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.