Plugin de Eclipse: Proceso Dinámico de Gestión de Defectos de Diseño mediante Métricas

Carlos López[1], Yania Crespo[2], Raúl Marticorena[1], y Esperanza Manso[2]

[1] Universidad de Burgos, EPS Edificio C, C/Francisco Vitoria s/n, Burgos, España,
clopezno@ubu.es, rmartico@ubu.es,
[2] Universidad de Valladolid, Campus Miguel Delibes, Valladolid, España,
yania@infor.uva.es, manso@infor.uva.es

Resumen La capacidad de gestionar defectos de diseño en el software puede ayudar a mejorar la calidad y a hacer más productivas las tareas de desarrollo y de mantenimiento. En sistemas reales, donde existe una gran cantidad de entidades de código, un enfoque de detección de defectos de diseño basado en la intuición y experiencia humana es inabordable. La detección de defectos debería ser asistida por un proceso que estructure una inspección semi-automática. Actualmente se cuenta con herramientas que implementan la detección de algunos defectos de diseño catalogados. No obstante, cuando un inspector realiza una inspección de defectos de diseño, existe una componente de subjetividad que debe ser gestionada en el proceso. Las herramientas disponibles actualmente no tienen en cuenta lo anterior. En este artículo se presenta una herramienta desarrollada como plugin para Eclipse que automatiza un proceso dinámico de gestión de defectos de diseño basado en métricas de código sobre un conjunto de entidades. El dinamismo de la definición de defectos se gestiona permitiendo al inspector añadir o eliminar entidades de código al conjunto inicial utilizado para predecir, generando nuevos clasificadores con herramientas de minería de datos.

Palabras clave: mantenimiento del software, defectos de diseño, métricas de código, algoritmo de clasificación J48

1. Introducción

El desarrollo y mantenimiento del software requiere preservar la calidad de los sistemas a lo largo de todas las actividades del proceso. En metodologías ligeras como *XP (eXtreme Programming)* [1], las operaciones de mantenimiento asociadas con la incorporación de una nueva funcionalidad al sistema, están dirigidas por la detección defectos sobre las entidades del sistema afectadas. Estos defectos, no hacen referencia directa a fallos del sistema en ejecución, sino a malas características de sus propiedades estructurales, que a su vez, en operaciones de mantenimiento pueden derivar en fallos del sistema. Un ejemplo claro de este escenario es el que ocurre con el defecto *ciclos entre componentes* [2] creados vía relaciones de uso, relaciones de herencia o una combinación de ambas. Como tal

no es un fallo del sistema pero afectan a atributos de calidad del componente: facilidad de comprensión, facilidad de prueba, reutilización y desarrollo paralelo.

La naturaleza de los defectos de código puede ser muy variada y compleja, lo cuál influye considerablemente en los posibles indicadores utilizados para su detección. En [3] se presenta un conjunto de posibles fuentes de información que pueden ayudar. El proceso de detección de cada defecto utiliza una combinación de indicadores que son extraídos de alguna de estas fuentes.

Para aceptar finalmente la existencia de un defecto en una determinada entidad, el auditor aplica sus conocimientos particulares. La subjetividad del proceso está asociada al tipo de conocimiento declarativo (heurísticas, patrones de diseño, mejores prácticas, principios...) [4] que tiene en cuenta para tomar la decisión final. Esta situación hace que los resultados en el proceso de detección puedan variar dependiendo del auditor que lo lleve a cabo.

En lo que sigue el artículo tiene la siguiente estructura. En la sección 2 se especifica el proceso dinámico de detección de defectos que se automatiza con el plugin de Eclipse presentado. Posteriormente, en la sección 3 se describe el prototipo de la herramienta software indicando aspectos de integración, junto con las métricas y defectos tratados. En la última, sección 4 se concluye y se proponen líneas de trabajo futuras.

2. Proceso de dinámico de gestión de defectos

Esta sección describe brevemente un proceso dinámico de gestión de defectos de diseño mediante métricas. En la Figura 1 se presenta una visión global del proceso: participantes, tareas y productos. La principal fortaleza del proceso es la incorporación de la tarea (validación) para gestionar la subjetividad asociada a la identificación de defectos a través de técnicas de clasificación de minería de datos [5].

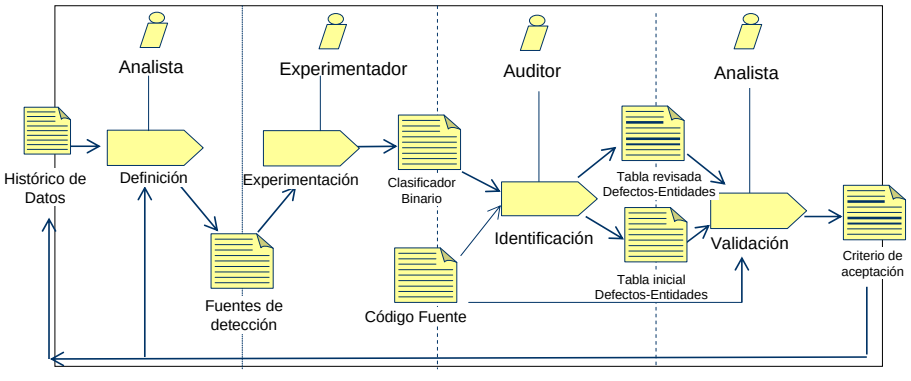

Figura 1: Proceso de dinámico de gestión de defectos

A continuación se describen las tareas principales del proceso:

- *Definición*, los defectos se definen como una expresión lógica formada con una composición de operadores lógicos, operadores aritméticos de comparación, métricas de código, valores umbrales de mediciones y estereotipo de entidad.
- *Experimentación*, a partir de la definición del defecto se genera un conjunto de instancias con sus características y se aplica el algoritmo J48 de clasificación basado en árboles para generar un clasificador.
- *Identificación*, sobre un sistema software se buscan entidades (capas, paquetes, subsistemas) con defectos de diseño.
- *Validación*, se revisa la tabla inicial de entidades con defectos buscando falsos positivos y falsos negativos. Con los resultados de esta revisión se permita realimentar de nuevo la definición de defectos generando un nuevo clasificador. Se utiliza *F-measure* como indicador de condición de parada del proceso. Un valor mínimo aceptable es 0.66, que se obtiene asignando a la precisión, la moda de probabilidad industrial en la detección de defectos del 50 % propuesta en [6] y un 100 % a la recuperación.

3. Descripción de la herramienta

Architecture Defects Finder (ADF) es un plugin para la herramienta Eclipse, que automatiza las tareas de identificación y validación del proceso descrito en la Sección 2. Permite al usuario realizar las siguientes operaciones:

- Calcular y analizar métricas de código sobre las entidades: paquetes, capas y componentes, que componen la aplicación a inspeccionar.
- Clasificar las entidades en estereotipos UML [7]: *entity, user interface, device interface, sytem interface, utility, test, control.*
- Detectar los posibles defectos de arquitectura, a nivel de paquete, capa o componente [2] , aplicando técnicas de minería de datos sobre métricas de código y la información del estereotipo UML de la entidad.

Los defectos arquitectónicos detectados son los relacionados con las entidades del sistema presentados en [2]. En concreto, *entidad no usada*, *ciclos entre entidades*, *entidad demasiado pequeña*, *entidad demasiado grande*.

La integración con Eclipse se ha basado en: obtener la información del sistema a evaluar desde la definición del proyecto, la creación de nuevos menús de acceso a la nueva funcionalidad, definición de una nueva perspectiva con tres vistas y la integración de la ayuda en el entorno.

El plugin utiliza los módulos de cálculo de métricas de las herramientas software *DependencyFinder* [8] y *JDepend* [9]. Estos módulos se han adaptado incorporando los conceptos de capa y componente, y proporcionando filtros de las entidades a analizar. Además integra cierta funcionalidad de la herramienta de minería de datos Weka [10], en concreto, la implementación del algoritmo de clasificación J48, la representación gráfica del árbol de clasificación que se genera y el cálculo de la medida *F-measure* para controlar el proceso de detección.

4. Conclusiones

En el trabajo se ha presentado un plugin de Eclipse que semi-automatiza un proceso dinámico de gestión de defectos, del cuál se ha descrito brevemente sus tareas junto con una condición de parada. El plugin ADF ha semi-automatizado un contexto concreto de detección de defectos arquitectónicos basado en métricas y la información del estereotipo UML de la entidad. Se gestiona la subjetividad del auditor a partir de técnicas de clasificación de minería de datos. Desde este trabajo se aboga por partir de un clasificador genérico, y responsabilizar a la propia organización a ajustar el clasificador en su propio contexto de desarrollo: experiencia del personal, tipo de software a evaluar, tipo de lenguaje de programación, códigos autogenerados, etc. El resultado consistirá en añadir instancias con y sin defectos y generar un nuevo clasificador.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el *Ministerio de Ciencia e Innovación*, en el proyecto TIN2008-05675.

Referencias

1. James Newkirk and Robert C. Martin. *La programación extrema en la práctica*. 2002.
2. Stefan Roock and Martin Lippert. *Refactoring in Large Software Projects*. Wiley, 2006.
3. B. Walter and B. Pietrzak. Multi-criteria detection of bad smells in code with uta method. In H. Baumeister, M. Marchesi, and M. Holcombe, editors, *Lecture Notes in Computer Science*, volume 3556 of *6th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2005*, pages 154–161, Sheffield, 2005.
4. J. Garzás and M. Piattini. An ontology for microarchitectural design knowledge. *Software, IEEE*, 22(2):28–33, 2005. 0740-7459.
5. Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007. Journal.
6. Michael Fagan. Design and code inspections to reduce errors in program development. In *Software pioneers: contributions to software engineering*, pages 575–607. Springer-Verlag New York, Inc., 2002. 944367.
7. Ivar Jacobson, Grady Booch, and James Rumbaugh. *El Proceso Unificado de Desarrollo del Software*. Addison Wesley, 2000.
8. Jean Tessier. Dependency finder metric tool.
9. Inc. Clarkware Consulting. Jdepend metric tool., 1999.
10. University of Waikato. Weka, 1999 - 2007.