin embargo, este mecanismo tiene un gran potencial y permitiría hacer cosas como simulaciones de qué pasaría si en esta versión un fragmento se hubiese programado de otra manera.

11.4.6 Mejora de la adaptabilidad del sistema a los usuarios

El trabajo realizado en esta tesis pone las bases hacia la construcción de un modelo de usuario consistente. El sistema debería implementar este modelo de usuario para conseguir una adaptabilidad a cada usuario que interactúe con el sistema.

11.4.7 Aplicación del sistema para automatizar la comprobación de requisitos de aceptación

Actualmente los clientes de productos software no se conforman con que el producto cumpla una determinada funcionalidad, lo cual se puede comprobar con pruebas de aceptación. Además, el código fuente del producto debería cumplir unos niveles mínimos

de calidad que faciliten su modificación para la corrección de problemas que se encuentren en un futuro y para la incorporación de nuevas funcionalidades. Actualmente esta segunda comprobación se realiza normalmente mediante inspecciones de código manuales basadas en una guía de estilo, esto tiene dos problemas: es difícil la revisión de todo el código y se invierte mucho tiempo en realizar esta tarea.

El sistema SICODE ya realiza automáticamente muchas de las comprobaciones que puede contener una guía de estilo convencional, por otra parte habría que añadir una forma de añadir reglas de comprobación de otras características propias de cada producto.

Apéndice A. Archivos configuración Sistema de Análisis de Errores en Programas

En este apéndice incluimos varios archivos de configuración utilizados por el Sistema de Análisis de Errores en Programas.

A.1 Archivo de configuración unidad de compilación simple

El Sistema de análisis de errores en programas (descrito en el Capítulo 7. Sistema de análisis de errores de programas: PBA) utiliza determinados archivos para configurar Unidades de Compilación. Una Unidad de Compilación, permite indicar al sistema los directorios donde se encuentran los archivos fuente y donde tiene que dejar los resultados; las herramientas que debe utilizar, las opciones de estas herramientas y en que orden debe utilizarlas; incluso permite condicionar el uso de determinadas herramientas a que la ejecución de otras proporcione determinados resultados. El formato de estos archivos es XML del tipo de los archivos build.xml de la herramienta Apache Ant [Ant 2003].

En este apartado incluimos un archivo que ejecuta el compilador estándar de Java, concretamente el incluido en el JSDK 1.4.2 y una aplicación que se encarga de analizar los resultados y guardarlos en la base de datos del Sistema de análisis de errores en programas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="miproyecto" default="compilar" basedir=".">
    <description>
        Nombre     : Sistema de compilacion Java
        Versión    : 1.0
       Parametros : -Druta="ruta del proyecto"
                    -Dusuario="usuario que lanza la compilacion"
                    -Dcompilacion="codigo de la compilacion"
    </description>

    <!-- Propiedades generales del sistema -->
    <property file="path.properties"/>
    <!-- Directorio temporal -->
    <property name="tmp" location="${usuarios}/${usuario}/tmp"/>
    <!-- Directorio donde se guarda informacion temporal en esta compilacion -->
    <property name="tmp.log" location="${tmp}/log${compilacion}"/>
    <!-- Directorio donde se guardan las fuentes del proyecto -->
    <property name="ruta.src" location="${ruta}/src"/>

    <!--Definiciones de las ampliaciones que vamos a usar-->
    <taskdef name="myjavac" classname="proyecto.utilidad.ant.JavacTask"/>
    <taskdef name="analizador"
        classname="proyecto.utilidad.ant.AnalizadorTask"/>
```

```
    <target name="inicio" description="Inicializaciones necesarias">
        <!-- Creamos el directorio temporal -->
       <mkdir dir="${tmp}"/>
       <mkdir dir="${tmp.log}"/>
    </target>

    <target name="compilar" depends="inicio"
        description="Compilamos los ficheros fuente">
        <myjavac output="${tmp.log}/javac.log" debug="true"
            home="${javac.home}">
            <src path="${ruta.src}"/>
            <include name="**/*.java"/>
        </myjavac>
        <analizador herramienta="javac-sdk-1.4.2"
                    fichero="${tmp.log}/javac.log"
                    filtro="false"
                    usuario="${usuario}"
                    compilacion="${compilacion}"
                    property="errores"/>
    </target>

</project>
```

A.2 Archivo unidad de compilación completa

Se pueden crear tantas unidades de compilación como se crean convenientes y cada una tiene un identificador único con lo que se pueden utilizar como bloque que unifica todo el trabajo que debe realizar el sistema sobre el conjunto de archivos de un proyecto.

En este apartado incluimos un archivo que ejecuta el compilador estándar de Java y varias herramientas de análisis estático de errores. Las herramientas utilizadas se describen en el apartado (Apartado 7.6. Herramientas de búsqueda de errores utilizadas en el proyecto). Una vez que las herramientas han terminado de realizar su tarea, una aplicación que se encarga de analizar los resultados y guardarlos en la base de datos del Sistema de análisis de errores en programas.

Esta Unidad de Compilación es la que se ha utilizado para analizar y capturar los avisos del (Capítulo 10. Clasificación de usuarios basada en la detección de errores).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="completa" default="busqueda_errores" basedir=".">
    <description>
        Nombre     : Sistema de compilacion Java
        Versión    : 1.0
       Parametros : -Druta="ruta del proyecto"
                    -Dusuario="usuario que lanza la compilacion"
                    -Dcompilacion="codigo de la compilacion"
    </description>

    <!-- Propiedades generales del sistema -->
    <property file="path.properties"/>
    <!-- Directorio temporal -->
    <property name="tmp" location="${usuarios}/${usuario}/tmp"/>
    <!-- Directorio donde se guarda informacion temporal en esta compilacion -->
    <property name="tmp.log" location="${tmp}/log${compilacion}"/>
    <!-- Directorio donde se guardan las fuentes del proyecto -->
    <property name="ruta.src" location="${ruta}/src"/>

    <!--Definiciones de las ampliaciones que vamos a usar-->
    <taskdef resource="net/sf/antcontrib/antcontrib.properties"/>
    <taskdef name="findbugs"
        classname="edu.umd.cs.findbugs.anttask.FindBugsTask"/>
    <taskdef name="myjavac" classname="proyecto.utilidad.ant.JavacTask"/>
    <taskdef name="analizador"
        classname="proyecto.utilidad.ant.AnalizadorTask"/>
    <taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"/>
```

```
    <taskdef name="antic" classname="proyecto.utilidad.ant.AnticTask"/>
    <taskdef name="jlint" classname="proyecto.utilidad.ant.JlintTask"/>

    <target name="inicio" description="Inicializaciones necesarias">
        <!-- Creamos el directorio temporal -->
       <mkdir dir="${tmp}"/>
       <mkdir dir="${tmp.log}"/>
    </target>

    <target name="compilar" depends="inicio"
        description="Compilamos los ficheros fuente">
        <echo message="" file="${tmp.log}/javac.log"/>
        <myjavac output="${tmp.log}/javac.log" debug="true"
            home="${javac.home}">
            <src path="${ruta.src}"/>
            <include name="**/*.java"/>
        </myjavac>
        <analizador herramienta="javac-sdk-1.4.2"
                    fichero="${tmp.log}/javac.log"
                    filtro="false"
                    usuario="${usuario}"
                    compilacion="${compilacion}"
                    failOnError="true"
                    property="errores"/>
    </target>

    <target name="busqueda_errores" depends="inicio,compilar"
        description="Buscamos en el cógigo errores potenciales">
    <if>
        <equals arg1="${errores}" arg2="0" casesensitive="false"/>
        <then>
            <parallel>
                <sequential>
                    <echo message="" file="${tmp.log}/jlint.log"/>
                    <jlint home="${jlint.home}" output="${tmp.log}/jlint.log"
                           omite="synchronization" failOnError="true">
                        <src path="${ruta.src}"/>
                        <include name="**/*.class"/>
                    </jlint>
                    <analizador herramienta="jlint"
                                fichero="${tmp.log}/jlint.log"
                                usuario="${usuario}"
                                compilacion="${compilacion}"/>
                </sequential>
            </parallel>
            <parallel>
                <sequential>
                    <echo message="" file="${tmp.log}/antic.log"/>
                    <antic home="${antic.home}" output="${tmp.log}/antic.log"
                        java= "true" failOnError="true">
                        <src path="${ruta.src}"/>
                        <include name="**/*.java"/>
                    </antic>
                    <analizador herramienta="antic"
                                fichero="${tmp.log}/antic.log"
                                usuario="${usuario}"
                                compilacion="${compilacion}"/>
                </sequential>
            </parallel>
            <parallel>
                <sequential>
                    <echo message="" file="${tmp.log}/findbugs.log"/>
                    <findbugs home="${findbugs.home}" output="emacs"
                        outputfile="${tmp.log}/findbugs.log"

omitVisitors="DontCatchIllegalMonitorStateException,FindInconsistentSync2,FindJSR
166LockMonitorenter,FindMismatchedWaitOrNotify,FindNakedNotify,FindReturnRef,Find
RunInvocations,FindSpinLoop,FindTwoLockWait,FindUnconditionalWait,FindUnreleasedL
ock,FindUnsyncGet,InitializationChain,LazyInit,LockedFields,MutableLock,MutableSt
aticFields,SerializableIdiom,StartInConstructor,SwitchFallthrough,WaitInLoop,Find
InconsistentSync2"
                        timeout="60000">
                        <sourcepath path="${ruta.src}"/>
                        <class location="${ruta.src}"/>
                    </findbugs>
                    <analizador herramienta="findbugs-0.7.3"
                                fichero="${tmp.log}/findbugs.log"
```

```
                                usuario="${usuario}"
                                compilacion="${compilacion}"/>
                </sequential>
            </parallel>
            <parallel>
                <sequential>
                    <echo message="" file="${tmp.log}/pmd.log"/>
                    <pmd
rulesetfiles="${pmd.home}\rulesets\reglas_pmd_predefinidas.xml">
                        <formatter toFile="${tmp.log}/pmd.log"
                            type="net.sourceforge.pmd.renderers.EmacsRenderer"/>
                        <fileset dir="${ruta.src}">
                            <include name="**/*.java"/>
                        </fileset>
                    </pmd>
                    <analizador herramienta="pmd-1.6"
                                fichero="${tmp.log}/pmd.log"
                                usuario="${usuario}"
                                compilacion="${compilacion}"/>
                </sequential>
            </parallel>
        </then>
        <else>
            <echo message="Numero de errores de compilacion = ${errores}"/>
        </else>
    </if>
    </target>

</project>
```

A.3 Esquema XML para validación de los archivos de configuración de estadísticas

Este archivo permite validar cualquier archivo de configuración de estadísticas mediante cualquier parser XML que soporte XML-Schema. En el apartado 7.7 se describen todas las posibilidades que permite este XML-Schema.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
    Document   : estadisticas.xsd
    Description:
        XML Schema de los ficheros de estadísticas.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Definición del elemento "estadisticas" -->
<!-- Puede tener anidados uno o más elementos que representan tipos
     de estadisticas. Cada uno de estos tipos de estadísticas están
     representados por un elemento distinto.
     Para el proyecto se concivieron dos tipos de estadísticas distintas:
        - Errores más frecuentes.
        - Media de errores.
     Pero pueden ser ampliables en un futuro
-->
    <xs:element name="estadisticas">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="ErroresFrecuentes"/>
                <xs:element ref="MediaErrores"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>

<!-- Definición de los tipos de estadísticas -->
<!-- Definición de la estadística de "Errores más frecuentes".
     Esta posee un atributo "numero" que representa el número de errores
     más frecuentes que se desean mostrar, por defecto es 5.
     Su otro atributo "manejador" indica la clase que maneja esta estadistica.
```

```
     El valor por defecto es "proyecto.estadisticas.ErroresFrecuentes", pero
     puede ser cambiada por los usuarios.
-->
    <xs:element name="ErroresFrecuentes">
        <xs:complexType>
            <xs:group ref="orden_estadistica"/>
            <xs:attribute ref="numero"/>
            <xs:attribute ref="manejador"
                default="proyecto.estadisticas.ErroresFrecuentes"/>
        </xs:complexType>
    </xs:element>

    <xs:attribute name="numero" default="5">
        <xs:simpleType>
            <xs:restriction base="xs:positiveInteger">
                <xs:maxInclusive value="10"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>

<!-- Definición de la estadística de "Media de Errores".
     Esta posee un atributo "evolucion" que representa el número de periodos
     para los que se desea conocer este valor, por defecto es 1.
     Su otro atributo "manejador" indica la clase que maneja esta estadistica.
     El valor por defecto es "proyecto.estadisticas.MediaErrores", pero puede
     ser cambiada por los usuarios.
-->
    <xs:element name="MediaErrores">
        <xs:complexType>
            <xs:group ref="orden_estadistica"/>
            <xs:attribute ref="evolucion"/>
            <xs:attribute ref="manejador"
                default="proyecto.estadisticas.MediaErrores"/>
        </xs:complexType>
    </xs:element>

    <xs:attribute name="evolucion" type="xs:positiveInteger" default="1"/>

<!-- El atributo "manejador" tiene como misión indicar cual es el manejador
     de las estadisticas.
-->
    <xs:attribute name="manejador" type="xs:string"/>

<!-- Todos los tipos de estadisticas tendrán una estructura base:
        - Periodo       (una aparición)
        - Usuarios      (una aparición)
        - Errores       (una o más apariciones)
        - Compilaciones (0 o 1 apariciones)
-->
    <xs:group name="orden_estadistica">
        <xs:sequence>
            <xs:element ref="periodo"/>
            <xs:element ref="usuarios"/>
            <xs:element ref="errores" maxOccurs="unbounded"/>
            <xs:element ref="compilaciones" minOccurs="0"/>
        </xs:sequence>
    </xs:group>

<!-- Definición de los periodos -->
<!-- Habrá tres formas posibles de indicar los periodos:
        - Mediante un intervalo temporal relativo (unidad dia).
            El final es hoy menos los dias que se indiquen como retardo.
        - Mediante un intervalo temporal absoluto (unidad dia).
            El final es la fecha indicada. El formato de la fecha es:
                AAAA-MM-DD (año - mes - día)
        - Mediante un intervalo de compilaciones absoluto (unidad compilacion).
            El final es la compilación indicada.
     Siempre habrá que especificar la duración del periodo, cuya unidad será la
     que se use en el inicio.
     El inicio de un periodo de final "f" y de duración "d" será "f-d"
-->
    <xs:element name="periodo">
        <xs:complexType>
            <xs:sequence>
                <xs:choice>
                    <xs:element ref="tiempo"/>
                    <xs:element ref="compilacion"/>
```

```
                </xs:choice>
                <xs:element ref="duracion"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="tiempo">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="relativo"/>
                <xs:element ref="fecha"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>

    <xs:element name="compilacion" type="xs:nonNegativeInteger"/>
    <xs:element name="duracion" type="xs:nonNegativeInteger" default="1"/>

    <xs:element name="relativo" type="xs:nonNegativeInteger" default="0"/>
    <xs:element name="fecha" type="xs:date"/>

<!-- Definicion del conjunto de usuarios -->
<!-- El elemento usuarios sirve para indicar los usuarios para los cuales se
     va a realizar el estudio estadistico.
     La forma de inclurlos es indicando dentro de elemento anidado usuario
     el código de usuario que se desee.
     El elemento usuarios tiene dos atributos booleanos:
        - "autor". Se usa para indicar que se haga la estadística para
                   el usuario que lanza el estudio estadístico.
                   Por defecto es "true".
        - "todos". Se usa para indicar que se haga la estadítica a todos los
                   usuarios del sistema.
                   Por defecto es "false".
-->
    <xs:element name="usuarios">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="usuario" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute ref="autor"/>
            <xs:attribute ref="todos"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="usuario" type="xs:string"/>

    <xs:attribute name="autor" type="xs:boolean" default="true"/>
    <xs:attribute name="todos" type="xs:boolean" default="false"/>

<!-- Definición de los conjuntos de errores -->
<!-- Se puede indicar que errores se quere usar en la muestra de estadísticas.
     En primer lugar se pueden meter condiciones de dos tipos:
        - elementos "incluir" - usar los errores que cumplan una condición.
        - elementos "excluir" - usar los errores que no cumplan una condición.
     Si es expecifica más de una condición se deberán indicar como se relacionan
     esas condiciones, para ello se emplean los elementos "and" y "or".
     Las condiciones pueden ser de tres tipos:
        - "codigo".      Donde se indica el código del error.
        - "tipo".        Donde se indica un tipo de errores.
        - "herramienta". Donde se indica la herramienta que generó los errores.
-->
    <xs:element name="errores">
        <xs:complexType>
            <xs:group ref="condiciones_error" minOccurs="0"/>
        </xs:complexType>
    </xs:element>

    <xs:group name="condiciones_error">
        <xs:sequence>
            <xs:choice>
                <xs:element ref="incluir"/>
                <xs:element ref="excluir"/>
            </xs:choice>
            <xs:group ref="mas_condiciones" minOccurs="0"
               maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:group>
```

```
    <xs:group name="mas_condiciones">
        <xs:sequence>
            <xs:choice>
                <xs:element name="and"/>
                <xs:element name="or"/>
            </xs:choice>
            <xs:choice>
                <xs:element ref="incluir"/>
                <xs:element ref="excluir"/>
            </xs:choice>
        </xs:sequence>
    </xs:group>

    <xs:element name="incluir">
        <xs:complexType>
            <xs:group ref="tipos_error"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="excluir">
        <xs:complexType>
            <xs:group ref="tipos_error"/>
        </xs:complexType>
    </xs:element>

    <xs:group name="tipos_error">
        <xs:choice>
            <xs:element ref="codigo"/>
            <xs:element ref="tipo"/>
            <xs:element ref="herramienta"/>
        </xs:choice>
    </xs:group>

    <xs:element name="codigo" type="xs:integer"/>
    <xs:element name="tipo" type="xs:string"/>
    <xs:element name="herramienta" type="xs:string"/>

<!-- Definición de los conjuntos de compilaciones -->
<!-- Se puede indicar que unidades de compilación se quieren usar a la hora de
     hacer las estadísticas.
-->
    <xs:element name="compilaciones">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="unidad_compilacion" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="unidad_compilacion" type="xs:string"/>

</xs:schema>
```

A.4 Archivo ejemplo de configuración de estadísticas

Este archivo permite configurar la información incluida en un informe de resultados de análisis (Ver apartado 7.7. Estadísticas).

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--
    Document   : ejemplo.xml
    Description:
        Ejemplo de como hacer los ficheros de estadísticas
-->

<estadisticas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="estadisticas.xsd">
```

```
    <ErroresFrecuentes numero="4">
        <periodo>
            <tiempo>
                <relativo>5</relativo>
            </tiempo>
            <duracion>14</duracion>
        </periodo>
        <usuarios todos="true"/>
        <errores>
            <excluir>
                <tipo>error de compilación</tipo>
            </excluir>
        </errores>
        <compilaciones>
            <unidad_compilacion>completa</unidad_compilacion>
        </compilaciones>
    </ErroresFrecuentes>

    <MediaErrores evolucion="5">
        <periodo>
            <compilacion>1000</compilacion>
            <duracion>200</duracion>
        </periodo>
        <usuarios autor="false" todos="true"/>
        <errores>
            <incluir>
                <herramienta>pmd-1.6</herramienta>
            </incluir>
        </errores>
        <errores/>
    </MediaErrores>

    <MediaErrores>
        <periodo>
            <tiempo>
                <fecha>2004-09-17</fecha>
            </tiempo>
            <duracion>7</duracion>
        </periodo>
        <usuarios todos="true"/>
        <errores>
            <excluir>
                <herramienta>jlint</herramienta>
            </excluir>
        </errores>
        <errores>
            <incluir>
                <herramienta>jlint</herramienta>
            </incluir>
            <or/>
            <incluir>
                <tipo>aviso de sincronización</tipo>
            </incluir>
        </errores>
    </MediaErrores>

    <ErroresFrecuentes numero="8">
        <periodo>
            <compilacion>10</compilacion>
            <duracion>10</duracion>
        </periodo>
        <usuarios/>
        <errores/>
    </ErroresFrecuentes>

    <ErroresFrecuentes numero="5">
        <periodo>
            <compilacion>10</compilacion>
            <duracion>10</duracion>
        </periodo>
        <usuarios/>
        <errores>
            <incluir>
                <codigo>13019</codigo>
            </incluir>
            <or/>
            <incluir>
```

```
                <codigo>13000</codigo>
            </incluir>
        </errores>
    </ErroresFrecuentes>
</estadisticas>
```

Apéndice B. Información sobre los errores frecuentes

En este capítulo se incluye información detallada sobre los errores más relevantes encontrados en el Capítulo 10 de los errores más frecuentes en los distintos cursos de la titulación de Ingeniería en Informática.

A.1 Errores que aparecen en todos los cursos

Evitar literales duplicados (13080)

Tipo de aviso:

Herramienta que detecta este aviso: PMD

El código que contiene literales String duplicados puede mejorarse declarando un String como campo constante.

Ejemplo:

The same String literal appears 8 times in this file; the first occurrence is on line 38

The same String literal appears 8 times in this file; the first occurrence is on line 41

```
…
public void escribir(int a[])
{
       escribir("{");
      for(int i=0; i<a.length; i++)
       escribir(a[i] + (i!=a.length-1 ? ", " : ""));
      escribir("}");
}

public void escribir(char a[])
{
      escribir("{");
      for(int i=0; i<a.length; i++)
       escribir(a[i] + (i!=a.length-1 ? ", " : ""));
      escribir("}");
}

public void escribir(long a[])
{
      escribir("{");
      for(int i=0; i<a.length; i++)
          escribir(a[i] + (i!=a.length-1 ? ", " : ""));
      escribir("}");
}
…
```

Evitar variables locales que no se utilizan (13084)

Tipo de aviso: Aviso de código no usado. Aviso sobre zonas de código no usadas en el programa.

Herramienta que detecta este aviso: PMD

Quitan claridad a la hora de leer el código y dificultan su mantenimiento.

Ejemplo:

```
Avoid unused local variables such as 'juegoRol'
public Principal() {
       realizarPruebas();
       Juego juegoRol= new Juego();
}
```

Ejemplo2:

Avoid unused local variables such as 'destreza'

Avoid unused local variables such as 'fuerza'

Avoid unused local variables such as 'magia'

Avoid unused local variables such as 'tipo'

Avoid unused local variables such as 'ataque'

```
public void arma(int i) {
       int opc, opcArma, fuerza, destreza, magia, n;
       String tipo;
      boolean ataque;
      p.escribirnl("\tTienes "+jugador.getnArmas()+" armas");
      p.escribirnl("\t\tPulsa 1 si quieres recogerla u otro NUMERO para
dejarla");
      opc = t.leerEntero();
      if(opc == 1) {
       if(jugador.getnArmas() >= 3) {
                        p.escribirnl("Tienes que dejar un arma para coger la
nueva");
                        n = mostrarArmas('x');
                        do {
                               p.escribirnl("\nEscribe el numero de arma que
deseas borrar:");
                               opcArma = t.leerEntero();
                        }while((opcArma > n)||(opcArma <= 0));
                        jugador.borrar(opcArma-1);
       }
       jugador.push((Arma)aventura.getSuceso(i));
      }
}
```

Evitar comparaciones innecesarias de expresiones booleanas (13039)

Tipo de aviso: Aviso de código innecesario. Aviso sobre zonas del código que son innecesarias o redundantes.

Herramienta que detecta este aviso: PMD

Se debería evitar el uso de operaciones de igualdad con operandos booleanos. Nunca se deberían utilizar los literales true o false en cláusulas condicionales. Esto permite mejorar la legibilidad del programa simplificando la expresión.

Ejemplo:

Avoid unnecessary comparisons in boolean expressions

```
    boolean insertado = false;
    for(int i = 0;i < 3; i++){
      if (inventario[i] == null && insertado == false){
          inventario[i] = objeto;
          insertado = true;
        }
    }
```

Evitar los bloques "catch" vacíos (13000)

Tipo de aviso: Aviso de estructura de control. Aviso de un posible error en una estructura de control de flujo del programa.

Herramienta que detecta este aviso: PMD

Ejemplo:

Avoid empty catch blocks

```
public void cerrarFlujo(){
    try{
      ficheroErrores.close();
    }catch(IOException e){}
  }
```

A.2 Errores que aparecen exclusivamente en un curso

Una sentencia vacía (punto y coma) no constituye el cuerpo de un bucle (13016)

Tipo de aviso: Aviso de código innecesario. Aviso sobre zonas del código que son innecesarias o redundantes.

Herramienta que detecta este aviso: PMD

A no ser que el punto y coma sea utilizado para sustituir el cuerpo de un bucle constituye problablemente un error.

Ejemplo:

An empty statement (semicolon) not part of a loop

```
…
    switch (estadoActual) {
      case INICIO:
        …
        break;
      case BASEDATOS:
        …
      case ERROR:
        gestorDatos.borrar();
        throw new ErrorSemantico();
      default:
        break;
    }
    ;
…
```

Este campo "final" debería de ser convertido en "static" (13046)

Tipo de aviso: Aviso de diseño. Aviso de un posible fallo a la hora de diseñar las aplicaciones, puede ser interesante pensar en una Refactorización del código.

Herramienta que detecta este aviso: PMD

Si a un campo final se le asigna una constante de tiempo de compilación, debería de convertirse en "static", esto ahorra la sobrecarga de cada objeto.

Ejemplo:

This final field could be made static

```
public class Buffer {
       //atributos
       private final int TAM_BUFFER=100;
       private final int CAR_RESPALDO=10;
       …
}
```

Campos no modificados deberían de ser estáticos (11021)

Tipo de aviso: Aviso de diseño. Aviso de un posible fallo a la hora de diseñar las aplicaciones, puede ser interesante pensar en una Refactorización del código.

Herramienta que detecta este aviso: FindBugs

Esta clase contiene una instancia de un campo "final" que es inicializado a un valor estático en tiempo de compilación. Se podría convertir en un campo "static".

Ejemplo:

SS: Unread field: modulo1.Buffer.CAR_RESPALDO; should this field be static? (M)

```
public class Buffer {
       //atributos
       private final int TAM_BUFFER=100;
       private final int CAR_RESPALDO=10;
       …
}
```

Un campo de la clase nunca es leido (11023)

Tipo de aviso: Aviso de código no usado. Aviso sobre zonas de código no usadas en el programa.

Herramienta que detecta este aviso: FindBugs

Se asignan valores a una variable; pero esos valores nunca se utilizan para nada por tanto o hay algún problema o esta variable y todo el código que la utiliza sobra.

Ejemplo:

Unread field: Menu.plantillaActual (M)

```
public class ElegirPlantilla {
       …
   String plantillaActual="";

public ElegirPlantilla(JRootPane rootPane) {
    …

plantillaActual=file.devolverLineaFicheroReferencia(plantillaElegida,System.getPr
operty("user.dir")+"\\Plantillas\\Plantillas.txt",1);
   }
}
```

Comparación redundante entre una referencia y NULL (11011)

Tipo de aviso: Aviso de código innecesario.

Herramienta que detecta este aviso: FindBugs

Si es evidente que una referencia es distinto de null, no hace falta compararla.

Ejemplo:

Redundant comparison to null in ImagePreview.loadImage() (M)

```
…
    //Obtiene la imagen del fichero seleccionado
    ImageIcon tmpIcon = new ImageIcon(file.getPath());

    if (tmpIcon != null) {
           if (tmpIcon.getIconWidth() > 90) {
              //Obtiene la imagen del fichero
           thumbnail = new ImageIcon(tmpIcon.getImage().getScaledInstance(90, -
1,Image.SCALE_DEFAULT));
            } else {
                thumbnail = tmpIcon;
            }
       }
    }
```

El valor de la variable referenciada podría ser NULL (12005)

Tipo de aviso: Referencia NULL. Podría haber problemas al utilizar la variable ya que podría ser NULL.

Herramienta que detecta este aviso: JLint

Se utilizan variables sin comprobar si son NULL o no, después de sentencias alternativas que han podido dejar sin asignar la variable.

Ejemplo:

Value of referenced variable 'tip' may be NULL.

```
…
               Tipo tip=null;
               Simbolo s=null;
               if (ei.isUnCampo()){
                      tip=estruct_padre.getCampo(ei.getNombre());
               }else{
                      …
                      }
                   if (s!=null)
                      tip=s.gettipo();

               }
…
               if (tip.getClass()==Funcion.class){
                      System.out.println("Error     ID     es     una     funcion
"+ei.getNombre()+" : "+ei.getN_linea());
                      System.out.println("\n ¿olvidas los \"( )\" ?");
                      System.exit(-501);
               }
…
```

Llamadas innecesarias a métodos (11003)

Tipo de aviso: Aviso de código innecesario. Código que realmente no es necesario, dificulta la comprensión del código y puede afectar al rendimiento.

Herramienta que detecta este aviso: FindBugs

Java garantiza que las constantes cadena idénticas serán representadas por el mismo objeto cadena. Por tanto, se puede emplear directamente la constante cadena directamente: “”

Ejemplo:

Sugerencias.mostrarSugerencias(String) invokes dubious new String() constructor; just use "" (M)

```
public void mostrarSugerencias(String nombreClase){

       String texto=new String();
       nombreFichero="Sugerencias/Sugerencias_"+ nombreClase + ".htm";
       // Crea el texto a partir del fichero de sugerencias y lo muestra en el
diálogo
       FicherosHTML file=new FicherosHTML();
       texto=file.devolverTextoFichero(nombreFichero);
       //Muestra el texto obtenido a partir del fichero
       JOptionPane.showMessageDialog(this,
texto,"Sugerencias",JOptionPane.INFORMATION_MESSAGE);

}
```

Evitar cláusulas return en bloques finally (13011)

Tipo de aviso: Aviso de excepciones. Aviso de un posible problema a la hora de lanzar o detectar excepciones en el código.

Herramienta que detecta este aviso: PMD

La introducción de return dentro de un bloque finally puede hacer que tengamos excepciones no controladas.

Ejemplo:

Avoid returning from a finally block

```
…
       }finally {
          //Se cierra el statement
          if (stmt!=null)  {
            try { stmt.close();
            }catch (SQLException e){
                System.out.println("Error cerrando Statement");
                System.out.println(e.getMessage());
                return -3;
                }
            }
            return 0;
         }//Fin de finally
 }//Fin de actualizarBaseDatos
```

A.3 Errores que aparecen en distintos cursos

Evitar que una variable local o parámetro oculte un componente de la clase (12002)

Tipo de aviso: Aviso de ocultación. Aviso provocado por una posible ocultación de campos o métodos en una clase.

Herramienta que detecta este aviso: JLint

Es una práctica habitual que en los constructores se utilicen como parámetros formales los mismos nombres que componentes de la clase. Dificulta la lectura del código.

Ejemplo:

Local variable 'jugador' shadows component of class 'juego_de_roll/Viaje'.

```
public class Viaje {
       …
       Jugador jugador;
```

```
…
public Viaje(int pasos,Jugador jugador) {
        this.pasos=pasos;
       this.jugador=jugador;
listaViaje=new Elemento[pasos];
        …
}
…
```

Un componente de la clase está ocultando a otro de una clase base (12001)

Tipo de aviso: Aviso de ocultación. Aviso provocado por una posible ocultación de campos o métodos en una clase.

Herramienta que detecta este aviso: JLint

Un campo en una clase derivada tiene el mismo nombre que otro campo en la clase base. Esto puede causar problemas porque estos dos campos apuntan a diferentes localizaciones y los métodos de la clase base no pueden acceder a uno de los campos; mientras que los métodos de la clase derivada (y a su vez las derivadas de esta) no podrán acceder al otro campo. A veces esto es lo que el programador espera; pero en cualquier caso no mejora la legibilidad del programa.

Ejemplo:

Component 'mag' in class 'el_viaje_a_mordor/Jugador' shadows one in base class 'el_viaje_a_mordor/Personaje'.

Comparación de Strings por medio de sus referencias (12014)

Tipo de aviso: Aviso de comparación. Aviso de posibles problemas a la hora de llevar acabo una comparación.

Herramienta que detecta este aviso: JLint

Se está utilizando los operadores == o != para comparar Strings. El operador == sólo devuelve true si los operandos apuntan al mismo objeto; por tanto, puede devolver false aunque los dos operandos contengan lo mismo. Normalmente en la mayoría de las ocasiones se quiere comprobar que dos Strings contengan el mismo valor y por tanto se debería emplear el método “equals”.

Ejemplo:

Compare strings as object references.

```
if(juego.devolverObjetosJugador(i).tipo=="Ataque"){
…
```

Evitar llamadas a métodos redefinibles durante la construcción (13044)

Tipo de aviso: Aviso de inicialización. Aviso sobre un posible problema en las inicializaciones o en los constructores.

Herramienta que detecta este aviso: PMD

Llamar a métodos redefinibles durante la construcción tiene el riesgo de invocar métodos sobre un objeto que todavía no está completamente construido. Esta situación puede ser difícil de percibir. Puede hacer que una subclase sea incapaz de construir su superclase, o

forzar a repetir todo el proceso de construcción dentro de ella, perdiendo la capacidad de llamar a super(). Si el constructor por defecto tiene una llamada a un método redefinible, puede hacer imposible la instanciación de la subclase.

Ejemplo:

Avoid calls to overridable methods during construction

```
…
public Juego() {
        …
        vida=(int)java.lang.Math.round( java.lang.Math.random() * 100);
        …
        //juego preparado...
        int opcion=0;
        interfaz.presentarAtributosJugador(miJugador);
        while (opcion != InterfazUsuario.MENU_SALIR) {
                if (opcion!=InterfazUsuario.MENU_CAMBIAR_MANO)
                        darPaso();
                if (miJugador.estasVivo()&&!estamosEnMordor()){
                        interfaz.presentarEstadoActual();
                opcion =
                        interfaz.menuGeneral(aventuraActual);
                switch (opcion) {
                        …
                        }
                }
        }
}
```

La jerarquía de llamadas a constructores presenta un interesante dilema. ¿Qué ocurre si uno está dentro de un constructor y se invoca a un método que utiliza el enlace dinámico del objeto que se está construyendo? Dentro de un método ordinario se puede imaginar lo que ocurrirá –la llamada que conlleva el enlace dinámico se resuelve en tiempo de ejecución, pues el objeto no puede saber si pertenece a la clase dentro de la que está el método o a alguna clase derivada de esta. Por consistencia, se podría pensar que esto es lo que debería pasar dentro de los constructores.

Este no es exactamente el caso. Si se invoca a un método de enlace dinámico dentro de un constructor, se utiliza la definición superpuesta de ese método. Sin embargo, el efecto puede ser bastante inesperado, y puede ocultar errores difíciles de encontrar.

Conceptualmente, el trabajo del constructor es crear el objeto (lo que es casi una proeza). Dentro de cualquier constructor, el objeto entero podría formarse sólo parcialmente – sólo se puede saber que se han inicializado los objetos de clase base, pero no se puede saber qué clases se heredan. Una llamada a un método de enlace dinámico, sin embargo, se "sale" de la jerarquía de herencias. Llamada a un método de una clase derivada. Si se hace esto dentro del constructor, se llama a un método que podría manipular miembros que no han sido aún inicializados – lo que ocasionará problemas.

Una cabecera de metodo no debería declarar que lanza la excepción "Exception" (13078)

Tipo de aviso: Aviso de excepciones. Aviso de zonas de código que pueden ser propensas a lanzar excepciones, o bien de zonas donde se está haciendo un mal tratamiento de estas.

Herramienta que detecta este aviso: PMD

Esto no proporciona ninguna información sobre las excepciones que realmente puede lanzar el método. Estos interfaces imprecisos hacen que estos métodos sean difíciles de documentar y de entender. Es mejor declarar una subclase de "Exception".

Ejemplo:

A signature (constructor or method) should'nt have Exception in throws declaration

```
public void insertar(Almacenable x, Nodo refLista)
   throws Exception {
        raiz = insertar(x, refLista, raiz);
}
```

La sentencia "switch" debe de tener una etiqueta "default" (13040)

Tipo de aviso: Aviso de estructura de control. Aviso de un posible error en una estructura de control de flujo del programa.

Herramienta que detecta este aviso: PMD

Ejemplo:

Switch statements should have a default label

```
    …
    switch (nivel) {
      case 0:
        if (nombreElemento.compareToIgnoreCase("basedatos") == 0) {
              …
        }
        break;

      case 1:
        if (atributos == 0) {
              …
        else {
              …
        }
        break;

      case 2:
        …
}
```

Evitar reasignación de parámetros (13042)

Tipo de aviso: Aviso de código innecesario. Aviso sobre zonas del código que son innecesarias o redundantes.

Herramienta que detecta este aviso: PMD

Ejemplo:

Avoid reassigning parameters such as 'nb'

```
  private NodoBinario insertar(Almacenable x, Nodo refLista, NodoBinario nb)
throws Exception {
    if (nb == null) {
      nb = new NodoBinario(x, refLista);
    }
    else if (x.comparadoCon(nb) < 0) {
      nb.izq = insertar(x, refLista, nb.izq);
    }
    …
  }
```

A.4 Errores derivados de las convenciones de código

Evitar el uso de sentencias "if…else" sin llaves (13019)

Tipo de aviso: Aviso de estilo. Aviso de una violación de los convenios de estilo de codificación del Lenguaje. Estas normas pueden ayudar a evitar errores en algunas ocasiones.

Herramienta que detecta este aviso: PMD

Explicación:

Tanto la parte "then" como la "else" de una sentencia "if" deberían de ser siempre bloques y estar entre llaves. Esto hace más fácil añadir sentencias sin añadir errores accidentalmente como consecuencia de olvidar añadir las llaves.

Ejemplo de este aviso:

Avoid using 'if...else' statements without curly braces

```
if (estamosEnMordor())
        interfaz.avisarEnMordor(miJugador.getNombre());
else
        interfaz.avisarMuerto();
```

Este es el aviso que se da más frecuentemente con mucha diferencia respecto a los demás y además es común para todos los grupos de proyectos. Es lógico que sea así ya que los alumnos no reciben ninguna instrucción sobre este convenio y en la evaluación tampoco se tiene en cuenta.

Solución:

Simplemente utilizar siempre llaves para la sentencia (aunque sea una sola) correspondiente de "if…else".

Evitar el uso de sentencias "if" sin llaves (13017)

Tipo de aviso: Aviso de estilo. Aviso de una violación de los convenios de estilo de codificación del Lenguaje. Estas normas pueden ayudar a evitar errores en algunas ocasiones.

Herramienta que detecta este aviso: PMD

Es análogo a lo que decíamos en el aviso anterior (13019), si la sentencia correspondiente del "if" está entre llaves, aunque sea una única sentencia, se reduce el riesgo de que al añadir más sentencias nos olvidemos las llaves e introduzcamos un error en el código.

Ejemplo:

Avoid using if statements without curly braces

```
if (opcion!=InterfazUsuario.MENU_CAMBIAR_MANO) darPaso();
```

Los alumnos no tienen en cuenta este convenio de la misma forma que el 13019 por tanto es lógico que se produzca este error.

El nombre de un método no empieza por minúscula (13071)

Tipo de aviso:

Herramienta que detecta este aviso: PMD

Los nombres de los métodos deberían empezar siempre por minúscula y no deberían contener el carácter guión bajo. Si se respetan las convenciones estándar de nombres, el código es más fácil de leer y de mantener.

Ejemplo:

Method name does not begin with a lower case character.

```
public void CrearObjeto(int num){
…
}
```

Evitar el uso de sentencias "for" sin llaves (13020)

Tipo de aviso: Aviso de estilo. Aviso de una violación de los convenios de estilo de codificación del Lenguaje. Estas normas pueden ayudar a evitar errores en algunas ocasiones.

Herramienta que detecta este aviso: PMD

Ejemplo:

Avoid using 'for' statements without curly braces

```
    …
    for(byte i=0; i<inventario.length; i++)
      if(inventario[i]!=null)
        if(inventario[i].getTipo().equalsIgnoreCase(tipo))
          lista += i+" "+inventario[i].getNombre()+'\n';
    …
```

Los nombres de los métodos no deberían contener el carácter guión bajo (13089)

Tipo de aviso: Aviso de nombre. Aviso sobre un problema potencial por no seguir las normas estándares de estilo para los nombres o por una mala elección de los nombres de campos, métodos o clases que pueden llevar a confusiones.

Herramienta que detecta este aviso: PMD

Ejemplo:

Method names should not contain underscores

```
private boolean inicio_elemento() throws Error {
…
}
```

Los nombres de variables deberían comenzar con minúscula (13088)

Tipo de aviso: Aviso de nombre. Aviso sobre un problema potencial por no seguir las normas estándares de estilo para los nombres o por una mala elección de los nombres de campos, métodos o clases que pueden llevar a confusiones.

Herramienta que detecta este aviso: PMD

Ejemplo:

Variables should start with a lowercase character

```
public class ArbolBB {

  //Nodo padre del árbol
  private NodoArbol raiz;
```

```
  boolean Encontrado = false;
    …
}
```

Apéndice C. Avisos por proyecto e informes de análisis

El sistema de análisis de programas desarrollado en el Capítulo 7. Sistema de análisis de errores de programas: PBA permite recopilar los avisos generados por las diferentes herramientas de análisis estático tras el análisis de los proyectos correspondientes. Como se explica en el Capítulo 10. Clasificación de usuarios basada en la detección de errores hemos utilizado el sistema para analizar distintos proyectos agrupados por distintas etapas en el aprendizaje de la programación. En aquel capítulo se hizo un análisis exhaustivo de los 10 errores más frecuentes para cada grupo de proyectos. En este capítulo incluimos, en primer lugar, las tablas completas de resultados del análisis que incluyen todos los avisos generados ordenados por el código que se le ha asignado. Por otra parte, también se incluyen dos informes de errores frecuentes generados directamente por el sistema de análisis de programas.

A.1 Datos generales para todas las tablas

Para realizar el estudio hemos utilizado seis conjuntos de proyectos correspondientes a dos asignaturas de la Escuela Universitaria de Ingeniería Técnica en Informática de Oviedo de la Universidad de Oviedo.

Los grupos de proyectos analizados se describen en la Tabla 23 (esta tabla ya se incluyó en el Capítulo 10; pero se vuelve a incluir para mayor facilidad de consulta).

Tabla 23. Grupos de proyectos que se utilizarán en el experimento y descripción de las asignaturas a las que pertenecen

Identificador	Asignatura	Curso y duración		Tipo	Año de las muestras
mpmod1	Metodología de la programación	1º	2º cuatrimestre	Troncal	Curso 2002-2003
mpmod2	Metodología de la programación	1º	2º cuatrimestre	Troncal	Curso 2002-2003
edi3mod1	Estructura de datos y de la información	2º	Anual	Troncal	Curso 2003-2004
edi3mod2	Estructura de datos y de la información	2º	Anual	Troncal	Curso 2003-2004
edi4mod1	Estructura de datos y de la información	2º	Anual	Troncal	Curso 2004-2005
edi4mod2	Estructura de datos y de la información	2º	Anual	Troncal	Curso 2004-2005
bd4mod2	Bases de Datos	3º	Anual	Troncal	Curso 2004-2005
pl4	Procesadores de Lenguaje	4º	Anual	Troncal	Curso 2004-2005
pfc	Proyecto Fin de Carrera	5º		Obligatorio	Distintos cursos

En las tablas de datos se incluyen las siguientes columnas:

- Código error. Es el código asignado al aviso correspondiente.
- Descripción del error. Descripción del aviso detectado.

- Número errores. Número total de avisos de este tipo en el grupo de proyectos analizado.
- Núm. proyectos con errores. Número de proyectos que han generado este tipo de aviso.
- Media de errores por proyecto. Es el número medio de avisos de un tipo determinado en los proyectos que contienen este tipo de error.
- Porcentaje de proy. errores. Es el porcentaje de proyectos que contienen este tipo de avisos respecto al total de proyectos analizados para este grupo.

A.2 Datos completos del análisis de mpmod1

En la Tabla 24 se muestran los datos de todos los avisos detectados en el grupo mpmod1, módulo 1 de la asignatura de Metodología de la Programación del curso 2002-2003, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 24. Datos completos del análisis de mpmod1

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
1	class <class name> is public, should be declared in a file named <class name>.java	6	6	1,00	3,28%
4	cannot resolve symbol symbol : class <A> location: class <B>	2	1	2,00	0,55%
5	cannot resolve symbol symbol : method <a> location: class <B>	10	3	3,33	1,64%
6	cannot resolve symbol symbol : variable <a> location: class <B>	3	3	1,00	1,64%
8	illegal start of expression	1	1	1,00	0,55%
9	;' expected'	1	1	1,00	0,55%
12	<identifier> expected	1	1	1,00	0,55%
13	)' expected'	37	11	3,36	6,01%
19	incompatible types found : <type found> required: <type required>	2	1	2,00	0,55%
25	unexpected type required: <type name> found : <type name>	7	1	7,00	0,55%
34	package <package name> does not exist	2	1	2,00	0,55%
37	unreachable statement	4	3	1,33	1,64%
38	duplicate class: <class name>	1	1	1,00	0,55%
41	<class name> is not abstract and does not override abstract method <method name> in %	6	1	6,00	0,55%
10005	May be wrong assumption about operators priorities	3	1	3,00	0,55%
10006	May be wrong assumption about logical operators precedence	7	4	1,75	2,19%
10010	May be wrong assumption about bit operation priority	3	1	3,00	0,55%
10011	May be wrong assumption about loop body	51	32	1,59	17,49%
10012	May be wrong assumption about IF body	40	23	1,74	12,57%
10013	May be wrong assumption about ELSE branch association	79	41	1,93	22,40%
10016	Possible miss of BREAK before CASE/DEFAULT	17	14	1,21	7,65%
11002	This method might ignore or drop an exception. In	2	1	2,00	0,55%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
	general, exceptions should be handled or reported in some way, or they should be thrown out of the method.				
11003	Innecesary calls to methods.	7	3	2,33	1,64%
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	199	75	2,65	40,98%
11006	Self assignment of field or local variable	32	15	2,13	8,20%
11010	Problems than can cause a NullPointerException.	8	3	2,67	1,64%
11011	This method contains a redundant comparison of a reference value to null.	16	5	3,20	2,73%
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	62	12	5,17	6,56%
11015	Useless control flow statement	6	5	1,20	2,73%
11017	Problems with confusing named methods.	2	2	1,00	1,09%
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless.	159	26	6,12	14,21%
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	124	25	4,96	13,66%
11022	This field is never used. Consider removing it from the class.	189	54	3,50	29,51%
11023	This field is never read. Consider removing it from the class.	174	68	2,56	37,16%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	3	1	3,00	0,55%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	1	1	1,00	0,55%
12000	Method <method name> is not overridden by method with the same name of derived class <class name>.	18	10	1,80	5,46%
12001	Component <name> in class <class name> shadows one in base class <super class name>.	1828	106	17,25	57,92%
12002	Local variable <variable name> shadows component of class <class name>.	487	104	4,68	56,83%
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	1	1	1,00	0,55%
12005	Value of referenced variable <variable name> may be NULL.	97	7	13,86	3,83%
12011	Comparison always produces the same result.	39	20	1,95	10,93%
12012	Compared expressions can be equal only when both of them are 0.	5	4	1,25	2,19%
12014	Compare strings as object references.	458	76	6,03	41,53%
12027	Zero operand for <operation symbol> operation.	17	8	2,13	4,37%
12028	Inequality comparison can be replaced with equality comparison.	19	13	1,46	7,10%
12029	Index [<min>,<max>] may be out of array bounds.	11	7	1,57	3,83%
13000	Avoid empty catch blocks	172	159	1,08	86,89%
13001	Avoid empty if' statements'	16	8	2,00	4,37%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13006	Avoid modifying an outer loop incrementer in an inner loop for update expression	1	1	1,00	0,55%
13016	An empty statement (semicolon) not part of a loop	197	159	1,24	86,89%
13017	Avoid using if statements without curly braces	1504	153	9,83	83,61%
13018	Avoid using while' statements without curly braces'	17	10	1,70	5,46%
13019	Avoid using if...else' statements without curly braces'	3426	160	21,41	87,43%
13020	Avoid using for' statements without curly braces'	1745	161	10,84	87,98%
13036	All methods are static. Consider using Singleton instead.	11	10	1,10	5,46%
13038	Avoid unnecessary if..then..else statements when returning a boolean	24	15	1,60	8,20%
13039	Avoid unnecessary comparisons in boolean expressions	693	110	6,30	60,11%
13040	Switch statements should have a default label	357	116	3,08	63,39%
13042	Avoid reassigning parameters such as <name>"	58	35	1,66	19,13%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	109	63	1,73	34,43%
13044	Avoid calls to overridable methods during construction	1277	151	8,46	82,51%
13046	This final field could be made static	101	25	4,04	13,66%
13051	The default label should be the last label in a switch statement	4	3	1,33	1,64%
13052	A non-case label was present in a switch statement	1	1	1,00	0,55%
13059	Avoid duplicate imports such as <package>"	2	2	1,00	1,09%
13060	Avoid importing anything from the package java.lang"	30	10	3,00	5,46%
13061	Avoid unused imports such as <package>"	7	6	1,17	3,28%
13071	Method name does not begin with a lower case character.	371	43	8,63	23,50%
13072	Class names should begin with an uppercase character and not include underscores	35	12	2,92	6,56%
13073	Abstract classes should be named AbstractXXX"	168	105	1,60	57,38%
13075	Classes should not have non-constructor methods with the same name as the class	1	1	1,00	0,55%
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	2	1	2,00	0,55%
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	684	161	4,25	87,98%
13081	Avoid instantiating String objects; this is usually unnecessary.	7	3	2,33	1,64%
13082	Avoid calling toString() on String objects; this is unnecessary	1	1	1,00	0,55%
13083	Avoid unused private fields such as <name>"	26	14	1,86	7,65%
13084	Avoid unused local variables such as <name>"	572	160	3,58	87,43%
13085	Avoid unused private methods such as <name>"	1	1	1,00	0,55%
13087	Variables that are not final should not contain underscores.	145	23	6,30	12,57%
13088	Variables should start with a lowercase character	336	38	8,84	20,77%
13089	Method names should not contain underscores	172	17	10,12	9,29%
13090	Variables that are final and static should be in all caps.	13	5	2,60	2,73%
13091	Class names should not contain underscores	19	5	3,80	2,73%

A.3 Datos completos del análisis de mpmod2

En la Tabla 25 se muestran los datos de todos los avisos detectados en el grupo mpmod2, módulo 2 de la asignatura de Metodología de la Programación del curso 2002-2003, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 25. Datos completos del análisis de mpmod2

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
1	class <class name> is public, should be declared in a file named <class name>.java	1	1	1,00	0,75%
5	cannot resolve symbol symbol : method <a> location: class <B>	5	3	1,67	2,26%
13	)' expected'	413	59	7,00	44,36%
37	unreachable statement	29	24	1,21	18,05%
10005	May be wrong assumption about operators priorities	10	1	10,00	0,75%
10006	May be wrong assumption about logical operators precedence	2	2	1,00	1,50%
10007	May be =' used instead of '=="	2	2	1,00	1,50%
10010	May be wrong assumption about bit operation priority	8	2	4,00	1,50%
10011	May be wrong assumption about loop body	19	14	1,36	10,53%
10012	May be wrong assumption about IF body	122	47	2,60	35,34%
10013	May be wrong assumption about ELSE branch association	116	49	2,37	36,84%
10014	Suspicios SWITCH without body	2	2	1,00	1,50%
10016	Possible miss of BREAK before CASE/DEFAULT	11	11	1,00	8,27%
11001	The return value of this method should be checked.	6	3	2,00	2,26%
11002	This method might ignore or drop an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.	9	6	1,50	4,51%
11003	Innecesary calls to methods.	28	8	3,50	6,02%
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	97	42	2,31	31,58%
11006	Self assignment of field or local variable	23	12	1,92	9,02%
11010	Problems than can cause a NullPointerException.	23	18	1,28	13,53%
11011	This method contains a redundant comparison of a reference value to null.	28	19	1,47	14,29%
11012	The method creates an IO stream object, does not assign it to any fields, pass it to other methods, or return it, and does not appear to close the stream on all paths out of the method. It is generally a good idea to use a finally block to ensure that str	43	33	1,30	24,81%
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	2	2	1,00	1,50%
11015	Useless control flow statement	11	5	2,20	3,76%
11017	Problems with confusing named methods.	2	2	1,00	1,50%
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless.	29	1	29,00	0,75%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	19	4	4,75	3,01%
11022	This field is never used. Consider removing it from the class.	63	25	2,52	18,80%
11023	This field is never read. Consider removing it from the class.	78	37	2,11	27,82%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	10	2	5,00	1,50%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	10	7	1,43	5,26%
12001	Component <name> in class <class name> shadows one in base class <super class name>.	45	12	3,75	9,02%
12002	Local variable <variable name> shadows component of class <class name>.	546	72	7,58	54,14%
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	2	2	1,00	1,50%
12005	Value of referenced variable <variable name> may be NULL.	192	32	6,00	24,06%
12011	Comparison always produces the same result.	69	35	1,97	26,32%
12014	Compare strings as object references.	311	43	7,23	32,33%
12027	Zero operand for <operation symbol> operation.	3	2	1,50	1,50%
12028	Inequality comparison can be replaced with equality comparison.	12	7	1,71	5,26%
12029	Index [<min>,<max>] may be out of array bounds.	8	4	2,00	3,01%
13000	Avoid empty catch blocks	244	78	3,13	58,65%
13001	Avoid empty if' statements'	53	17	3,12	12,78%
13002	Avoid empty while' statements'	2	2	1,00	1,50%
13016	An empty statement (semicolon) not part of a loop	92	70	1,31	52,63%
13017	Avoid using if statements without curly braces	1753	112	15,65	84,21%
13018	Avoid using while' statements without curly braces'	129	33	3,91	24,81%
13019	Avoid using if...else' statements without curly braces'	5097	114	44,71	85,71%
13020	Avoid using for' statements without curly braces'	622	75	8,29	56,39%
13036	All methods are static. Consider using Singleton instead.	13	9	1,44	6,77%
13038	Avoid unnecessary if..then..else statements when returning a boolean	21	12	1,75	9,02%
13039	Avoid unnecessary comparisons in boolean expressions	791	73	10,84	54,89%
13040	Switch statements should have a default label	163	75	2,17	56,39%
13042	Avoid reassigning parameters such as <name>"	124	45	2,76	33,83%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	41	36	1,14	27,07%
13044	Avoid calls to overridable methods during construction	1179	109	10,82	81,95%
13046	This final field could be made static	20	5	4,00	3,76%
13059	Avoid duplicate imports such as <package>"	2	2	1,00	1,50%
13060	Avoid importing anything from the package java.lang"	1	1	1,00	0,75%
13061	Avoid unused imports such as <package>"	5	3	1,67	2,26%
13071	Method name does not begin with a lower case	575	49	11,73	36,84%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
	character.				
13072	Class names should begin with an uppercase character and not include underscores	19	9	2,11	6,77%
13073	Abstract classes should be named AbstractXXX"	63	59	1,07	44,36%
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	181	24	7,54	18,05%
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	529	107	4,94	80,45%
13081	Avoid instantiating String objects; this is usually unnecessary.	33	6	5,50	4,51%
13082	Avoid calling toString() on String objects; this is unnecessary	4	3	1,33	2,26%
13083	Avoid unused private fields such as <name>"	25	15	1,67	11,28%
13084	Avoid unused local variables such as <name>"	609	115	5,30	86,47%
13085	Avoid unused private methods such as <name>"	10	7	1,43	5,26%
13086	Avoid unused formal parameters such as <name>"	5	5	1,00	3,76%
13087	Variables that are not final should not contain underscores.	83	8	10,38	6,02%
13088	Variables should start with a lowercase character	294	41	7,17	30,83%
13089	Method names should not contain underscores	94	6	15,67	4,51%
13091	Class names should not contain underscores	10	3	3,33	2,26%

A.4 Datos completos del análisis de edi3mod1

En la Tabla 26 se muestran los datos de todos los avisos detectados en el grupo edi3mod1, módulo 1 de la asignatura de Estructura de Datos y de la Información del curso 2003-2004, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 26. Datos completos del análisis de edi3mod1

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
5	cannot resolve symbol symbol : method <a> location: class <B>	3	1	3,00	3,70%
9	;' expected'	1	1	1,00	3,70%
14	missing method body, or declare abstract	10	1	10,00	3,70%
26	not a statement	1	1	1,00	3,70%
37	unreachable statement	1	1	1,00	3,70%
38	duplicate class: <class name>	2	1	2,00	3,70%
10005	May be wrong assumption about operators priorities	3	2	1,50	7,41%
10006	May be wrong assumption about logical operators precedence	1	1	1,00	3,70%
10010	May be wrong assumption about bit operation priority	3	2	1,50	7,41%
10011	May be wrong assumption about loop body	2	2	1,00	7,41%
10012	May be wrong assumption about IF body	15	7	2,14	25,93%
10013	May be wrong assumption about ELSE branch association	26	7	3,71	25,93%
10016	Possible miss of BREAK before CASE/DEFAULT	9	2	4,50	7,41%
11002	This method might ignore or drop an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.	12	7	1,71	25,93%
11003	Innecesary calls to methods.	15	1	15,00	3,70%
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	43	10	4,30	37,04%
11006	Self assignment of field or local variable	1	1	1,00	3,70%
11010	Problems than can cause a NullPointerException.	5	2	2,50	7,41%
11011	This method contains a redundant comparison of a reference value to null.	10	6	1,67	22,22%
11012	The method creates an IO stream object, does not assign it to any fields, pass it to other methods, or return it, and does not appear to close the stream on all paths out of the method. It is generally a good idea to use a finally block to ensure that str	4	4	1,00	14,81%
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	1	1	1,00	3,70%
11015	Useless control flow statement	5	3	1,67	11,11%
11019	This method ignores the return value of one of the variants of java.io.InputStream.read() which can return multiple bytes.	1	1	1,00	3,70%
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless.	4	3	1,33	11,11%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	176	9	19,56	33,33%
11022	This field is never used. Consider removing it from the class.	20	10	2,00	37,04%
11023	This field is never read. Consider removing it from the class.	19	9	2,11	33,33%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	2	2	1,00	7,41%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	4	4	1,00	14,81%
12000	Method <method name> is not overridden by method with the same name of derived class <class name>.	9	2	4,50	7,41%
12001	Component <name> in class <class name> shadows one in base class <super class name>.	14	4	3,50	14,81%
12002	Local variable <variable name> shadows component of class <class name>.	76	16	4,75	59,26%
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	2	2	1,00	7,41%
12005	Value of referenced variable <variable name> may be NULL.	27	5	5,40	18,52%
12011	Comparison always produces the same result.	26	9	2,89	33,33%
12012	Compared expressions can be equal only when both of them are 0.	2	1	2,00	3,70%
12014	Compare strings as object references.	71	11	6,45	40,74%
12027	Zero operand for <operation symbol> operation.	2	1	2,00	3,70%
12029	Index [<min>,<max>] may be out of array bounds.	8	6	1,33	22,22%
13000	Avoid empty catch blocks	90	13	6,92	48,15%
13001	Avoid empty if' statements'	11	2	5,50	7,41%
13002	Avoid empty while' statements'	2	1	2,00	3,70%
13009	Ensure you override both equals() and hashCode()	7	1	7,00	3,70%
13015	Do not use if' statements that are always true or always false'	4	2	2,00	7,41%
13016	An empty statement (semicolon) not part of a loop	15	10	1,50	37,04%
13017	Avoid using if statements without curly braces	693	19	36,47	70,37%
13018	Avoid using while' statements without curly braces'	31	11	2,82	40,74%
13019	Avoid using if...else' statements without curly braces'	2041	19	107,42	70,37%
13020	Avoid using for' statements without curly braces'	43	11	3,91	40,74%
13036	All methods are static. Consider using Singleton instead.	8	3	2,67	11,11%
13038	Avoid unnecessary if..then..else statements when returning a boolean	44	11	4,00	40,74%
13039	Avoid unnecessary comparisons in boolean expressions	24	7	3,43	25,93%
13040	Switch statements should have a default label	116	18	6,44	66,67%
13042	Avoid reassigning parameters such as <name>"	20	10	2,00	37,04%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	24	15	1,60	55,56%
13044	Avoid calls to overridable methods during construction	106	13	8,15	48,15%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13046	This final field could be made static	145	10	14,50	37,04%
13059	Avoid duplicate imports such as <package>"	1	1	1,00	3,70%
13060	Avoid importing anything from the package java.lang"	16	5	3,20	18,52%
13061	Avoid unused imports such as <package>"	8	4	2,00	14,81%
13062	No need to import a type thats in the same package'	12	4	3,00	14,81%
13071	Method name does not begin with a lower case character.	208	9	23,11	33,33%
13072	Class names should begin with an uppercase character and not include underscores	5	2	2,50	7,41%
13073	Abstract classes should be named AbstractXXX"	5	5	1,00	18,52%
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	204	11	18,55	40,74%
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	195	21	9,29	77,78%
13081	Avoid instantiating String objects; this is usually unnecessary.	24	2	12,00	7,41%
13082	Avoid calling toString() on String objects; this is unnecessary	2	1	2,00	3,70%
13083	Avoid unused private fields such as <name>"	15	9	1,67	33,33%
13084	Avoid unused local variables such as <name>"	150	21	7,14	77,78%
13085	Avoid unused private methods such as <name>"	4	4	1,00	14,81%
13087	Variables that are not final should not contain underscores.	66	10	6,60	37,04%
13088	Variables should start with a lowercase character	61	9	6,78	33,33%
13089	Method names should not contain underscores	115	12	9,58	44,44%
13090	Variables that are final and static should be in all caps.	9	2	4,50	7,41%
13091	Class names should not contain underscores	12	5	2,40	18,52%

A.5 Datos completos del análisis de edi3mod2

En la Tabla 27 se muestran los datos de todos los avisos detectados en el grupo edi3mod2, módulo 2 de la asignatura de Estructura de Datos y de la Información del curso 2003-2004, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 27. Datos completos del análisis de edi3mod2

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
4	cannot resolve symbol symbol : class <A> location: class <B>	4	1	4,00	0,04
5	cannot resolve symbol symbol : method <a> location: class <B>	5	2	2,50	0,08
13	)' expected'	38	4	9,50	0,16
34	package <package name> does not exist	1	1	1,00	0,04
10005	May be wrong assumption about operators priorities	3	2	1,50	0,08
10006	May be wrong assumption about logical operators precedence	1	1	1,00	0,04
10010	May be wrong assumption about bit operation priority	3	2	1,50	0,08
10011	May be wrong assumption about loop body	3	3	1,00	0,12
10012	May be wrong assumption about IF body	21	10	2,10	0,4
10013	May be wrong assumption about ELSE branch association	27	8	3,38	0,32
10016	Possible miss of BREAK before CASE/DEFAULT	9	2	4,50	0,08
11001	The return value of this method should be checked.	3	1	3,00	0,04
11002	This method might ignore or drop an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.	36	11	3,27	0,44
11003	Innecesary calls to methods.	7	5	1,40	0,2
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	45	9	5,00	0,36
11006	Self assignment of field or local variable	2	2	1,00	0,08
11010	Problems than can cause a NullPointerException.	13	5	2,60	0,2
11011	This method contains a redundant comparison of a reference value to null.	11	7	1,57	0,28
11012	The method creates an IO stream object, does not assign it to any fields, pass it to other methods, or return it, and does not appear to close the stream on all paths out of the method. It is generally a good idea to use a finally block to ensure that str	5	4	1,25	0,16
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	3	3	1,00	0,12
11015	Useless control flow statement	7	5	1,40	0,2
11019	This method ignores the return value of one of the variants of java.io.InputStream.read() which can return multiple bytes.	1	1	1,00	0,04
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have	4	3	1,33	0,12

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
	been initialized?), or remove it if it is useless.				
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	241	13	18,54	0,52
11022	This field is never used. Consider removing it from the class.	49	14	3,50	0,56
11023	This field is never read. Consider removing it from the class.	43	14	3,07	0,56
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	2	2	1,00	0,08
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	7	6	1,17	0,24
12000	Method <method name> is not overridden by method with the same name of derived class <class name>.	8	3	2,67	0,12
12001	Component <name> in class <class name> shadows one in base class <super class name>.	31	8	3,88	0,32
12002	Local variable <variable name> shadows component of class <class name>.	126	19	6,63	0,76
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	2	2	1,00	0,08
12005	Value of referenced variable <variable name> may be NULL.	52	11	4,73	0,44
12011	Comparison always produces the same result.	27	11	2,45	0,44
12012	Compared expressions can be equal only when both of them are 0.	2	1	2,00	0,04
12014	Compare strings as object references.	73	10	7,30	0,4
12027	Zero operand for <operation symbol> operation.	7	3	2,33	0,12
12028	Inequality comparison can be replaced with equality comparison.	1	1	1,00	0,04
12029	Index [<min>,<max>] may be out of array bounds.	8	6	1,33	0,24
13000	Avoid empty catch blocks	333	18	18,50	0,72
13001	Avoid empty if' statements'	14	4	3,50	0,16
13002	Avoid empty while' statements'	3	2	1,50	0,08
13009	Ensure you override both equals() and hashCode()	11	1	11,00	0,04
13015	Do not use if' statements that are always true or always false'	4	2	2,00	0,08
13016	An empty statement (semicolon) not part of a loop	23	12	1,92	0,48
13017	Avoid using if statements without curly braces	865	19	45,53	0,76
13018	Avoid using while' statements without curly braces'	43	11	3,91	0,44
13019	Avoid using if...else' statements without curly braces'	2441	20	122,05	0,8
13020	Avoid using for' statements without curly braces'	121	17	7,12	0,68
13036	All methods are static. Consider using Singleton instead.	20	7	2,86	0,28
13038	Avoid unnecessary if..then..else statements when returning a boolean	51	13	3,92	0,52
13039	Avoid unnecessary comparisons in boolean expressions	40	9	4,44	0,36
13040	Switch statements should have a default label	125	18	6,94	0,72
13042	Avoid reassigning parameters such as <name>"	49	20	2,45	0,8

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	24	16	1,50	0,64
13044	Avoid calls to overridable methods during construction	76	16	4,75	0,64
13046	This final field could be made static	210	14	15,00	0,56
13047	Avoid instantiating Boolean objects; you can usually invoke Boolean.valueOf() instead.	12	3	4,00	0,12
13049	Object clone() should be implemented with super.clone()	3	1	3,00	0,04
13059	Avoid duplicate imports such as <package>''	11	3	3,67	0,12
13060	Avoid importing anything from the package java.lang''	26	10	2,60	0,4
13061	Avoid unused imports such as <package>''	24	7	3,43	0,28
13062	No need to import a type thats in the same package'	12	4	3,00	0,16
13071	Method name does not begin with a lower case character.	301	14	21,50	0,56
13072	Class names should begin with an uppercase character and not include underscores	13	3	4,33	0,12
13073	Abstract classes should be named AbstractXXX''	12	7	1,71	0,28
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	369	19	19,42	0,76
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	254	22	11,55	0,88
13081	Avoid instantiating String objects; this is usually unnecessary.	2	1	2,00	0,04
13082	Avoid calling toString() on String objects; this is unnecessary	2	1	2,00	0,04
13083	Avoid unused private fields such as <name>''	18	11	1,64	0,44
13084	Avoid unused local variables such as <name>''	221	22	10,05	0,88
13085	Avoid unused private methods such as <name>''	6	6	1,00	0,24
13086	Avoid unused formal parameters such as <name>''	2	2	1,00	0,08
13087	Variables that are not final should not contain underscores.	67	10	6,70	0,4
13088	Variables should start with a lowercase character	92	11	8,36	0,44
13089	Method names should not contain underscores	122	13	9,38	0,52
13090	Variables that are final and static should be in all caps.	10	3	3,33	0,12
13091	Class names should not contain underscores	13	5	2,60	0,2

A.6 Datos completos del análisis de edi4mod1

En la Tabla 28 se muestran los datos de todos los avisos detectados en el grupo edi4mod1, módulo 1 de la asignatura de Estructura de Datos y de la Información, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 28. Datos completos del análisis de edi4mod1

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
1	class <class name> is public, should be declared in a file named <class name>.java	1	1	1,00	0,83%
4	cannot resolve symbol symbol : class <A> location: class <B>	13	2	6,50	1,65%
5	cannot resolve symbol symbol : method <a> location: class <B>	85	3	28,33	2,48%
6	cannot resolve symbol symbol : variable <a> location: class <B>	5	2	2,50	1,65%
9	;' expected'	32	2	16,00	1,65%
11	illegal character: <character in hexadecimal>	3	1	3,00	0,83%
13	)' expected'	13	7	1,86	5,79%
26	not a statement	11	3	3,67	2,48%
34	package <package name> does not exist	8	1	8,00	0,83%
37	unreachable statement	3	3	1,00	2,48%
38	duplicate class: <class name>	59	5	11,80	4,13%
10005	May be wrong assumption about operators priorities	5	2	2,50	1,65%
10006	May be wrong assumption about logical operators precedence	8	6	1,33	4,96%
10007	May be =' used instead of '=="	6	3	2,00	2,48%
10010	May be wrong assumption about bit operation priority	3	1	3,00	0,83%
10011	May be wrong assumption about loop body	24	16	1,50	13,22%
10012	May be wrong assumption about IF body	135	42	3,21	34,71%
10013	May be wrong assumption about ELSE branch association	323	58	5,57	47,93%
10016	Possible miss of BREAK before CASE/DEFAULT	40	23	1,74	19,01%
11001	The return value of this method should be checked.	3	2	1,50	1,65%
11002	This method might ignore or drop an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.	62	29	2,14	23,97%
11003	Innecesary calls to methods.	166	19	8,74	15,70%
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	60	26	2,31	21,49%
11006	Self assignment of field or local variable	14	9	1,56	7,44%
11007	Problems with the method finalize() use.	56	5	11,20	4,13%
11010	Problems than can cause a NullPointerException.	26	12	2,17	9,92%
11011	This method contains a redundant comparison of a reference value to null.	32	11	2,91	9,09%
11012	The method creates an IO stream object, does not assign it to any fields, pass it to other methods, or return it, and does not appear to	39	28	1,39	23,14%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
	close the stream on all paths out of the method. It is generally a good idea to use a finally block to ensure that str				
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	7	7	1,00	5,79%
11015	Useless control flow statement	26	13	2,00	10,74%
11017	Problems with confusing named methods.	3	3	1,00	2,48%
11019	This method ignores the return value of one of the variants of java.io.InputStream.read() which can return multiple bytes.	10	9	1,11	7,44%
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless.	26	15	1,73	12,40%
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	1021	73	13,99	60,33%
11022	This field is never used. Consider removing it from the class.	142	56	2,54	46,28%
11023	This field is never read. Consider removing it from the class.	228	75	3,04	61,98%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	3	2	1,50	1,65%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	43	26	1,65	21,49%
12000	Method <method name> is not overridden by method with the same name of derived class <class name>.	1	1	1,00	0,83%
12001	Component <name> in class <class name> shadows one in base class <super class name>.	16	4	4,00	3,31%
12002	Local variable <variable name> shadows component of class <class name>.	483	88	5,49	72,73%
12003	Method finalize() doesnt call super.finalize().'	37	5	7,40	4,13%
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	2	2	1,00	1,65%
12005	Value of referenced variable <variable name> may be NULL.	183	28	6,54	23,14%
12011	Comparison always produces the same result.	37	22	1,68	18,18%
12012	Compared expressions can be equal only when both of them are 0.	32	10	3,20	8,26%
12014	Compare strings as object references.	156	29	5,38	23,97%
12027	Zero operand for <operation symbol> operation.	4	3	1,33	2,48%
12028	Inequality comparison can be replaced with equality comparison.	108	15	7,20	12,40%
12029	Index [<min>,<max>] may be out of array bounds.	25	11	2,27	9,09%
13000	Avoid empty catch blocks	347	67	5,18	55,37%
13001	Avoid empty if' statements'	100	33	3,03	27,27%
13002	Avoid empty while' statements'	7	4	1,75	3,31%
13004	Avoid empty finally blocks	1	1	1,00	0,83%
13006	Avoid modifying an outer loop incrementer in an inner loop for update expression	2	2	1,00	1,65%
13013	Avoid unnecessary return statements	16	3	5,33	2,48%
13015	Do not use if' statements that are always true or always false'	4	3	1,33	2,48%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13016	An empty statement (semicolon) not part of a loop	180	39	4,62	32,23%
13017	Avoid using if statements without curly braces	2547	92	27,68	76,03%
13018	Avoid using while' statements without curly braces'	133	47	2,83	38,84%
13019	Avoid using if...else' statements without curly braces'	7050	96	73,44	79,34%
13020	Avoid using for' statements without curly braces'	218	67	3,25	55,37%
13036	All methods are static. Consider using Singleton instead.	70	42	1,67	34,71%
13038	Avoid unnecessary if..then..else statements when returning a boolean	157	54	2,91	44,63%
13039	Avoid unnecessary comparisons in boolean expressions	334	42	7,95	34,71%
13040	Switch statements should have a default label	349	90	3,88	74,38%
13042	Avoid reassigning parameters such as <name>"	412	95	4,34	78,51%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	67	45	1,49	37,19%
13044	Avoid calls to overridable methods during construction	260	61	4,26	50,41%
13046	This final field could be made static	1192	77	15,48	63,64%
13053	Avoid empty finalize methods	3	2	1,50	1,65%
13056	Last statement in finalize method should be a call to super.finalize()	34	4	8,50	3,31%
13057	Explicit call of finalize method	16	1	16,00	0,83%
13058	If you override finalize(), make it protected	37	4	9,25	3,31%
13059	Avoid duplicate imports such as <package>"	19	6	3,17	4,96%
13060	Avoid importing anything from the package java.lang"	47	14	3,36	11,57%
13061	Avoid unused imports such as <package>"	11	5	2,20	4,13%
13062	No need to import a type thats in the same package'	7	3	2,33	2,48%
13071	Method name does not begin with a lower case character.	1016	39	26,05	32,23%
13072	Class names should begin with an uppercase character and not include underscores	43	16	2,69	13,22%
13073	Abstract classes should be named AbstractXXX"	9	7	1,29	5,79%
13075	Classes should not have non-constructor methods with the same name as the class	3	3	1,00	2,48%
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	246	25	9,84	20,66%
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	628	100	6,28	82,64%
13081	Avoid instantiating String objects; this is usually unnecessary.	249	16	15,56	13,22%
13082	Avoid calling toString() on String objects; this is unnecessary	2	2	1,00	1,65%
13083	Avoid unused private fields such as <name>"	143	63	2,27	52,07%
13084	Avoid unused local variables such as <name>"	513	102	5,03	84,30%
13085	Avoid unused private methods such as <name>"	39	23	1,70	19,01%
13086	Avoid unused formal parameters such as <name>"	24	17	1,41	14,05%
13087	Variables that are not final should not contain underscores.	253	47	5,38	38,84%
13088	Variables should start with a lowercase character	306	51	6,00	42,15%
13089	Method names should not contain underscores	404	76	5,32	62,81%
13090	Variables that are final and static should be in all caps.	34	14	2,43	11,57%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13091	Class names should not contain underscores	8	4	2,00	3,31%

A.7 Datos completos del análisis de edi4mod2

En la Tabla 29 se muestran los datos de todos los avisos detectados en el grupo edi4mod2, módulo 2 de la asignatura de Estructura de datos y de la Información del curso 2004-2005, ordenados por código, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 29. Datos completos del análisis de edi4mod2

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
4	cannot resolve symbol symbol : class <A> location: class <B>	75	1	75,00	0,61%
5	cannot resolve symbol symbol : method <a> location: class <B>	2	1	2,00	0,61%
9	;' expected'	34	3	11,33	1,83%
13	)' expected'	58	24	2,42	14,63%
26	not a statement	2	2	1,00	1,22%
34	package <package name> does not exist	1	1	1,00	0,61%
37	unreachable statement	7	2	3,50	1,22%
40	cannot resolve symbol symbol : class <class name> location: interface <interface name>	1	1	1,00	0,61%
10005	May be wrong assumption about operators priorities	54	14	3,86	8,54%
10006	May be wrong assumption about logical operators precedence	14	10	1,40	6,10%
10007	May be =' used instead of '=="	6	5	1,20	3,05%
10010	May be wrong assumption about bit operation priority	29	3	9,67	1,83%
10011	May be wrong assumption about loop body	83	35	2,37	21,34%
10012	May be wrong assumption about IF body	226	63	3,59	38,41%
10013	May be wrong assumption about ELSE branch association	644	93	6,92	56,71%
10016	Possible miss of BREAK before CASE/DEFAULT	84	43	1,95	26,22%
11001	The return value of this method should be checked.	27	10	2,70	6,10%
11002	This method might ignore or drop an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.	96	47	2,04	28,66%
11003	Innecesary calls to methods.	301	43	7,00	26,22%
11004	This code compares java.lang.String objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the String.intern() method, the same string value may be represented by	130	55	2,36	33,54%
11006	Self assignment of field or local variable	41	29	1,41	17,68%
11007	Problems with the method finalize() use.	56	5	11,20	3,05%
11010	Problems than can cause a NullPointerException.	42	25	1,68	15,24%
11011	This method contains a redundant comparison of a reference value to null.	76	35	2,17	21,34%
11012	The method creates an IO stream object, does not assign it to any fields, pass it to other	87	53	1,64	32,32%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
	methods, or return it, and does not appear to close the stream on all paths out of the method. It is generally a good idea to use a finally block to ensure that str				
11014	This constructor reads a field which has not yet been assigned a value. This is often caused when the programmer mistakenly uses the field instead of one of the constructors parameters.'	45	31	1,45	18,90%
11015	Useless control flow statement	36	18	2,00	10,98%
11017	Problems with confusing named methods.	7	7	1,00	4,27%
11019	This method ignores the return value of one of the variants of java.io.InputStream.read() which can return multiple bytes.	14	12	1,17	7,32%
11020	This field is never written. All reads of it will return the default value. Check for errors (should it have been initialized?), or remove it if it is useless.	50	28	1,79	17,07%
11021	This class contains an instance final field that is initialized to a compile-time static value. Consider making the field static.	2354	116	20,29	70,73%
11022	This field is never used. Consider removing it from the class.	315	94	3,35	57,32%
11023	This field is never read. Consider removing it from the class.	440	121	3,64	73,78%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	29	7	4,14	4,27%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	118	62	1,90	37,80%
12000	Method <method name> is not overridden by method with the same name of derived class <class name>.	15	10	1,50	6,10%
12001	Component <name> in class <class name> shadows one in base class <super class name>.	346	36	9,61	21,95%
12002	Local variable <variable name> shadows component of class <class name>.	999	139	7,19	84,76%
12003	Method finalize() doesnt call super.finalize().'	37	5	7,40	3,05%
12004	Method <method name> can be invoked with NULL as <position> parameter and this parameter is used without check for null.	15	12	1,25	7,32%
12005	Value of referenced variable <variable name> may be NULL.	344	60	5,73	36,59%
12007	Result of operation <operation> is always 0.	7	3	2,33	1,83%
12011	Comparison always produces the same result.	149	64	2,33	39,02%
12012	Compared expressions can be equal only when both of them are 0.	64	17	3,76	10,37%
12014	Compare strings as object references.	244	62	3,94	37,80%
12015	Switch case constant <constant> cant be produced by switch expression.'	3	2	1,50	1,22%
12027	Zero operand for <operation symbol> operation.	26	12	2,17	7,32%
12028	Inequality comparison can be replaced with equality comparison.	88	25	3,52	15,24%
12029	Index [<min>,<max>] may be out of array bounds.	77	24	3,21	14,63%
13000	Avoid empty catch blocks	994	109	9,12	66,46%
13001	Avoid empty if' statements'	213	68	3,13	41,46%
13002	Avoid empty while' statements'	12	7	1,71	4,27%
13004	Avoid empty finally blocks	2	2	1,00	1,22%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13006	Avoid modifying an outer loop incrementer in an inner loop for update expression	1	1	1,00	0,61%
13008	Avoid unnecessary temporaries when converting primitives to Strings	1	1	1,00	0,61%
13009	Ensure you override both equals() and hashCode()	13	2	6,50	1,22%
13013	Avoid unnecessary return statements	18	10	1,80	6,10%
13015	Do not use if' statements that are always true or always false'	3	2	1,50	1,22%
13016	An empty statement (semicolon) not part of a loop	291	68	4,28	41,46%
13017	Avoid using if statements without curly braces	4687	127	36,91	77,44%
13018	Avoid using while' statements without curly braces'	245	67	3,66	40,85%
13019	Avoid using if...else' statements without curly braces'	12213	131	93,23	79,88%
13020	Avoid using for' statements without curly braces'	582	97	6,00	59,15%
13036	All methods are static. Consider using Singleton instead.	109	75	1,45	45,73%
13038	Avoid unnecessary if..then..else statements when returning a boolean	339	83	4,08	50,61%
13039	Avoid unnecessary comparisons in boolean expressions	723	77	9,39	46,95%
13040	Switch statements should have a default label	689	128	5,38	78,05%
13042	Avoid reassigning parameters such as <name>"	861	142	6,06	86,59%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	186	83	2,24	50,61%
13044	Avoid calls to overridable methods during construction	519	92	5,64	56,10%
13046	This final field could be made static	2614	123	21,25	75,00%
13049	Object clone() should be implemented with super.clone()	2	1	2,00	0,61%
13050	Non-static initializers are confusing	1	1	1,00	0,61%
13051	The default label should be the last label in a switch statement	2	2	1,00	1,22%
13053	Avoid empty finalize methods	3	1	3,00	0,61%
13056	Last statement in finalize method should be a call to super.finalize()	34	5	6,80	3,05%
13057	Explicit call of finalize method	15	3	5,00	1,83%
13058	If you override finalize(), make it protected	39	5	7,80	3,05%
13059	Avoid duplicate imports such as <package>"	53	18	2,94	10,98%
13060	Avoid importing anything from the package java.lang"	200	55	3,64	33,54%
13061	Avoid unused imports such as <package>"	100	38	2,63	23,17%
13062	No need to import a type thats in the same package'	98	5	19,60	3,05%
13071	Method name does not begin with a lower case character.	2249	95	23,67	57,93%
13072	Class names should begin with an uppercase character and not include underscores	56	18	3,11	10,98%
13073	Abstract classes should be named AbstractXXX"	122	67	1,82	40,85%
13075	Classes should not have non-constructor methods with the same name as the class	7	7	1,00	4,27%
13078	A signature (constructor or method) shouldnt have Exception in throws declaration'	539	43	12,53	26,22%
13080	The same String literal appears <number> times in this file; the first occurrence is on line <line number>	1172	144	8,14	87,80%
13081	Avoid instantiating String objects; this is usually unnecessary.	410	40	10,25	24,39%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13082	Avoid calling toString() on String objects; this is unnecessary	10	7	1,43	4,27%
13083	Avoid unused private fields such as <name>''	425	103	4,13	62,80%
13084	Avoid unused local variables such as <name>''	1109	147	7,54	89,63%
13085	Avoid unused private methods such as <name>''	108	55	1,96	33,54%
13086	Avoid unused formal parameters such as <name>''	40	28	1,43	17,07%
13087	Variables that are not final should not contain underscores.	790	136	5,81	82,93%
13088	Variables should start with a lowercase character	1834	137	13,39	83,54%
13089	Method names should not contain underscores	1688	146	11,56	89,02%
13090	Variables that are final and static should be in all caps.	124	43	2,88	26,22%
13091	Class names should not contain underscores	22	9	2,44	5,49%

A.8 Datos completos del análisis de bd4mod2

En la Tabla 30 se muestran los datos de todos los avisos detectados en el grupo bd4mod2, módulo 2 de la asignatura de Bases de Datos de tercer curso, ordenados por el número de errores que han sido detectados de cada tipo, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 30. Datos completos del análisis de bd4mod2

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13089	Method names should not contain underscores	924	15	61,60	68,18%
13091	Class names should not contain underscores	752	15	50,13	68,18%
13087	Variables that are not final should not contain underscores.	496	14	35,43	63,64%
13088	Variables should start with a lowercase character	324	10	32,40	45,45%
13078	A method shouldnt have Exception in throws declaration.'	269	15	17,93	68,18%
13071	Method name does not begin with a lower case character.	182	12	15,17	54,55%
13019	Avoid using "if...else" statements without curly braces	130	15	8,67	68,18%
13084	Avoid unused local variables.	113	13	8,69	59,09%
11023	This field is never read. Consider removing it from the class.	113	13	8,69	59,09%
13080	The same String literal appears several times in this file.	111	17	6,53	77,27%
38	duplicate class: <class name>	107	3	35,67	13,64%
13044	Avoid calls to overridable methods during construction	97	16	6,06	72,73%
13061	Avoid unused imports such as <package>"	58	13	4,46	59,09%
13059	Avoid duplicate imports such as <package>"	38	12	3,17	54,55%
13017	Avoid using "if" statements without curly braces	26	11	2,36	50,00%
11011	This method contains a redundant comparison of a reference value to null.	25	2	12,50	9,09%
13000	Avoid empty catch blocks	21	5	4,20	22,73%
13020	Avoid using "for" statements without curly braces	16	4	4,00	18,18%
13039	Avoid unnecessary comparisons in boolean expressions	15	5	3,00	22,73%
13083	Avoid unused private fields.	15	5	3,00	22,73%
13	")" expected	13	5	2,60	22,73%
11022	This field is never used. Consider removing it from the class.	13	7	1,86	31,82%
10013	May be wrong assumption about ELSE branch association	13	6	2,17	27,27%
13040	Switch statements should have a default label	12	5	2,40	22,73%
13036	All methods are static. Consider using Singleton instead.	7	5	1,40	22,73%
11006	Self assignment of field or local variable.	5	4	1,25	18,18%
13038	Avoid unnecessary if..then..else statements when returning a boolean	5	2	2,50	9,09%
10012	May be wrong assumption about IF body	5	3	1,67	13,64%
13072	Class names should begin with an uppercase character and not include underscores	5	2	2,50	9,09%
11014	Uninitialized read of field in constructor	4	3	1,33	13,64%
10011	May be wrong assumption about loop body	3	2	1,50	9,09%
13016	An empty statement (semicolon) not part of a loop	3	3	1,00	13,64%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
11007	Problems with the method finalize() use.	2	1	2,00	4,55%
13042	Avoid reassigning parameters.	2	1	2,00	4,55%
13056	Last statement in finalize method should be a call to super.finalize()	2	1	2,00	4,55%
13085	Avoid unused private methods.	2	2	1,00	9,09%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	2	2	1,00	9,09%
13058	If you override finalize(), make it protected	2	1	2,00	4,55%
13060	Avoid importing anything from the package java.lang"	2	2	1,00	9,09%
13090	Variables that are final and static should be in all caps.	2	1	2,00	4,55%
11002	Method might drop exception or ignore exception.	2	2	1,00	9,09%
11004	Comparison of String objects using "==" or "!="	2	1	2,00	4,55%
13073	Abstract classes should be named AbstractXXX"	1	1	1,00	4,55%
13081	Avoid instantiating String objects; this is usually unnecessary.	1	1	1,00	4,55%
11020	Unwritten field.	1	1	1,00	4,55%
34	package <package name> does not exist	1	1	1,00	4,55%
37	unreachable statement	1	1	1,00	4,55%

A.9 Datos completos del análisis de pl4

En la Tabla 31 se muestran los datos de todos los avisos detectados en el grupo pl4, de la asignatura de Procesadores de Lenguaje de cuarto curso, ordenados por el número de errores que han aparecido de cada tipo, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 31. Datos completos del análisis de pl4

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13019	Avoid using "if...else" statements without curly braces	427	4	106,75	80,00%
13017	Avoid using "if" statements without curly braces	350	4	87,50	80,00%
13087	Variables that are not final should not contain underscores.	262	2	131,00	40,00%
13071	Method name does not begin with a lower case character.	182	1	182,00	20,00%
13089	Method names should not contain underscores	98	4	24,50	80,00%
13088	Variables should start with a lowercase character	76	2	38,00	40,00%
13084	Avoid unused local variables.	64	3	21,33	60,00%
13090	Variables that are final and static should be in all caps.	57	4	14,25	80,00%
13080	The same String literal appears several times in this file.	50	4	12,50	80,00%
13081	Avoid instantiating String objects; this is usually unnecessary.	47	2	23,50	40,00%
13020	Avoid using "for" statements without curly braces	45	4	11,25	80,00%
13078	A method shouldnt have Exception in throws declaration.'	34	3	11,33	60,00%
11003	Innecesary calls to methods.	33	2	16,50	40,00%
13039	Avoid unnecessary comparisons in boolean expressions	30	1	30,00	20,00%
12005	Value of referenced variable may be NULL.	23	1	23,00	20,00%
13091	Class names should not contain underscores	21	1	21,00	20,00%
12001	Component in this class shadows one in base class.	21	3	7,00	60,00%
13000	Avoid empty catch blocks	20	2	10,00	40,00%
12014	Compare strings as object references.	18	2	9,00	40,00%
13001	Avoid empty if' statements'	18	2	9,00	40,00%
13061	Avoid unused imports such as <package>"	17	1	17,00	20,00%
13073	Abstract classes should be named AbstractXXX"	17	4	4,25	80,00%
12002	Local variable shadows component of class.	15	2	7,50	40,00%
13040	Switch statements should have a default label	11	4	2,75	80,00%
5	cannot resolve symbol symbol : method <a> location: class <B>	11	1	11,00	20,00%
11004	Comparison of String objects using "==" or "!="	9	2	4,50	40,00%
11022	This field is never used. Consider removing it from the class.	9	3	3,00	60,00%
13042	Avoid reassigning parameters.	8	4	2,00	80,00%
11023	This field is never read. Consider removing it from the class.	8	3	2,67	60,00%
13082	Avoid calling toString() on String objects; this is unnecessary	7	1	7,00	20,00%
13059	Avoid duplicate imports such as <package>"	7	2	3,50	40,00%
13018	Avoid using "while" statements without curly braces	7	1	7,00	20,00%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13036	All methods are static. Consider using Singleton instead.	6	3	2,00	60,00%
11015	Useless control flow statement	5	1	5,00	20,00%
13083	Avoid unused private fields.	4	1	4,00	20,00%
13009	Ensure you override both equals() and hashCode()	4	1	4,00	20,00%
13044	Avoid calls to overridable methods during construction	3	3	1,00	60,00%
13016	An empty statement (semicolon) not part of a loop	3	1	3,00	20,00%
13072	Class names should begin with an uppercase character and not include underscores	3	2	1,50	40,00%
13079	The catch clause shouldnt check the exception type - catch several exceptions instead'	2	1	2,00	20,00%
11002	Method might drop exception or ignore exception.	2	1	2,00	20,00%
11010	Problems than can cause a NullPointerException.	2	2	1,00	40,00%
11020	Unwritten field.	2	2	1,00	40,00%
12011	Comparison always produces the same result.	1	1	1,00	20,00%
12029	Index [<min>,<max>] may be out of array bounds.	1	1	1,00	20,00%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	1	1	1,00	20,00%
13060	Avoid importing anything from the package java.lang''	1	1	1,00	20,00%

A.10 Datos completos del análisis de pfc

En la Tabla 32 se muestran los datos de todos los avisos detectados en el grupo pfc: proyectos fin de carrera del último curso de la titulación de ingeniería en informática ordenados por el número de errores detectado, sólo aparecen en la tabla los avisos que tienen al menos una ocurrencia en uno de los proyectos.

Tabla 32. Datos completos del análisis de pfc

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13019	Avoid using "if...else" statements without curly braces	2352	4	588,00	44,44%
13017	Avoid using "if" statements without curly braces	1381	4	345,25	44,44%
13080	The same String literal appears several times in this file.	354	4	88,50	44,44%
13088	Variables should start with a lowercase character	221	4	55,25	44,44%
13061	Avoid unused imports such as <package>"	220	1	220,00	11,11%
13000	Avoid empty catch blocks	128	2	64,00	22,22%
13084	Avoid unused local variables.	119	4	29,75	44,44%
13081	Avoid instantiating String objects; this is usually unnecessary.	116	4	29,00	44,44%
13044	Avoid calls to overridable methods during construction	110	2	55,00	22,22%
13011	Avoid returning from a finally block	98	3	32,67	33,33%
6	cannot resolve symbol symbol : variable <a> location: class <B>	92	1	92,00	11,11%
13071	Method name does not begin with a lower case character.	88	2	44,00	22,22%
13039	Avoid unnecessary comparisons in boolean expressions	88	3	29,33	33,33%
13090	Variables that are final and static should be in all caps.	88	2	44,00	22,22%
13020	Avoid using "for" statements without curly braces	72	2	36,00	22,22%
13087	Variables that are not final should not contain underscores.	67	2	33,50	22,22%
13078	A method shouldnt have Exception in throws declaration.'	65	1	65,00	11,11%
13042	Avoid reassigning parameters.	54	3	18,00	33,33%
13059	Avoid duplicate imports such as <package>"	52	1	52,00	11,11%
13060	Avoid importing anything from the package java.lang"	47	1	47,00	11,11%
13040	Switch statements should have a default label	39	3	13,00	33,33%
13077	A catch statement should never catch throwable since it includes errors	35	1	35,00	11,11%
13083	Avoid unused private fields.	29	2	14,50	22,22%
12002	Local variable shadows component of class.	20	2	10,00	22,22%
13089	Method names should not contain underscores	19	1	19,00	11,11%
13016	An empty statement (semicolon) not part of a loop	17	1	17,00	11,11%
13001	Avoid empty if' statements'	16	1	16,00	11,11%
12014	Compare strings as object references.	15	2	7,50	22,22%
38	duplicate class: <class name>	13	1	13,00	11,11%
11003	Innecesary calls to methods.	12	2	6,00	22,22%
13018	Avoid using "while" statements without curly braces	12	1	12,00	11,11%

Código error	Descripción del error	Número errores	Núm. proyectos con errores	Media errores por proyecto	Porcentaje de proy errores
13047	Avoid instantiating Boolean objects; you can usually invoke Boolean.valueOf() instead.	11	1	11,00	11,11%
13073	Abstract classes should be named AbstractXXX"	10	1	10,00	11,11%
13036	All methods are static. Consider using Singleton instead.	8	2	4,00	22,22%
13079	The catch clause shouldnt check the exception type - catch several exceptions instead'	7	1	7,00	11,11%
13038	Avoid unnecessary if..then..else statements when returning a boolean	7	3	2,33	33,33%
13008	Avoid unnecessary temporaries when converting primitives to Strings	7	1	7,00	11,11%
11004	Comparison of String objects using "==" or "!="	6	2	3,00	22,22%
13013	Avoid unnecessary return statements	6	1	6,00	11,11%
13086	Avoid unused formal parameters.	6	2	3,00	22,22%
13009	Ensure you override both equals() and hashCode()	5	1	5,00	11,11%
13085	Avoid unused private methods.	5	2	2,50	22,22%
11011	Redundant comparison of a reference value to null.	4	2	2,00	22,22%
11023	This field is never read. Consider removing it from the class.	4	1	4,00	11,11%
12011	Comparison always produces the same result.	4	1	4,00	11,11%
13015	Do not use if' statements that are always true or always false'	3	1	3,00	11,11%
11025	This code seems to be using non-short-circuit logic (e.g., & or |) rather than short-circuit logic (&& or ||). Non-short-circuit logic causes both sides of the expression to be evaluated even when the result can be inferred from knowing the left-hand side	3	1	3,00	11,11%
13062	No need to import a type thats in the same package'	3	1	3,00	11,11%
34	package <package name> does not exist	3	1	3,00	11,11%
13091	Class names should not contain underscores	3	1	3,00	11,11%
5	cannot resolve symbol symbol : method <a> location: class <B>	2	1	2,00	11,11%
11026	This private method is never called. Although it is possible that the method will be invoked through reflection, it is more likely that the method is never used, and should be removed.	2	1	2,00	11,11%
27	.' expected'	2	1	2,00	11,11%
39	inconvertible types found : <type name> required: <type name>	2	1	2,00	11,11%
13043	A high ratio of statements to labels in a switch statement. Consider refactoring.	1	1	1,00	11,11%
1	class <class name> is public, should be declared in a file named <class name>.java	1	1	1,00	11,11%
13046	This final field could be made static	1	1	1,00	11,11%
4	cannot resolve symbol symbol : class <A> location: class <B>	1	1	1,00	11,11%
19	incompatible types found : <type found> required: <type required>	1	1	1,00	11,11%
12028	Inequality comparison can be replaced with equality comparison.	1	1	1,00	11,11%
13072	Class names should begin with an uppercase character and not include underscores	1	1	1,00	11,11%
11002	Method might drop exception or ignore exception.	1	1	1,00	11,11%

A.11 Página de Errores frecuentes completa

A continuación se incluye el informe generado por el Sistema de Análisis de Programas en el que se representan los 10 avisos más frecuentes para cada uno de los grupos de proyectos analizados. El formato de este informe está descrito en el apartado 7.9.2. Descripción del informe de estadísticas.

RESULTADOS DE LAS ESTADÍSTICAS

1 - Estadística de errores frecuentes

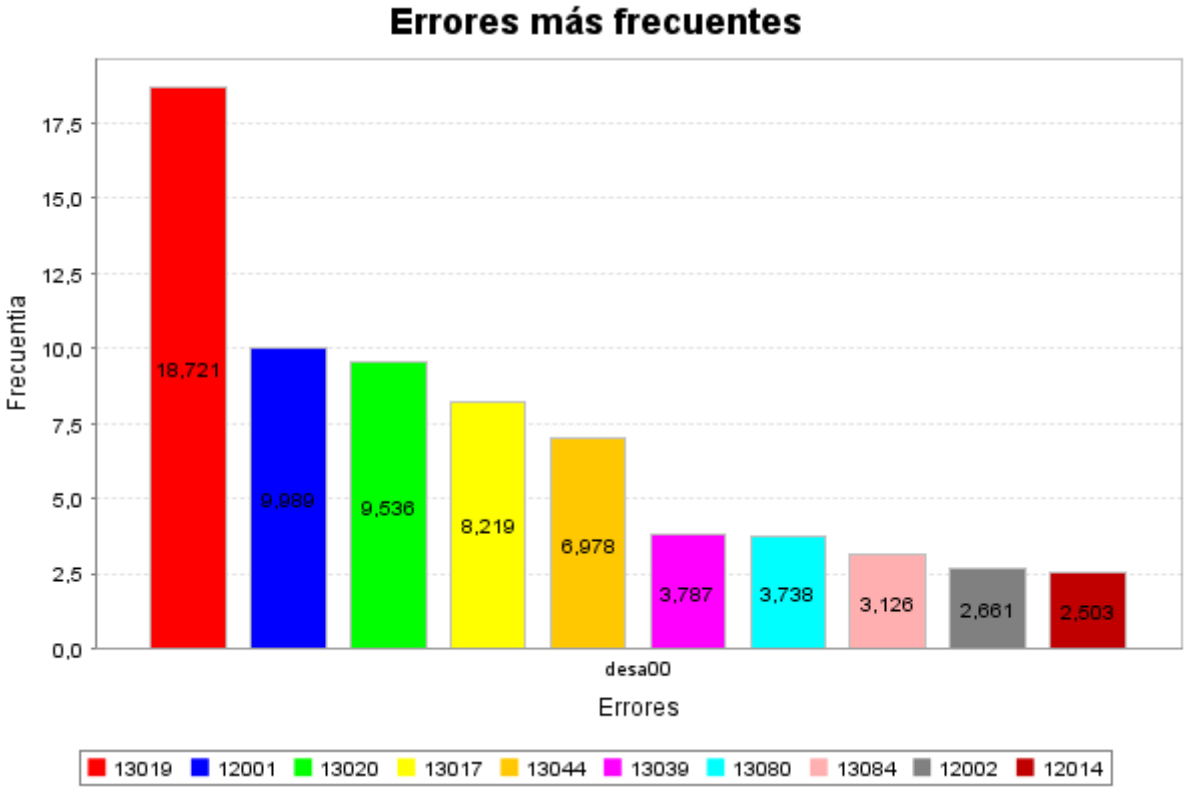
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24000 y acaba en la compilación 24182

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	12001	13020	13017	13044	13039	13080	13084	12002	12014
Frecuencia	18,721	9,989	9,536	8,219	6,978	3,787	3,738	3,126	2,661	2,503

Número de compilaciones : 183

2 - Estadística de errores frecuentes

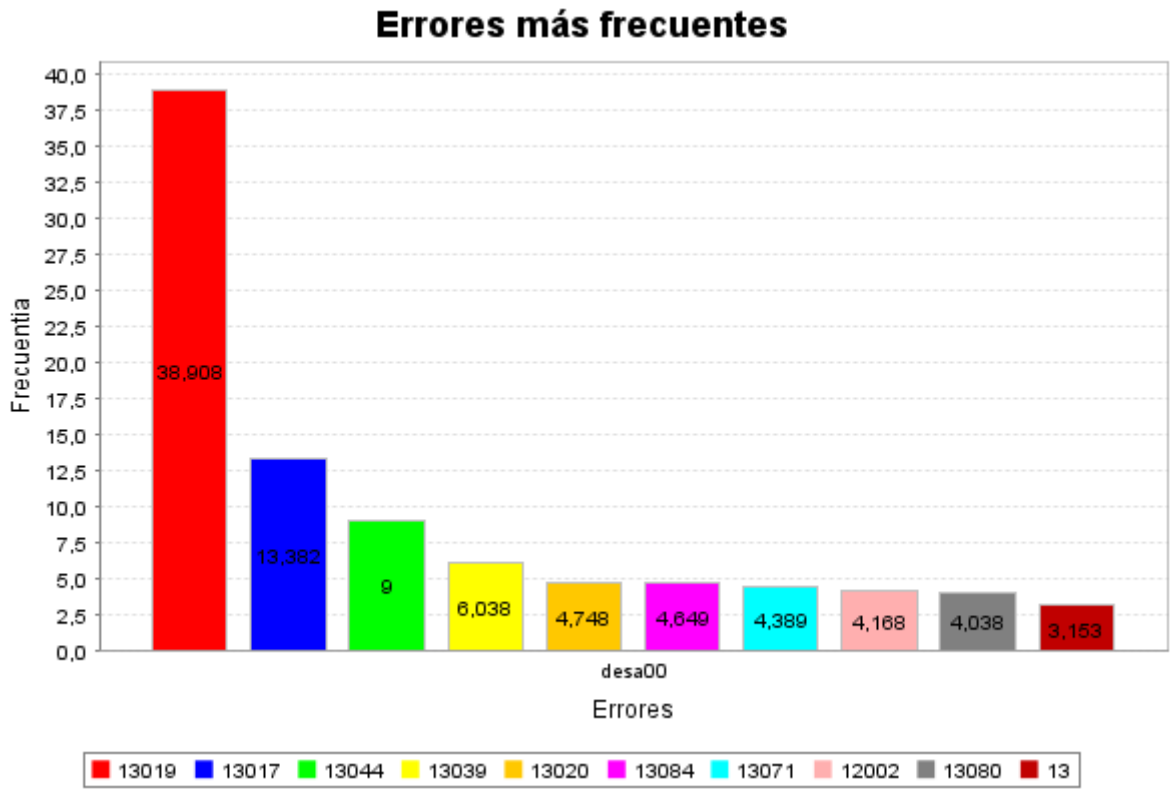
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24200 y acaba en la compilación 24330

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	13017	13044	13039	13020	13084	13071	12002	13080	13
Frecuencia	38,908	13,382	9,00	6,038	4,748	4,649	4,389	4,168	4,038	3,153

Número de compilaciones : 131

3 - Estadística de errores frecuentes

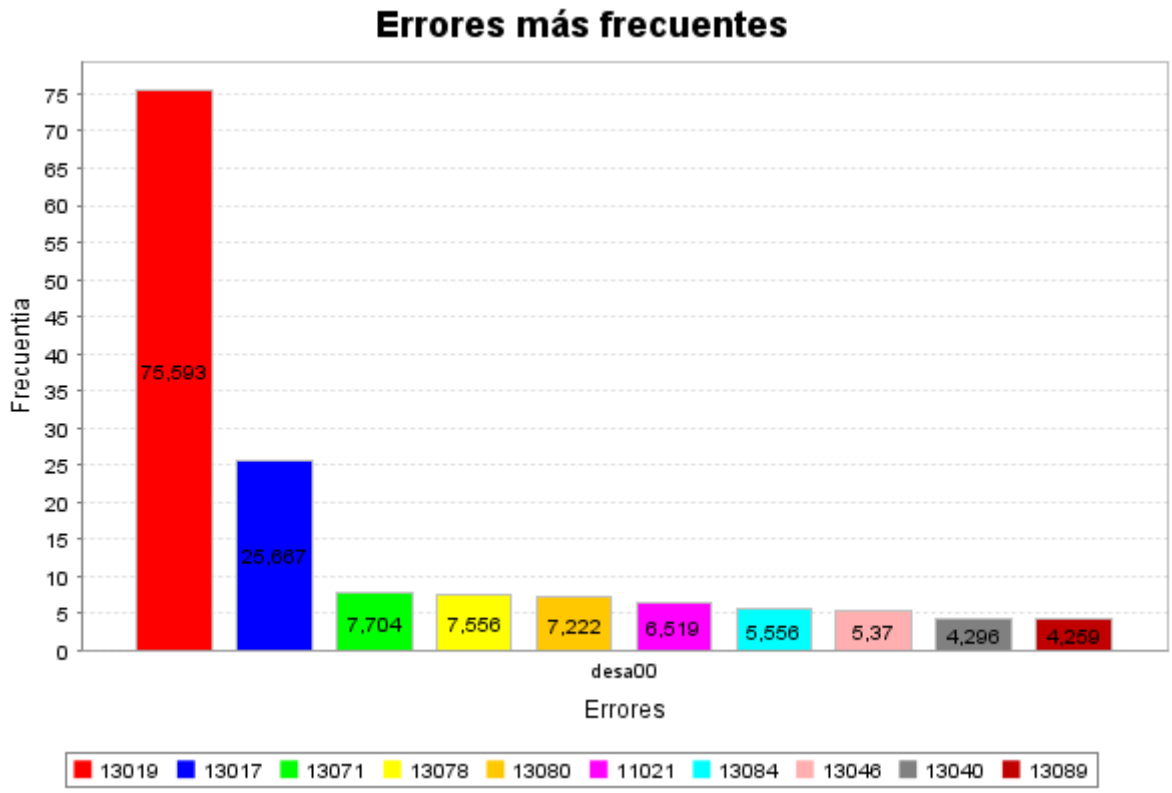
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24400 y acaba en la compilación 24426

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	13017	13071	13078	13080	11021	13084	13046	13040	13089
Frecuencia	75,593	25,667	7,704	7,556	7,222	6,519	5,556	5,37	4,296	4,259

Número de compilaciones : 27

4 - Estadística de errores frecuentes

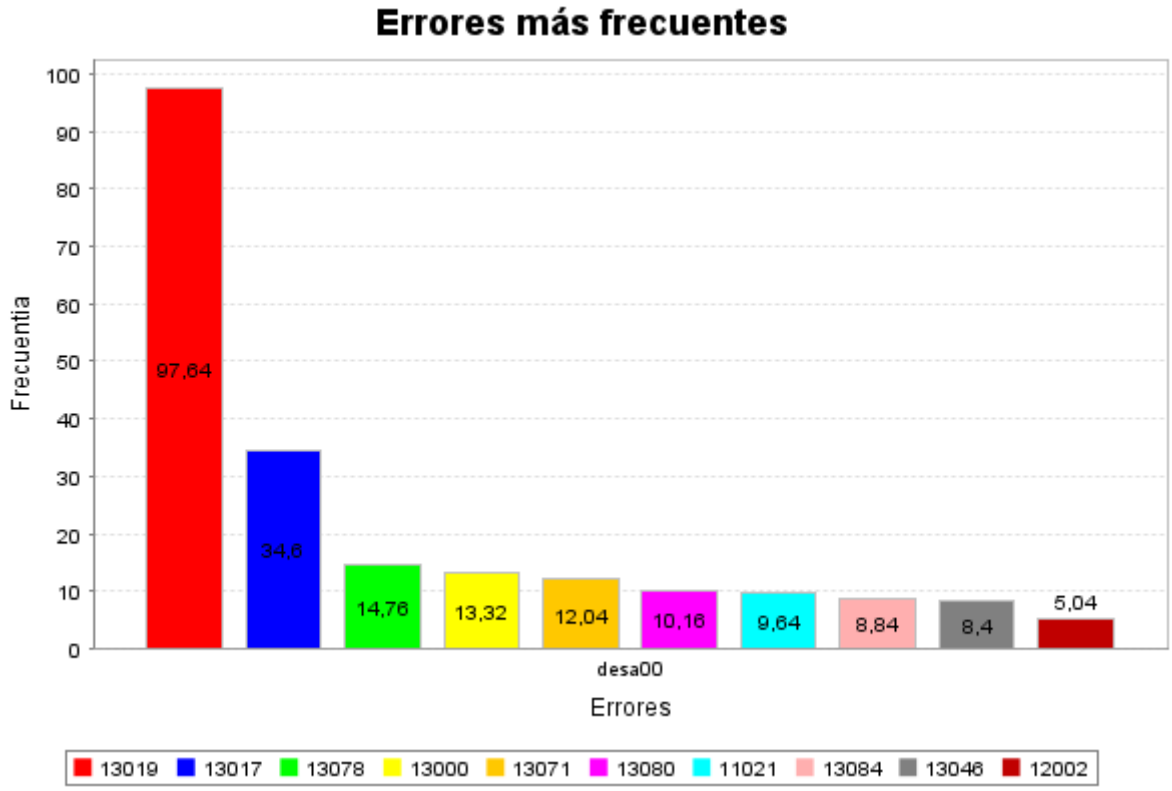
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24500 y acaba en la compilación 24524

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	13017	13078	13000	13071	13080	11021	13084	13046	12002
Frecuencia	97,64	34,60	14,76	13,32	12,04	10,16	9,64	8,84	8,40	5,04

Número de compilaciones : 25

5 - Estadística de errores frecuentes

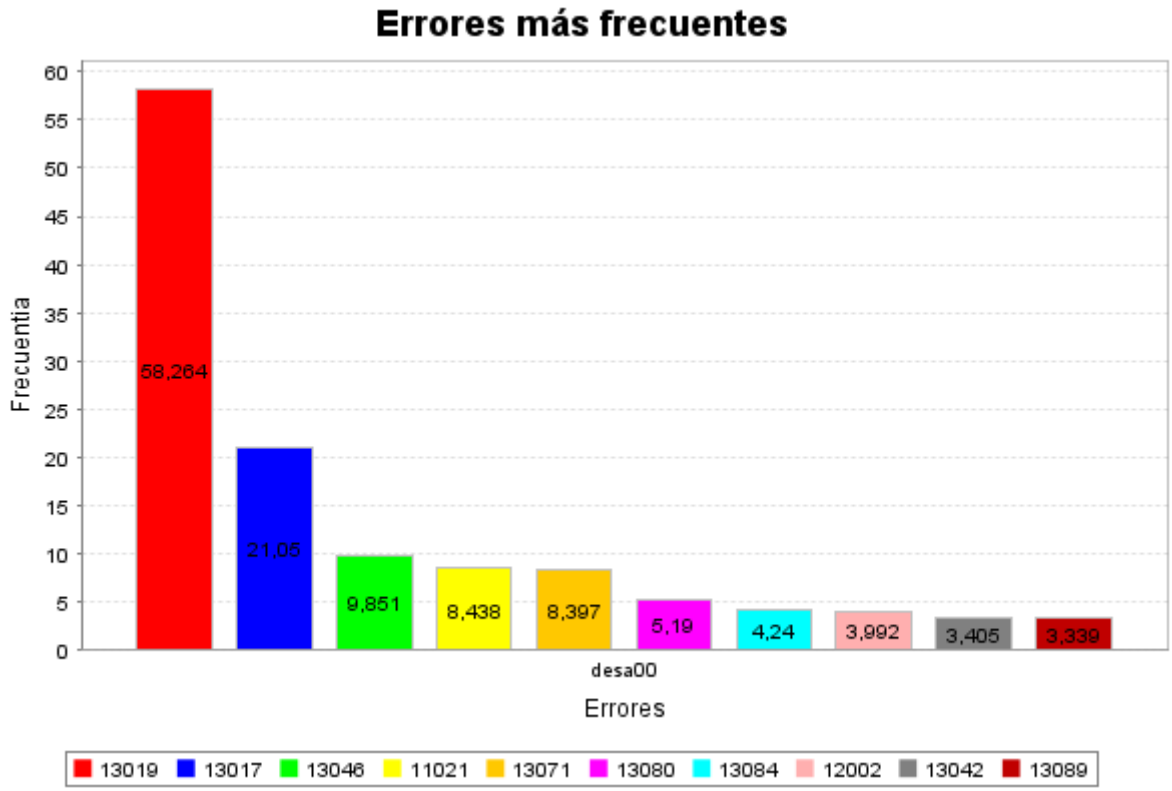
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24600 y acaba en la compilación 24720

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	13017	13046	11021	13071	13080	13084	12002	13042	13089
Frecuencia	58,264	21,05	9,851	8,438	8,397	5,19	4,24	3,992	3,405	3,339

Número de compilaciones : 121

6 - Estadística de errores frecuentes

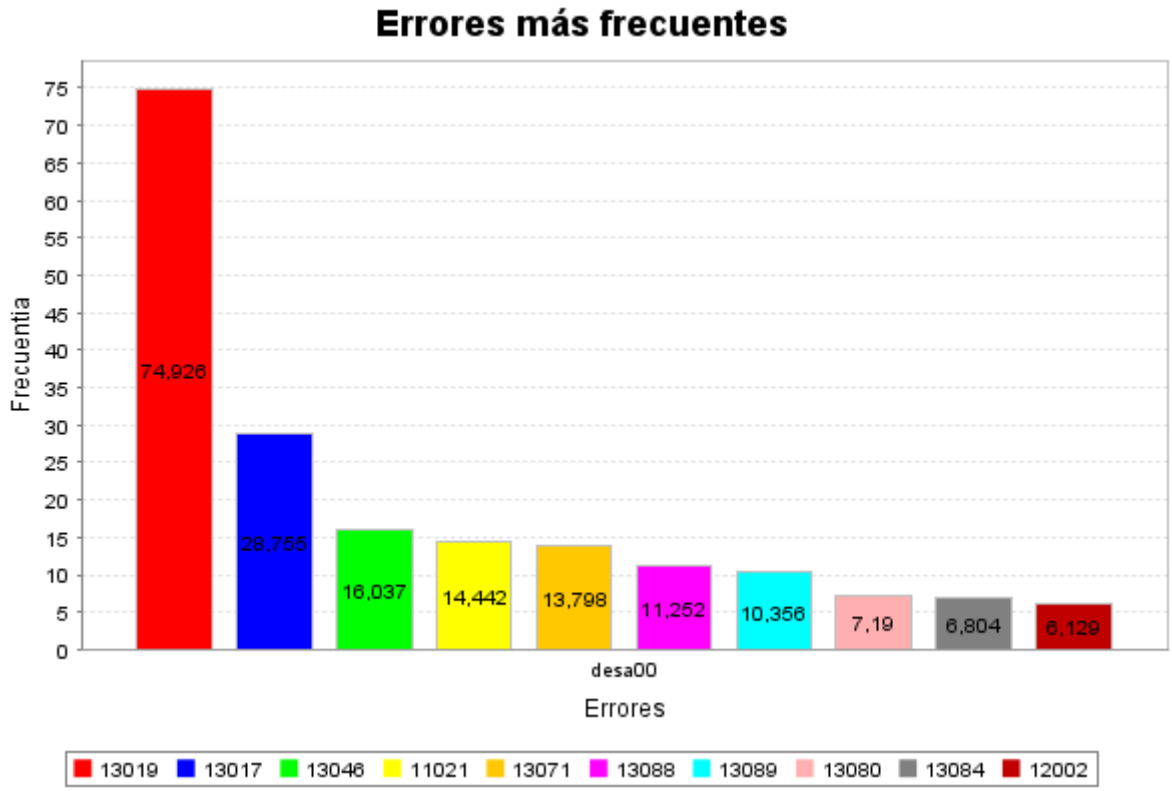
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24800 y acaba en la compilación 24963

Conjunto de Errores 1

Contenido : Todos los errores

Resultados para el usuario desa00



Código de Error	13019	13017	13046	11021	13071	13088	13089	13080	13084	12002
Frecuencia	74,926	28,755	16,037	14,442	13,798	11,252	10,356	7,19	6,804	6,129

Número de compilaciones : 163

A.12 Página errores frecuentes excluyendo avisos de incumplimiento de convenios de nombres y de código

A continuación se incluye el informe generado por el Sistema de Análisis de Programas en el que se representan los 10 avisos más frecuentes para cada uno de los grupos de proyectos analizados excluyendo aquellos avisos relacionados con incumplimiento de convenios para los nombres de paquetes, clases, métodos y variables; y convenios para la escritura del código fuente: indentación, uso de llaves, etc. Los avisos excluidos se incluyen en la Tabla 33. El formato de este informe está descrito en el apartado 7.9.2. Descripción del informe de estadísticas.

Tabla 33. Avisos eliminados en el análisis del apartado A.12

Código error	Descripción del error	Tipo de error
13019	Avoid using "if...else" statements without curly braces	Convenciones de código
13017	Avoid using "if" statements without curly braces	Convenciones de código
13071	Method name does not begin with a lower case character.	Convenciones de nombres
13020	Avoid using "for" statements without curly braces	Convenciones de código
13089	Method names should not contain underscores	Convenciones de nombres
13088	Variables should start with a lowercase character	Convenciones de nombres
13087	Variables that are not final should not contain underscores.	Convenciones de nombres
13091	Class names should not contain underscores	Convenciones de nombres
13073	Abstract classes should be named AbstractXXX"	Convenciones de nombres

RESULTADOS DE LAS ESTADÍSTICAS

1 - Estadística de errores frecuentes

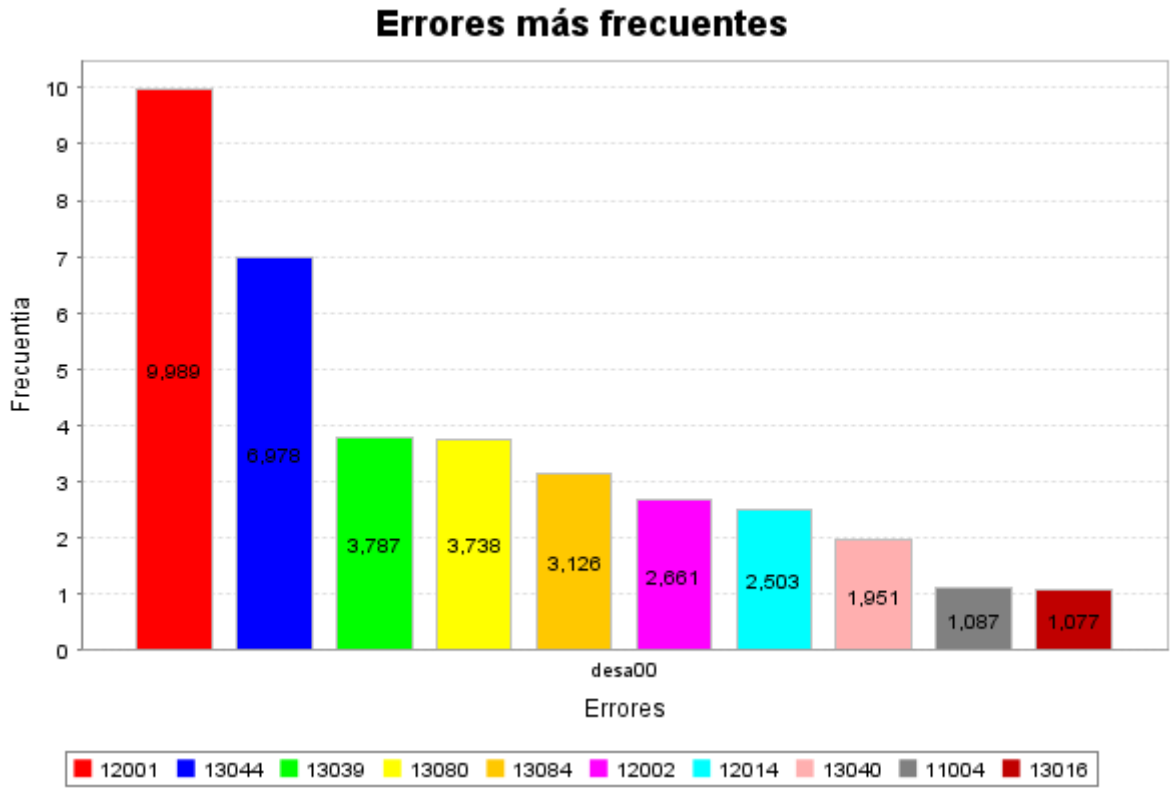
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24000 y acaba en la compilación 24182

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	12001	13044	13039	13080	13084	12002	12014	13040	11004	13016
Frecuencia	9,989	6,978	3,787	3,738	3,126	2,661	2,503	1,951	1,087	1,077

Número de compilaciones : 183

2 - Estadística de errores frecuentes

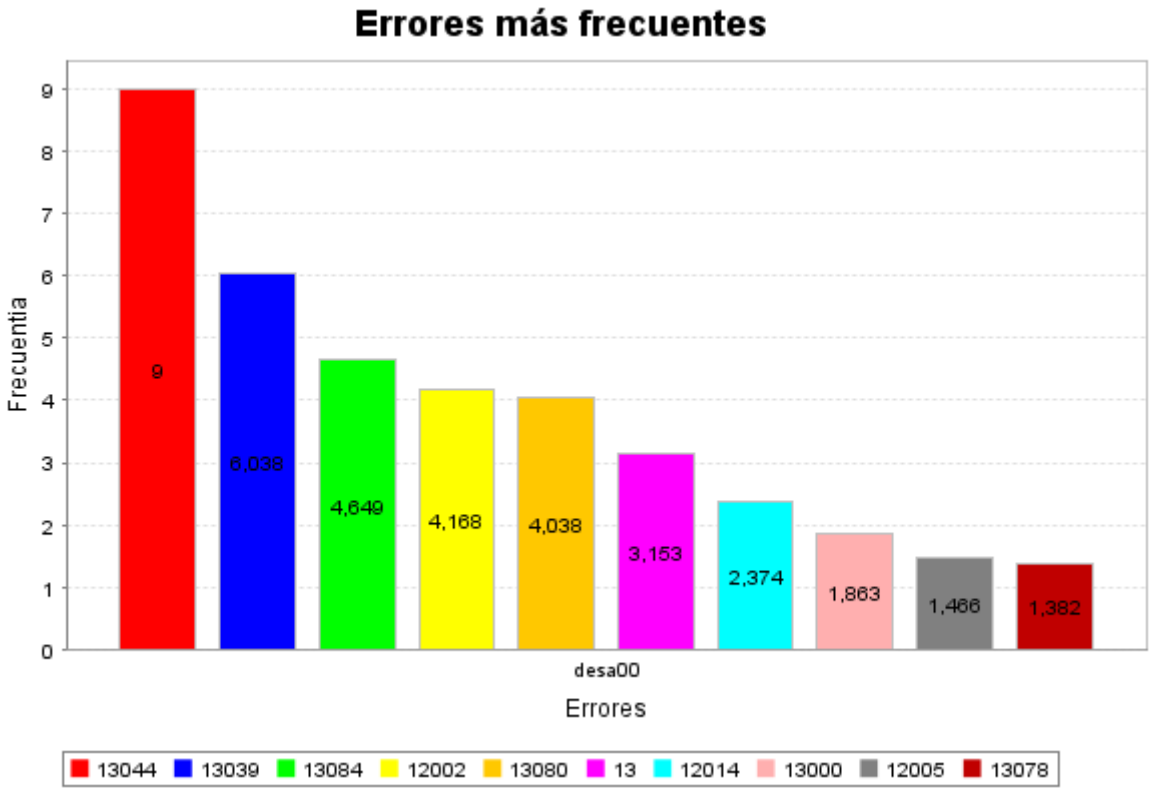
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24200 y acaba en la compilación 24330

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	13044	13039	13084	12002	13080	13	12014	13000	12005	13078
Frecuencia	9,00	6,038	4,649	4,168	4,038	3,153	2,374	1,863	1,466	1,382

Número de compilaciones : 131

3 - Estadística de errores frecuentes

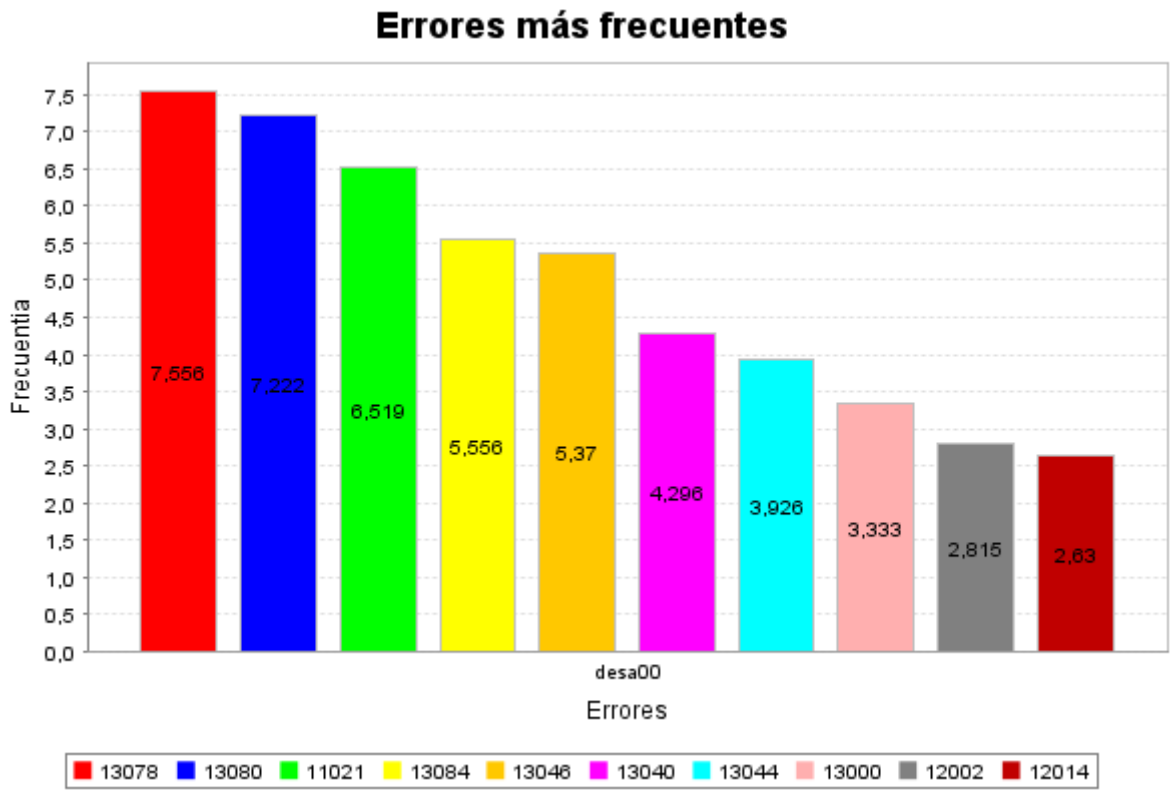
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24400 y acaba en la compilación 24426

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	13078	13080	11021	13084	13046	13040	13044	13000	12002	12014
Frecuencia	7,556	7,222	6,519	5,556	5,37	4,296	3,926	3,333	2,815	2,63

Número de compilaciones : 27

4 - Estadística de errores frecuentes

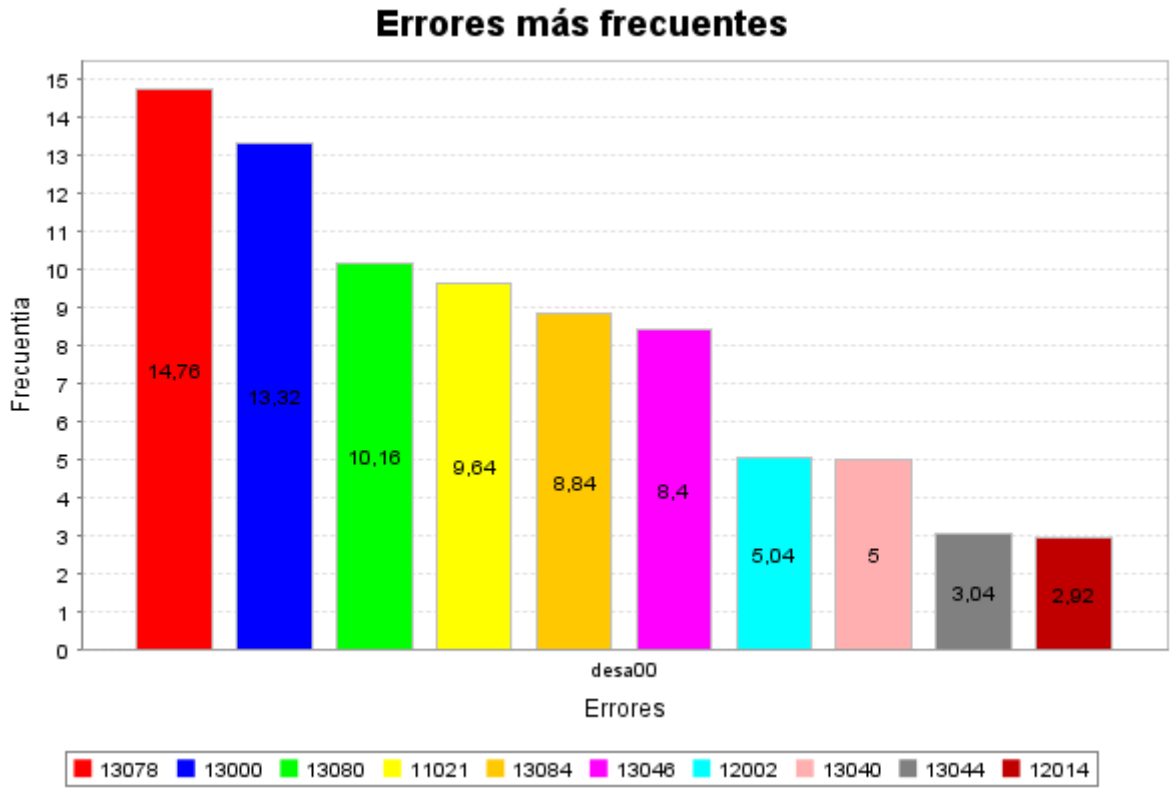
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24500 y acaba en la compilación 24524

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	13078	13000	13080	11021	13084	13046	12002	13040	13044	12014
Frecuencia	14,76	13,32	10,16	9,64	8,84	8,40	5,04	5,00	3,04	2,92

Número de compilaciones : 25

5 - Estadística de errores frecuentes

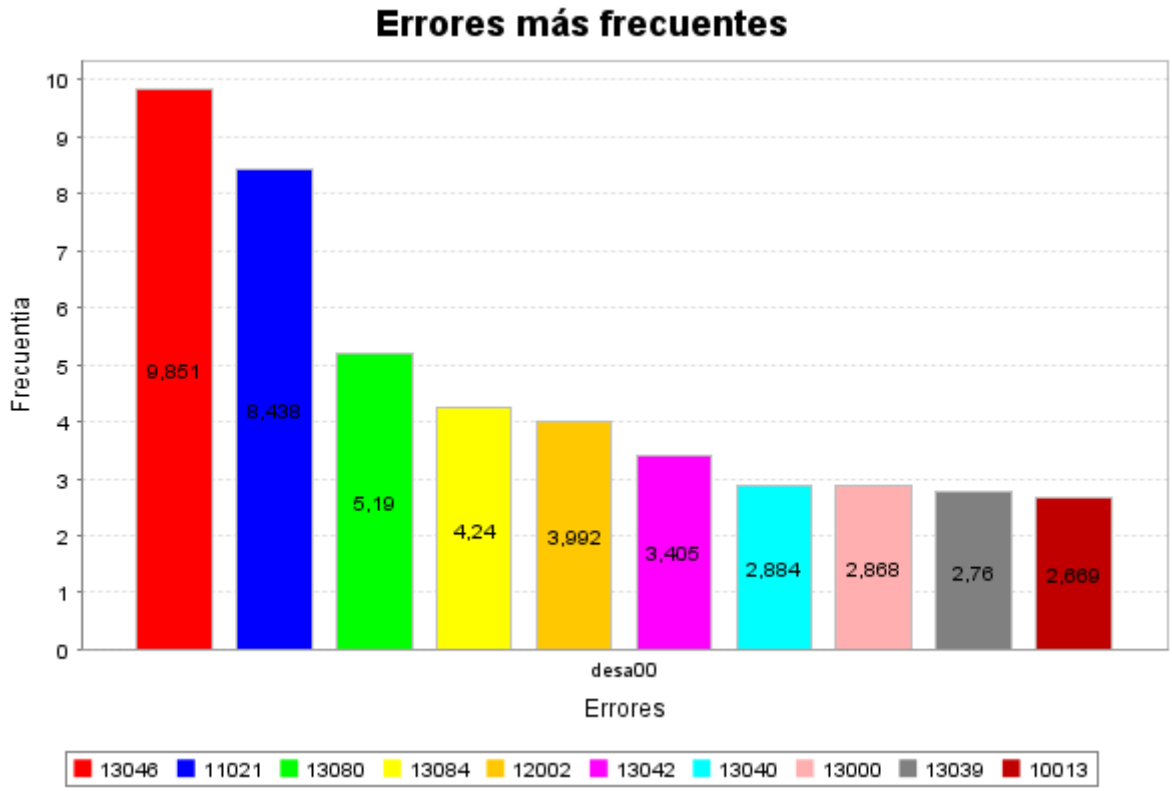
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24600 y acaba en la compilación 24720

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	13046	11021	13080	13084	12002	13042	13040	13000	13039	10013
Frecuencia	9,851	8,438	5,19	4,24	3,992	3,405	2,884	2,868	2,76	2,669

Número de compilaciones : 121

6 - Estadística de errores frecuentes

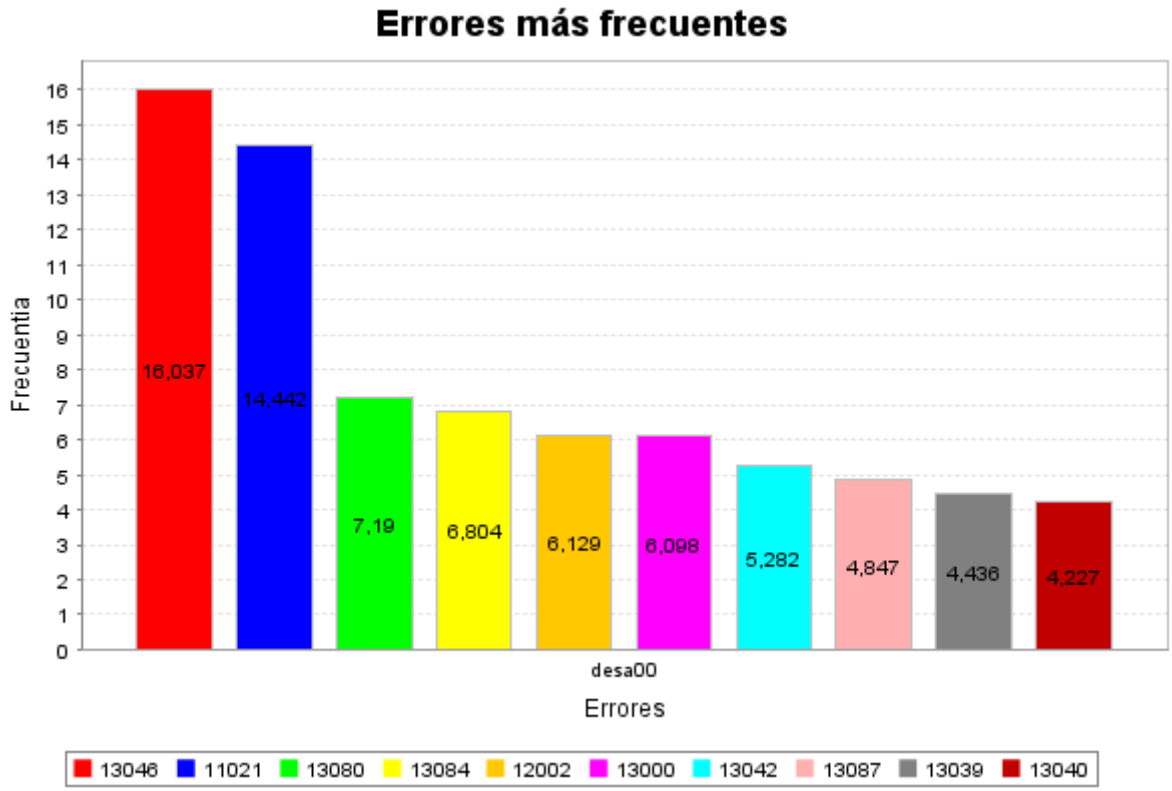
Aspectos generales de la estadística

Periodo : Empieza en la compilación 24800 y acaba en la compilación 24963

Conjunto de Errores 1

Contenido : Todos los errores que no cumplen que su código sea 13019 y que no cumplen que su código sea 13020 y que no cumplen que su código sea 13017 y que no cumplen que su código sea 13071 y que no cumplen que su código sea 13089 y que no cumplen que su código sea 13088

Resultados para el usuario desa00



Código de Error	13046	11021	13080	13084	12002	13000	13042	13087	13039	13040
Frecuencia	16,037	14,442	7,19	6,804	6,129	6,098	5,282	4,847	4,436	4,227

Número de compilaciones : 163

Apéndice D. Referencias

[Allen 2002] Allen, Eric; (2002) *Bug Patterns in Java*. Apress.

[ANECA 2005] ANECA; (2005) *Libro blanco del Título de grado en Ingeniería Informática*. (http://www.aneca.es/modal_eval/docs/libroblanco_informatica.pdf)

[Ant 2003] Apache Ant. The Apache Software Foundation. 2003. (http://ant.apache.org/)

[Arnold 1996] Arnold, Ken; Gosling, James (1996) The Java™ Programming Language. Addison Wesley.

[Artho 2001] ARTHO, Cyrille; (2001) *Finding faults in multi-threaded programs*. Master's thesis report.

[Barros 1999] Barros Blanco, Beatriz (1999) Aprendizaje Colaborativo En Enseñanza A Distancia: Entorno Genérico Para Configurar, Realizar Y Analizar Actividades En Grupo. Tesis doctoral Departamento de Inteligencia Artificial. Universidad Politécnica De Madrid.

[Barros 2000] B.Barros, M.F.Verdejo. "DEGREE: un sistema para la realización y evaluación de experiencias de aprendizaje colaborativo en enseñanza a distancia". (2000) Revista Iberoamericana de Inteligencia Artificial Vol 9 ISSN 1137-3601. Pages 27-37.

[Barros 2001] BARROS, Beatriz; Verdejo, M. F.; (2001) *Entornos para la realización de actividades de aprendizaje colaborativo a distancia*. Revista Iberoamericana de Inteligencia Artificial. ISSN 1137-3601.

[BCEL 2004] The Byte Code Engineering Library, Apache Software Foundation, 2004. (http://jakarta.apache.org/bcel/)

[Beck 2000] Beck, Ken. Extreme Programming Explained. Embrace Change. Pearson Education, Inc. Publishing as Addison Wesley Logran, 2000.

[Benford 1994] Benford, S. D., Burke, E. K., Foxley, E., Gutteridge, N., & Zin, A. M. (1994). Ceilidh as a Course Management Support System. Journal of Educational Technology Systems, 22, 235-250.

[Bergin 1997] Bergin, Joseph, Stehlik, M., Roberts, J., and Pattis, R., 1997: *Karel++ - A Gentle Introduction to the Art of Object-Oriented Programming*, John Wiley & Sons. (http://www.csis.pace.edu/~bergin/karel.html)

[Bergin 2000] Bergin, Joseph, 2000: "Introducing Objects with Karel J. Robot", Procs. of the Workshop "Tools and Environments for Understanding Object-Oriented Concepts", European Conference on Object-Oriented Programming.

[Bergin 2003] Bergin, Joseph, Stehlik, M., Roberts, J., and Pattis, R., 2003: *Karel J. Robot – A Gentle Introduction to the Art of Object-Oriented Programming in Java.* http://csis.pace.edu/%7Ebergin/KarelJava2ed/Karel++JavaEdition.html

[Boder 1992] Boder, A. "The process of knowledge reification in human-human interaction" Journal of Computer Assisted Learning, Vol. 8, No. 3, September, pp. 177-185. (1992)

[Boroni 2001] Boroni, C.M., Goosey, F.W., Grinder, M.T. and Ross, R.J., 2001: "Engaging Students with Active Learning Resources: Hypertextbooks for the Web", *Procs. of the 32nd SIGCSE Technical Symposium on Computer Science Education.*

[Boulay 1989] du Boulay, B., Some Difficulties Of Learning To Program, In Studying The Novice Programmer, Soloway, E., Sprohrer, J. (Eds.), Lawrence Erlbaum Associates, 1989, pp. 283-300.

[Boyle 1994] Boyle, T., Gray, J., Wendl, B., & Davies, M. (1994). Taking the plunge with CLEM: the design and evaluation of a large scale CAL system. Computers and Education, 22, 19-26.

[Bravo 2004] Bravo, Crescencio, Redondo, M.A., Ortega, M.; (2004) Aprendizaje en grupo de la programación mediante técnicas de colaboración distribuida en tiempo real. Actas del V Congreso Interacción Persona Ordenador, Lleida.

[Brown 1983] Brown, J. S. "Process versus product: a perspective on tools for communal and informal electronic learning", Report from the Learning: Education in the Electronic Age. (1983)

[Brusilovsky 1996] Brusilovsky, Peter, and Weber, G., 1996: "Collaborative example selection in an intelligent example-based programming environment", Procs. of the International Conference on Learning Sciences.

[Brusilovsky 1998] Brusilovsky, Peter, Calabrese, E., Hvorecky, J., Kouchnirenko, A. & Miller P., (1998) Mini-languages: a way to learn programming principles, Education and Information Technologies, 2, pp. 65 -83.

[Brusilovsky 2001] Brusilovsky, Peter, 2001: "WebEx: Learning from Examples in a Programming Course", Procs. of the World Conference of the WWW and Internet.

[Buck 2001] Buck, D., and Stucki, D.J., 2001: "JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum", *Procs. of th 32nd SIGCSE Technical Symposium on Computer Science Education.*

[Cavaness 2004] Chuck Cavaness 2004. Programming Jakarta Struts, 2nd Edition. O'Reilly Media, Inc. ISBN: 0596006519

[Cederqvist 1993] Per Cederqvist et al (1993) Version Management with CVS for cvs 1.11.2. Signum Support AB. (https://www.cvshome.org/docs/manual/)

[Champbell 2001] David A. Champbell y Tyler Jewwll (2001) Java Web Services. O'Reilly.

[Checkstyle] Página principal proyecto Checkstyle: http://checkstyle.sourceforge.net/

[Collis 1997] Collis, B. & Smith, C. "Desktop multimedia environments to support collaborative distance learning", Instructional Science, Vol. 25, nº 6, November, pp. 433-462. (1997)

[Cooper 2003] Cooper, S., Dann, W., and Pausch, R., 2003: "Teaching Objects-First in Introductory Computer Science", aceptado para 34th SIGCSE Technical Symposium on Computer Science Education (http://db.grinnel.edu/sigcse/sigcse2003/programAtaGlance.asp)

[Coward 2001] Coward, Danny (2001) Java™ Servlet Specification Version 2.3. Sun Microsystems, Inc. (http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html)

[Crescenzi 2000] Crescenzi, P., Demetrescu, C., Finocchi, I. and Petrschi, R., 2000: "Reversible execution and visualization of programs with Leonardo", *Journal of Visual Languages and Computing*, 11(2).

[Dann 2000] Dann, W., Cooper, S., and Pausch, R., 2000: "Making the Connection: Programming with Animated Small World", *Procs. of the Annual Conference on Innovation and Technology in Computer Science Education.*

[Dawson-Howe 1996] Dawson-Howe, Kenneth M.; (1996) *Automatic submission and administration of programming assignments.* SIGCSE Bull. 28, 2 (Jun. 1996) 40-42.

[Demetrescu 2000] Demetrescu, C. and Finocchi, I., 2000: "Smooth animation of algorithms in a declarative framework", *Journal of Visual Languages and Computing*, 12(3).

[Diaz 1996] Díaz, P., Catenazzi, N. and Aedo, I. (1996) De la multimedia a la hipermedia. Ra-Ma.

[DoctorJ] DoctorJ sitio Web: http://www.incava.org/projects/doctorj/

[DOM 1998] Document Object Model (DOM) Level 1 Specification. Version 1.0. W3C Recommendation 1 October, 1998

[Faries 1988] Faries, J.M., and Reiser, B.J., 1988: "Access an d use of previous solutions in a problem solving situation", *Procs. Of the 10th Annual Conference of th Coginitive Science Society*.

[Fernández 2003] Fernández González, Raúl; Paule Ruíz, María del Puerto; Pérez Pérez, Juan Ramón; González Rodríguez, Martín; González Gallego, Marcos (2003): Adaptable Contents Visulization (VIC). ICWE 2003: 167-170

[Flanagan 2002] C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for Java. In Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 234–245, Berlin, Germany, June 2002.

[Foster 2002] J. S. Foster, T. Terauchi, and A. Aiken. Flow-sensitive type qualifiers. In Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 1–12, Berlin, Germany, June 2

[Fowler 1993] Fowler, W. A. L., & Fowler, R. H. (1993). A hypertext-based approach to computer science education unifying programming principles. In T. Ottmann & I. Tomek (Eds.), Proceedings of ED-MEDIA'93 - World conference on educational multimedia and hypermedia. Charlottesville, VA: AACE. 203-209.

[Geis 1999] Jennifer Geis; (1999) JavaWizard User Guide. Collaborative Software Development Laboratory. Department of Information and Computer Sciences. University of Hawai. (http://csdl.ics.hawaii.edu/techreports/98-15/98-15.html)

[Gomez 2003] Gómez Albarrán, Mercedes; (2003) Una revisión de métodos pedagógicos innovadores para la enseñanza de la programación. Proc. Jornadas de Enseñanza Universitaria de la Informática (JENUI'03).

[González 2002] González Rodríguez, Martín; López Pérez, Benjamin; Paule Ruíz, María del Puerto; Pérez Pérez, Juan Ramón (2002): Dynamic Generation of Interactive Dialogs Based on Intelligent Agents. AH 2002: 564-567

[González 2004] González Rodríguez, Martín; Pérez Pérez, Juan Ramón; Paule Ruíz, María del Puerto (2004): Designing User Interfaces Tailored to the Current User's Requirements in Real Time. ICCHP 2004: 69-75

[Grindstaff 2004] Grindstaff, Chris; (2004) FindBugs, Part 2: Writing custom detectors. DeveloperWorks. IBM's resource for developers. (http://www-128.ibm.com/developerworks/java/library/j-findbug2/)

[Guzdial 1995] Guzdial, M., 1995: "Software-realized scaffolding to facilitate programming for science learning", *Interactive Learning Environments*, 4(1).

[Hitz 1997] Hitz, M. and Kögeler, S. Teaching C++ on the WWW. In Proc. ITiCSE'97, ACM Press, 11-13.

[Hovemeyer 2004a] Hovemeyer, David H.; (2004) FindBugs *Manual*. University of Maryland (http://findbugs.sourceforge.net/index.html).

[Hovemeyer, 2004b] Hovemeyer, David H.; Pugh, William; (2004). *Finding Bugs is Easy*. 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004).Vancouver, British Columbia, Canada.

[Huizinga 2001] Huizinga, Dorota M.; (2001) Identifying Topics for Instructional Improvement Through On-line Tracking of Programming Assignments. SIGCSE Bulletin 33, 3 (Sep. 2001) 129-132.

[Humphrey 1989] Humphrey, Watts S.; (1989) *Managing the Software Process*. Reading, MA: Addison-Wesley.

[Humphrey 1997] Humphrey, Watts S.; (1997) Introduction to the Personal Software Process, First Edition. 1ª edición. Pearson Education.

[Hundhausen 2002] Hundhausen, C., Douglas, S., and Stasko, J., 2002: "A Meta-Study of Algorithm Visualization Effectiveness", *Journal of Visual Languages and Computing*, 13(3)

[Jacobson 2002] Jacobson, Ivar. Call for expert systems. Application Development Trends Magazine. Número de junio de 2002. (http://www.adtmag.com)

[JCSC] Página principal del proyecto JCSC. http://jcsc.sourceforge.net

[JHA] JHAVÉ. http://csf11.acs.uwosh.edu/

[Johnson 1998] JOHNSON, P.M. (1998) Reengineering Inspection: The Future of Formal Technical Review. Communications of the ACM, vol. 41, pp. 49-52.

[Jones 1998] Jones, Capers; (1997) *The Impact of Poor Quality and Canceled Project son the Software Labor Shortage*. Informe técnico, Software Productivity Research, Inc.

[JUnit 2002] Kent Beck, Erich Gamma. (2002) JUnit Cookbook. JUnit 3.8. (http://junit.sourceforge.net/doc/cookbook/cookbook.htm)

[Knizhnik 2003] KNIZHNIK, Konstantin y Artho, Cyrille; (2003) Jlint manual. Java Program Checker (http://artho.com/jlint/).

[Kölling 1999a] Kölling, M. (1999a). The Problem of Teaching Object-Oriented Programming, Part 2: Environments. Journal of Object-Oriented Programming, 11(9), 6-12.

[Kölling 1999b] Kölling, M. (1999b). Teaching Object Orientation with the Blue Environment. Journal of Object-Oriented Programming, 12(2), 14-23.

[Kölling 1999c] Kölling, Michael (1999). The design of an object-oriented environment and language for teaching. Dissertation of DPh. Department of Computer Science, University of Sydney

[Kölling, 2003] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., (2003) The BlueJ system and its pedagogy, Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, Vol 13, No 4, Dec 2003.

[Koschmann 1996] Koschmann, T. (Editor) "CSCL: Theory and Practice of an emerging paradigm". Lawrence Erlbaum (1996).

[Labra 2003] Labra Gayo, José Emilio; Morales Gil, J. M.; Fernández Álvarez, A. M.; Sagastegui Chigne, H. (2003): A generic e-learning multiparadigm programming language system: IDEFIX project. SIGCSE 2003: 391-395

[LEO] Leonardo. http://www.cc.gatech.edu/gvu/softviz/algoanim/

[Leuf 2001] Bo Leuf, Ward Cunningham. (2001) The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley.

[Luján-Mora 2003] Luján-Mora, Sergio; (2003) Un enfoque para la enseñanza de la depuración de errores en las asignaturas de programación. Actas de IX Jornadas de

Enseñanza Universitaria de la Informática. Jenui 2003, pp 473-480. Thomson Paraninfo, S.A., Madrid, 2003.

[Martínez-Unanue 2002] R. Martínez-Unanue, M. Paredes-Velasco, C. Pareja-Flores, J. Urquiza-Fuentes and J. Á. Velázquez-Iturbide. (2002) Electronic books for programming education: A review and future prospects. 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)

[MEC 2003] Ministerio de Educación y Ciencia. (2003) Aplicación de Nuevas Tecnologías en el ámbito universitario. (http://wwwn.mec.es/univ/jsp/plantilla.jsp?id=2149)

[Meyerowitz 1995] Meyerowitz, J. (1995). PasTIL: a multiple-environment interactive Pascal tutor. In H. Maurer (Ed.), Proceedings of ED-MEDIA'95 - World conference on educational multimedia and hypermedia. Charlottesville, VA: AACE. 786.

[Naps 2000] Naps, T., Eagan, J., and Norton, L., 2000: "JHAVÉ – An Environment to Actively Engage Students in Web-based Algorithm Visualization", *Procs. of th 31st SIGCSE Technical Symposium on Computer Science Education.*

[Neal 1989] Neal, L.R., 1989: "A system for example-based programming", *Procs. of Human Factors in Computing Systems.*

[Nielsen 1999] Nielsen, Jakob (1999) Designing Web Usability : The Practice of Simplicity. New Riders Press.

[Nosek 1998] NOSEK, J.T. (1998) The Case for Collaborative Programming. Communications of the ACM, Vol. 41, No. 3 (March), pp. 105-108.

[Pattis 1981] Pattis, R., 1981: *Karel the Robot*, John Wiley & Sons.

[Paule 2003] Paule Ruíz, María del Puerto; Ocio Barriales, Sergio; Pérez Pérez, Juan Ramón; González Rodríguez, Martín (2003): Feijoo.net: An Approach to Personalized E-learning Using Learning Styles. ICWE 2003: 112-115

[Pelegrí-Llopart 2001] Pelegrí-Llopart, Eduardo (editor) (2001) JavaServer Pages™ Specification Version 1.2. Sun Microsystems, Inc. (http://www.jcp.org/aboutJava/communityprocess/final/jsr053/)

[Pérez 2003a] Pérez Pérez, Juan Ramón; Paule Ruíz, María del Puerto; González Rodríguez, Martín. "Development Web Environment" for Learing Programming Languages. ICWE 2003, LNCS 2722, pp. 128-129, 2003. Springer, Berlin 2003.

[Pérez 2003b] Pérez Pérez, Juan Ramón; Paule Ruiz, Mª del Puerto; González Rodríguez, Martín. Entorno web de desarrollo para el aprendizaje de paradigmas de programación. Actas de IX Jornadas de Enseñanza Universitaria de la Informática. Jenui 2003, pp 465-471. Thomson Paraninfo, S.A., Madrid, 2003.

[Pérez 2004a] Pérez Pérez, Juan Ramón; Gonzalez Rodríguez, Martín; Paule Ruiz, María del Puerto. A development environment for cooperative programming. En Actas del V Congreso Interacción Persona Ordenador. Interacción 2004. Seminario de Doctorado. Mayo de 2004.

[Pérez 2004b] Pérez Pérez, Juan Ramón; Iglesias Suárez, Mª Cristina; Paule Ruiz, Mª del Puerto; (2004) SACODE: Sistema de Aprendizaje Colaborativo de la Programación. VI Simposio Internacional de Informática Educativa (SIIE 2004).

[Pérez 2004c] Pérez Pérez, Juan Ramón; Rodríguez Fernández, Daniel; González Rodríguez, Martín; (2004) ¿Es posible la eliminación de los errores de los programas? VI Simposio Internacional de Informática Educativa (SIIE 2004).

[phpwiki] Página de PhpWiki: http://phpwiki.sourceforge.net/

[PMD] Página del proyecto PMD: http://pmd.sourceforge.net/.

[Pressman 1997] Pressman, Roger S.; (1997) *Software Engineering: A Practitioner's approach*. Cuarta edición. McGraw-Hill.

[Redondo 2002] Redondo, M.A., Bravo, C., Ortega, M. & Verdejo, M.F.: PlanEdit: An adaptive tool for design learning by problem solving. En: De Bra, P., Brusilovsky, P. & Conejo, R. (eds.) Adaptive Hypermedia and Adaptive Web-Based Systems. Springer Verlag. Lecture Notes in Computer Science. Berlin (2002) 29-31

[ROB] Página official de Robocode. http://robocode.sourceforge.net/?Open&ca=daw-prod-robocode

[Rößling 2000a] Rößling, Guido, and Freisleben, B., 2000: "Experiences in Using animations in Introductory Computer Science Lectures", *Procs. of the 31st SIGCSE Technical Symposium on Computer Science Education*.

[Rößling 2000b] Rößling, Guido, Schüler, M. and Freisleben, B., 2000: "The ANIMAL Algorithm Animation Tool", *Procs. of the Annual Conference on Innovation and Technology in Computer Science Education*.

[Rößling 2000c] Rößling, Guido. ANIMAL User Guide. University of Siegen. 2000. http://www.animal.ahrgr.de/index.php3

[Rößling 2001] Rößling, Guido, and Freisleben, B., 2001: "The Extensible Visualization Framework ANIMAL", *Procs. Of the 32nd SIGCSE Technical Symposium on Computer Science Education*.

[Rößling, 2002] Rößling, Guido, and Naps, T., 2002: "A Testbed for Pedagogical Requirements in Algorithm Visualization", *Procs. of the Annual Conference on Innovation and Technology in Computer Science Education*.

[Rubio 2003] Enrique Rubio Royo, Domingo J. Gallego, Catalina Alonso García. e-Learning en la formación a distancia y en los nuevos contextos corporativos. NOVATICA nº 165. Septiembre-octubre 2003.

[Rutar 2004] Rutar, Nick; Almazan, Christian; Foster, Jeff (2004) *A Comparison of Bug Finding Tools for Java*. The 15th IEEE International Symposium on Software Reliability Engineering (ISSRE'04). Saint-Malo, Bretagne, France.

[Sama 2003] SAMA VILLANUEVA, Sergio; PÉREZ PÉREZ, Juan Ramón; OCIO BARRIALES, Sergio; GONZÁLEZ RODRÍGUEZ, Martín; (2003) DSTool: A Reflection-based debugger for Data Structures Comprehension in Computing

Science Learning. p. 1026 - 1030. Human - Computer Interaction: Theory and Practice (Part I). Volume 1. JACKO, Julie; STEPHANIDIS, Constantine (Eds.). LEA: Lawrence Erlabaum Associates, Publishers. Mahwah, New Jersey. ISBN 0-8058-4930-0

[Sama 2005] Sama Villanueva, Sergio; (2005) DSTOOL: Herramienta para la Programación con Estructuras de Datos. Proyecto Fin de Carrera de la Escuela Politécnica Superior de Ingeniería de Gijón. Nº 1042036.

[Sapsomboon, 2000] Sapsomboon, Bordin (2000) *Shared Defect Detection : The Effects of Annotations in Asynchronous Software Inspection.* Doctoral Dissertation (http://www.sis.pitt.edu/~cascade/bordin/).

[Satratzemi 2001] Satratzemi, Maria; Dagdilelis, Vassilios; Evagelidis, Georgios (2001) A system for program visualization and problem-solving path assessment of novice programmers. ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education.

[Scotts 2003] Scotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D., Jackson, A., Virtual Teaming: Experiments and Experiences with Distributed Pair Programming, Extreme Programming/Agile Universe 2003

[Seffah 1999] Seffah, Ahmed; Bari, Moncef; Desmaris, Michel (1999) Assessing Object-Oriented Technology Skills Using an Internet-Based System. . ITiCSE '99: Proceedings of the 4th annual conference on Innovation and technology in computer science education.

[Shen 2000] Shen, H. & Sun, C.; (2000) RECIPE: a prototype for Internet-based real-time collaborative programming. Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems in conjunction with ACM CSCW Conference, December 2-6, Philadelphia, Pennsylvania, USA.

[Stage3 2005] Alice. Learn to Program Interactive 3D Graphics. Stage3 Research Group. Carnegie Mellon University. 2005. http://www.alice.org/

[Stasko 1990] Stasko, J., 1990: "TANGO: A Framework and System for Algotirm Animation", *Computer*, 23(9).

[Stasko 1998] Stasko, J., Domingue, J., Price, B.A., Brown, M.H., 1998: *Software Visualization Programming as a Multimedia Experience*, MIT Press.

[Schwarz 1996] Schwarz, E., Brusilovsky, P. and Weber, G. World-wide intelligent textbooks. In Proc. ED-TELECOM'96 (1996), AACE, 302-307.

[Verdejo 1998] Verdejo, M.F. & Barros, B. "Creating an organizational learning memory for collaborative experiences in distance education" en Teleteaching´98, pp.1035-1046. (1998) (http://sensei.ieec.uned.es/~steed/papers/verdejo-tt98.pdf)

[WebCT 2003] WebCT, Inc. (2003) *Whitepaper: Learning Without Limits*. www.webct.com.

[Wessner 1999] Wessner, M., Hans-Rüdiger, P. & Miao, Y. (1999) Using Learning Protocols to Structure Computer- Supported Cooperative Learning. Proceedings of

the ED-MEDIA'99. World Conference on Educational Multimedia, Hypermedia & Telecommunications, pp. 471-476, Seattle, Washington, June 19-24.

[Williams 2000] WILLIAMS, L.A. & Kessler, R.R. (2000) All I really need to know about pair programming learned in kindergarten. Communications of the ACM, BOl. 43, No. 5.

[Williams 2001] WILLIAMS, L. & Upchurch, R.L. (2001) In Support of Student Pair-Programming. ACM SIGCSE Conference for Computer Science Educators, February.

[Xerces 2004] Xerces. The Apache Software Foundation. 2004. (http://xml.apache.org/xerces2-j/)

[Xie 2002] Y. Xie and D. Engler. Using redundancies to find errors. In Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering, November 2002.

[XML 2004] Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 04 February 2004 (http://www.w3.org/TR/2004/REC-xml-20040204/)

[XML Schema 2004] XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004 (http://www.w3.org/TR/xmlschema-0/)

[XTA] XTango. http://www.cc.gatech.edu/gvu/softviz/algoanim/

[Zeller 2000] Zeller, Andreas; (2000) *Making Students Read and Review Code*. 5th ACM SIGCSE/SIGCUE Annual Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, July 2000