Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 4, Número 2 (especial X JICS), septiembre, 2008

Web de la editorial: www.ati.es/reicis
E-mail: editor-reicis@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos de Informática
www.ati.es

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Editorial

REICIS

El grupo de Calidad del Software de ATI ha consolidado su posición como principal promotor de la disciplina de ingeniería y calidad del software con la décima edición de las Jornadas sobre Innovación y Calidad del Software (las tradicionales JICS). Estas X JICS pretenden además potenciar la presencia iberoamericana en este foro de promoción de la cultura de la calidad del software y de la innovación en el desarrollo de sistemas y aplicaciones por lo que constituyen la promoción de una I Conferencia Iberoamericana de Calidad del Software (CICS). Por otra parte, las X JICS incorporan la presencia de la ponencia de un destacado experto europeo en la disciplina de ingeniería de software como es Darren Dalcher, Director del UK National Centre for Project Management en la Middlesex University y editor de la revista Software Process Improvement and Practice.

Por otra parte, queremos resaltar la línea de calidad de los trabajos, eminentemente prácticos pero rigurosos, aceptados entre los remitidos en la convocatoria de contribuciones: las ponencias aceptadas (con una tasa de rechazo del 40%) han sido sometidos a un completo proceso de revisión por el comité de programa así como a una cuidadosa labor de revisión de estilo, de terminología y de ortotipografía para garantizar el mejor resultado para nuestros lectores. Por supuesto, no cabe olvidar el apoyo de los patrocinadores (Telelogic, Steria, Deiser, GESEIN y SOGETI) no sólo aportando recursos sino también interesantes presentaciones de experiencias prácticas de sus expertos. Los debates promovidos en las mesas redondas así como la promoción de las actividades de comunicación y *networking* entre los participantes, tanto a nivel presencial como a través de la lista de distribución, los medios electrónicos y la nueva oferta formativa con plataforma *e-learning*. En definitiva, el evento más completo con toda la información disponible en la página del grupo de Calidad del Software (www.ati.es/gtcalidadsoft) acorde a la trayectoria pionera en España que, desde 1997, está proporcionando, a través de la Asociación de Técnicos de Informática, el apoyo para la productividad y la calidad en los proyectos de software. Este perfil ha sido reconocido por el apoyo del Ministerio de Industria, Turismos y Comercio con su apoyo institucional dentro de la convocatoria de la orden ITC/390/2007. Por último, debemos resaltar la aportación de datos de gran importancia no sólo mediante los eventos organizados sino también a través de la realización de estudios específicos (por ejemplo, sobre las prácticas de pruebas, el diseño de casos y los factores que dificultan su implantación eficiente y eficaz en las organizaciones) que permiten un mejor conocimiento de la práctica real de esta disciplina en España.

Luis Fernández Sanz
Juan J. Cuadrado-Gallego
Editores

Presentación

REICIS

En este número especial de septiembre de 2008 de REICIS, por primera vez en la historia de nuestra revista, esta publicación se convierte en el vehículo de difusión del evento decano en España en el ámbito de la ingeniería y la calidad del software: las Jornadas de Innovación y Calidad del Software (JICS) que alcanzan así su décima edición desde su inicio en 1998. En esta ocasión, el Grupo de Calidad del Software de ATI (www.ati.es/gtcalidadsoft) no sólo ha querido cumplir con esta decena de ediciones sino que ha apostado por una apertura a nuevos retos como la presencia de eminentes ponentes invitados de gran presencia internacional y la potenciación de los vínculos iberoamericanos para convertir a este evento en la referencia sobre calidad del software en la amplia comunidad latina. Los trabajos aceptados han sido sometidos a un completo proceso de revisión por el comité de programa así como a una cuidadosa labor de revisión de estilo, terminología y ortotipografía para garantizar la mejor calidad para nuestros lectores. Este número especial constituye en definitiva la publicación de las actas de las X JICS y, por ello, cuenta con un tamaño mayor del habitual. Esperamos repetir este número especial el próximo año con la undécima edición de las Jornadas de Innovación y Calidad del Software. Agradecemos la labor del comité de programa coordinado por la Dr. M.Idoia Alarcón (Universidad Autónoma de Madrid) y compuesto por la siguiente lista de expertos:

- Antonia Mas (Universitat de les Illes Balears)
- Luis de Salvador (AGPD)
- Ricardo Vargas (Universidad del Valle de Méjico)
- Javier Tuya (Universidad de Oviedo)
- Antonio de Amescua (Universidad Carlos III de Madrid)
- María Moreno (Universidad de Salamanca)
- José Antonio Calvo-Manzano (Universidad Politécnica de Madrid)
- José Antonio Gutiérrez de Mesa (Universidad de Alcalá)
- Isabel Ramos (Universidad de Sevilla)
- Esperança Amengual (Universitat de les Illes Balears)
- José Ramón Hilera (Universidad de Alcalá)
- Mercedes Ruiz (Universidad de Cádiz)
- María Teresa Villalba (Universidad Europea de Madrid)
- Adolfo Vázquez (INSA)
- María José Escalona (Universidad de Sevilla)
- Ana Araújo (Ministerio de Medio Ambiente)
- Antonio Rodríguez (Telelogic)
- Gurutze Miguel (TQS)
- Beatriz Pérez (Centro de Ensayos de Software, Uruguay)
- José Javier Martínez (Universidad de Alcalá)
- José Díaz (SSQTB)

Luis Fernández Sanz

Revisiones de código en el contexto del aseguramiento de calidad. Un caso práctico

M. J. Escalona, M. Pérez-Pérez, O. González-Barroso,
J. Ponce, J. M. Correa, A. I. Merino
Universidad de Sevilla
escalona@lsi.us.es

Abstract

The verification and validation of source code is one of the most critic tasks in quality assurance. Nowadays, development teams and technical offices are demanding valid tools and quality environments to perform verification and validation tasks. Nowadays, there are several powerful tools which allow to automatism these tasks. This paper introduces a practical case using PMD tool in real software projects.

Key words: quality of software, verification and validation of source code.

Resumen

La verificación y validación del código es una de las actividades más críticas en los desarrollos de software que sirven para verificar la calidad de los productos que se generan. En la actualidad, los equipos de desarrollo y las oficinas de calidad demandan herramientas y entornos de referencia adecuados para esta verificación y validación. Hoy en día existen entornos de herramientas potentes y que permiten automatizar este tipo de tareas. En este trabajo se presenta una muy concreta, PMD, y se ofrece una visión de cómo se ha adaptado a diferentes proyectos reales.

Palabras clave: calidad del software, verificación y validación de código.

1. Introducción

Cada día, la complejidad de los proyectos de software aumenta y se incrementa la necesidad de controles e inspectores que permitan medir la calidad de los trabajos que se van realizando sin llegar hasta el final del proceso [1].

Detectar los errores en las etapas más tempranas posibles garantiza un menor coste de resolución; por ello, hoy en día las empresas están potenciando el uso de herramientas y entornos que permitan hacer una medida automática de estos indicadores.

El grupo de autores de este artículo orienta su línea de investigación hacia estos aspectos. Sin embargo, este trabajo se centra una propuesta para analizar la calidad de las entregas en la fase de construcción.

La herramienta PMD incluye una serie de reglas y permite validar el código Java de un proyecto. La herramienta, de software libre y que se integra fácilmente en entornos como eclipse, permite aplicar una serie de reglas para medir y validar la calidad del código Java. Estas características, además de la existencia de una comparativa publicada en la ISSRE 2004 [2], hicieron que se eligiera PMD frente a otras propuestas. Sin embargo, la configuración por defecto de PMD no siempre resulta operativa en todo tipo de proyectos.

En el epígrafe 2 de este trabajo se presenta la tecnología PMD; luego se continúa analizando cómo puede adaptarse a proyectos reales y se explica la experiencia que se ha obtenido en proyectos reales. Con este fin, en el apartado 3 se analiza la adaptación que se ha realizado y en el apartado 4 se presenta la visión práctica y la experiencia obtenida. Finalmente, se termina con un análisis de los trabajos relacionados y con las conclusiones y trabajos futuros.

2. La tecnología PMD

El objetivo principal de una herramienta como PMD es ofrecer un entorno automático que permite la obtención de métricas objetivas para medir la calidad de un código fuente. En este contexto, se entiende por métricas de código a un conjunto de medidas de software que proporcionan a los programadores una mejor visión del código que están desarrollando.

Normalmente, la calidad y la velocidad de desarrollo no van unidas y la perfección tiene, en la mayoría de los casos, un coste demasiado alto. De esta forma, se tiende a construir software de acuerdo con unos tiempos y unos costes que en muchos casos son tan irreales que hacen que la calidad de los productos obtenidos se vea mermada.

Herramientas como PMD tratan de definir entornos normalizados que ayuden a mejorar la integración y herencia de código ajeno, así como a facilitar el establecimiento de un entorno común de desarrollo, favoreciendo la comprensión del código y, sobre todo, la minimización del impacto de los errores de integración.

Con este objetivo, el primer problema que hay que solventar es cuantificar la calidad de un código fuente Java, lo cual puede llegar a ser algo bastante subjetivo. Por esta razón, es necesario buscar mecanismos que aseguren una homogeneidad de criterios y, por tanto, de los resultados de la evaluación, independientemente de si el código es evaluado

por el propio grupo de desarrollo, por un grupo externo de calidad o por cualquier otro miembro del proyecto.

Se hace necesario que la validación del código, o bien se realice de acuerdo con una lista de comprobación bien definida (prueba, error y valoración objetiva), o que se haga de un modo automático.

A tal fin nacen productos de software como PMD, que automatiza esta validación y, por tanto, nos permite garantizar esta homogeneidad.

PMD es totalmente integrable en entornos como NetBeans o Eclipse y, tal como se analiza en el apartado de trabajos relacionados, no excluye a otros. Entre sus características fundamentales se encuentra su capacidad de analizar código desde el punto de vista de la ejecución y su capacidad de detectar:

- Posibles errores: try/catch/finally/switch incompletos.
- Código que no se usa: variables locales, parámetros y métodos privados.
- Código no óptimo: mejor uso de String/StringBuffer.
- Expresiones redundantes: if innecesarios, for loops que podrían ser while loops, etc.
- Código duplicado: código copiado y pegado significa bugs copiados y pegados.

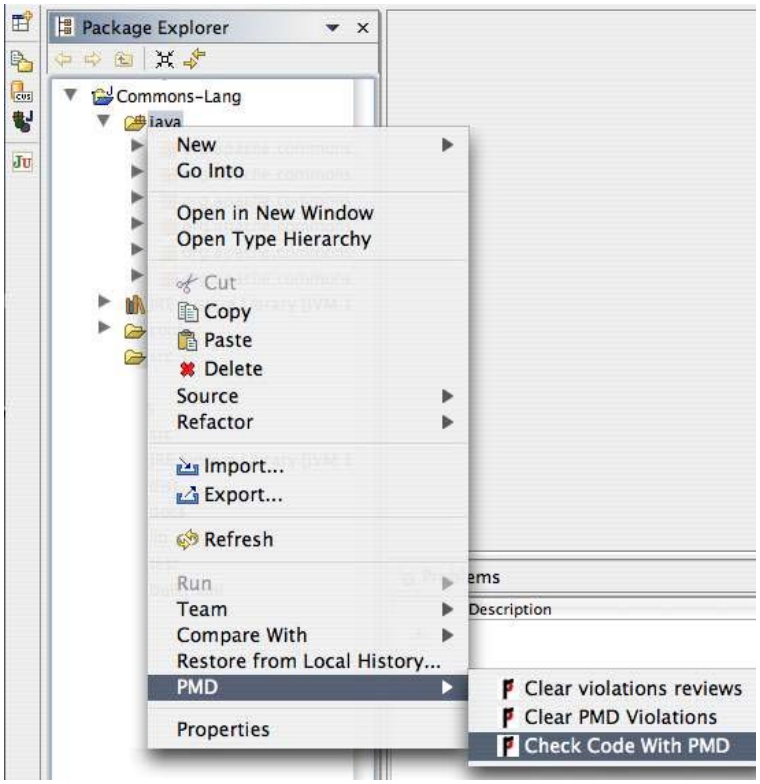

Figura 1. Interfaz de PMD.

Además, por ser un producto de software libre, está integrado con los principales IDEs[1]: JDeveloper, Eclipse, jEdit, JBuilder, BlueJ, CodeGuide, NetBeans / Sun Java Studio Enterprise / Creador, IntelliJ IDEA, Textpad, Maven, Ant, gel, JCreator y Emacs, de los cuales existen abundantes manuales para cada IDE en las web habituales de consultas de desarrolladores.

También cabe resaltar el soporte ofrecido en la web oficial (http://sourceforge.net/project/showfiles.php?group_id=56262), desde donde se puede descargar la última versión liberada, un resumen de todas las reglas (Current Rulesets), documentación específica, etc.

En la Figura 1 se muestra, a modo ilustrativo, la sencillez con la que PMD se integra dentro del propio eclipse.

3. La adaptación de PMD

La herramienta PMD presenta una serie de reglas definidas. Sin embargo, para ofrecer un mayor rendimiento en los proyectos es más adecuado redefinir nuevas reglas y hacer un estudio de aquellas que resultan más convenientes para el entorno en el que se trabaje.

Uno de los objetivos que nos marcamos como grupo de investigación es ofrecer entornos para evaluar el código fuente para la fase de construcción e implantación de proyectos. Decidimos hacer uso de PMD para detectar posibles errores y código ineficiente. El uso de PMD requiere una serie de adaptaciones al trabajo.

El proceso para definir el entorno más adecuado a los proyectos en los que colaboramos siguió una secuencia de pasos. En un primer lugar, había que seleccionar la plataforma, PMD, que puede ejecutarse sobre la línea de comandos, pero también ofrece la posibilidad de integrarse en plataformas de desarrollo. En nuestro caso, la elección fue Eclipse, puesto que era el entorno de desarrollo que más se utilizaba en aquellos entornos en los que trabajábamos. De esta forma, se propuso el plugin que facilita PMD mediante el cual se integra dentro del entorno Eclipse (http://pmd.sourceforge.net/integrations.html#eclipse).

[1] Para todas las herramientas que se mencionan en este artículo, consúltese su URL en el Apéndice A.

El siguiente paso era definir las reglas que, de manera efectiva, había que utilizar para la implantación. PMD ofrece un conjunto de reglas predefinidas (Rulesets) en el que se incluye un gran número de reglas cuya criticidad es evaluada desde el 1, muy críticas, hasta el 5, leves.

Para decidir y adecuar estas reglas, se evaluó sobre un proyecto real el alcance de los errores detectados con los Rulesets originales de PMD. Tras un estudio de esos errores, se hizo un estudio para adaptar esos Rulesets a nuestras necesidades, adoptando nuevas reglas de verificación de código y eliminando aquellas que, de manera empírica, parecía que detectaban errores no relevantes al entorno de trabajo.

Las reglas en PMD se pueden definir con código Java o con XPath (http://www.w3.org/TR/xpath). Las reglas incluidas en nuestra adaptación han sido definidas de ambas maneras, puesto que en algunos casos sencillos era más fácil definir la regla con Java, mientras que para reglas con una alta complejidad, la mejor opción era utilizar XPath.

Para añadir nuestras propias reglas, hemos usado la herramienta de diseño de reglas que viene incluida con PMD. El proceso para crear una regla se resume en escribir un trozo de código que incumple dicha regla, generar con la herramienta el árbol de sintaxis abstracta y escribir la expresión XPath que encuentre solo los nodos del árbol que incumplen la regla. La misma herramienta de diseño de reglas de PMD genera el código XML de la regla para poder incluirla dentro del Ruleset.

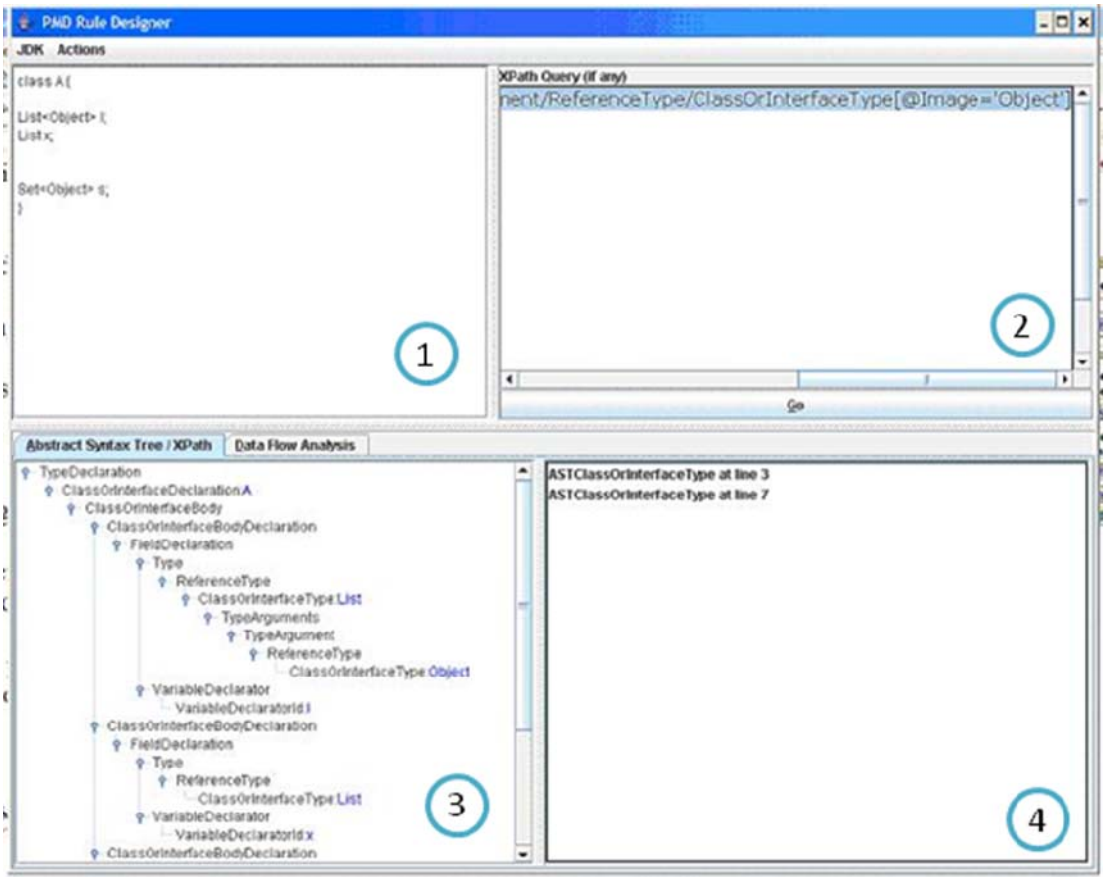

Figura 2. Diseño de una nueva regla para PMD.

En la Figura 2 se muestra una captura de la herramienta de diseño de reglas con una nueva regla que verificará que no se utilicen genéricos de tipo Object (por ejemplo, List<Object>). En el cuadro 1 de la Figura 2 se ha escrito un fragmento de código Java que tiene dos incumplimientos de la regla (las líneas “List<Object> l;” y “Set<Object> s;”). En el cuadro 3 se puede ver el árbol de sintaxis abstracta generado a partir del código de ejemplo del cuadro 1. En el cuadro 2, se muestra parcialmente una expresión XPath que permite encontrar aquellos nodos del árbol que declaran genéricos de tipo Object. Esa expresión se muestra a continuación:

```
//ReferenceType/ClassOrInterfaceType/TypeArguments/TypeArgument/ReferenceTy
pe/ClassOrInterfaceType[@Image='Object']
```

Finalmente, en el cuadro 4 se puede comprobar cómo la expresión detecta aquellas líneas que declaran genéricos de tipo Object.

Tras estudiar los diferentes niveles de error definidos en el Ruleset original, decidimos que, para los proyectos que evaluamos, los errores de nivel 1 y 2 eran inadmisibles para la aceptación del código. Los errores de nivel 3 serían revisados manualmente, ya que algunos podrían ser aceptables o no, y los restantes errores de nivel 4 y 5 serían aceptados.

Otro de los pasos que hay que seguir en la adaptación es la parametrización de la herramienta CPD (Copy&Paste Detector) (http://pmd.sourceforge.net/cpd.html). Esta herramienta busca en el código trozos duplicados. La parametrización de la herramienta se ha modificado a 50 caracteres (en su versión por defecto está en 25 caracteres), ya que es más que suficiente para detectar bloques de 5 o 6 líneas si estas tienen muchos caracteres, o unas 10 líneas si tienen pocos.

Y, por último, había que adecuar la salida de la evaluación realizada por PMD. A la hora de presentar un informe de la evaluación de la calidad del código, en un principio usamos la opción que ofrece el plugin de Eclipse de exportar el reporte de incidencias a CSV y posteriormente lo importamos desde una base de datos Access. En esa base de datos procedíamos a ejecutar consultas y a generar informes con los diferentes errores, según paquete de código y según nivel de criticidad.

Posteriormente, tras entrar a fondo en la propia herramienta PMD mediante línea de comando, decidimos hacer uso del paquete Renders (http://net.sourceforge.pmd.renderers), haciendo una ampliación de esta funcionalidad mediante la creación de una clase que genera un documento RTF con la librería iText (http://www.lowagie.com/iText/), consistente en una tabla con todas las incidencias encontradas por PMD, que muestra el archivo en el que está el error, la línea, el mensaje y el nivel de criticidad del error.

4. Aceptación de PMD

Tras adaptar PMD a nuestra forma de trabajo para conseguir un grado de optimización de código Java basado en el estándar más actual, había que hacerlo llegar a las empresas. De esta forma, colaborando con diferentes entidades, hemos implantando el uso de PMD en la evaluación de 22 proyectos de software, de diferente tamaño y complejidad, en la

Consejería de Cultura de la Junta de Andalucía y en un proyecto de gran envergadura de la Empresa Municipal de Aguas de Sevilla, Emasesa.

Ante el desconocimiento de la herramienta, en cierto modo encontramos cierta resistencia ante los resultados obtenidos por PMD, puesto que suponía un atraso en la entrega de sus proyectos, debido en gran parte al tiempo que empleaban los equipos de desarrollo de estas entidades para corregir los errores detectados por PMD.

Se impartieron cursos de formación en los que se trataban temas como la compatibilidad de PMD para ser instalado en muchas de las plataformas de trabajo (en nuestro caso, Eclipse), así como su uso a la vez que se desarrollaba el código, lo cual evitaba, en gran medida, muchos de los errores con los que llegaba el proyecto.

La finalidad de estos cursos era convencer a los diferentes jefes de proyectos de la implantación de esta herramienta a sus equipos de desarrollo para que, al tiempo que se iba desarrollando la herramienta, esta se pudiera ir ejecutando y se fueran corrigiendo los errores detectados de una forma limpia. De este modo, aquel código que provenía de entornos de desarrollo que venía aplicando la herramienta, era entregado totalmente limpio de errores inaceptables (errores y errores altos o graves), o con un número bajo de errores de otros niveles que se consideran admisibles desde el punto de vista de la calidad.

La transición desde el uso de esta herramienta ha sido buena, puesto que al inicio la gran mayoría de los proyectos que se entregaban tenían una calidad muy baja, a causa de numerosos errores graves (que no eran permitidos dentro de nuestro baremo) y de la no utilización de la herramienta.

Actualmente la forma de trabajo es buena, puesto que los diferentes proyectos que aún carecen de esta herramienta como un valor añadido a su proyecto desde el punto de la vista de la calidad del código, se van pasando por estas verificaciones cada poco tiempo. De este modo se van solucionando los problemas que van surgiendo, en lugar de realizar una única entrega que no haya pasado progresivamente bajo los estudios de calidad realizados por la herramienta PMD. Por el contrario, aquellas empresas que cuentan con esta herramienta como base para aplicar diferentes criterios de calidad a su código, llegan con mínimos errores y se devuelven muy pocos proyectos con incidencias serias que, como ya comentamos antes, en los niveles de gravedad de PMD serían errores graves y muy graves.

5. Trabajos relacionados

Como ya se ha mencionado con anterioridad, existen distintas herramientas que permiten estudiar métricas de código Java y revisar conjuntos de reglas. Al consultar distintos buscadores de Internet, tanto genéricos (como Google) como específicos en el ámbito de la programación (como Sourceforge), se han encontrado 18 herramientas, 9 de ellas de libre descarga y uso, y 9 comerciales.

Herramientas gratuitas	Enlaces
Checkstyle	http://checkstyle.sourceforge.net/
DoctorJ	http://www.incava.org/projects/java/doctorj/
ESC/Java	http://kind.ucd.ie/products/opensource/ESCJava2/
FindBugs	http://findbugs.sourceforge.net/
Hammurapi	http://www.hammurapi.biz/
JCSC	http://jcsc.sourceforge.net/
Jikes	http://jikes.sourceforge.net/
JLint	http://www.artho.com/jlint/
JWiz	http://csdl.ics.hawaii.edu/Research/JWiz/JWiz.html

Tabla 1. Herramientas gratuitas.

Herramientas de pago	Enlaces
AppPerfect	http://www.appperfect.com/
Assent	http://www.pune.tcs.co.in/Assent_registration.htm
Aubjex	http://mindprod.com/jgloss/aubjex.html
AzoJavaChecker	http://www.andiz.de/azosystems/en/index.html
CodePro Studio	http://www.instantiations.com/codepro/index.html
Flaw Detector	http://www.excelsior-usa.com/fd.html
JStyle	http://www.mmsindia.com/jstyle.html
JTest	http://www.parasoft.com/jsp/products/home.jsp?product=Jtest
Lint4J	http://www.jutils.com/

Tabla 2. Herramientas de pago.

Dado que aquí es imposible describirlas todas en detalle, a continuación se citan algunas de las herramientas gratuitas más relevantes y se comparan con la herramienta PMD.

5.1. Checkstyle

Como ya se ha indicado, esta herramienta verifica el código fuente para asegurar que este sigue un estándar de codificación. Sin embargo, el número de reglas que incorpora es menor que PMD, muchas de las reglas ya están incluidas en PMD y las que no lo están son

muy sencillas de añadir, como se ha visto en secciones anteriores. Sin embargo, la herramienta Checkstyle es más difícil de extender, ya que solo ofrece la opción de extenderse escribiendo código Java y aplicando la implementación del patrón visitante (VP).

5.2. Hammurapi

La documentación de esta herramienta es inferior a la documentación de la herramienta PMD. Además, también es necesario escribir nuevo código Java para extender el conjunto de reglas.

5.3. JLint

A diferencia de PMD y de las herramientas anteriores, la herramienta JLint está desarrollada en C++. Al igual que las herramientas anteriores, es más difícil de extender e incluye menos reglas que PMD. Experimentos publicados con anterioridad también muestran que esta herramienta tiene un porcentaje de falsos errores más alto que PMD. Sin embargo, JLint sí es capaz de verificar el tamaño de las tablas y detectar tamaños de tablas menores de cero, lo cual PMD no es capaz de hacer.

5.4. JCSC

Esta herramienta tiene más opciones de análisis que PMD, como el control de la documentación, la nomenclatura y el orden de la definición de las clases. JSCS tiene menos plugins que PMD, si bien tiene un plugin completo en ejecución de código que generará un documento con los errores dado por el código.

5.5. Assent

Assent es una herramienta comercial alternativa a PMD. No obstante, existe un plugin de Assent para Eclipse que nos permitiría usar las reglas de PMD complementadas con Assent y viceversa. Tiene una amplia gama de control en documentación de clases, métodos y variables.

6. Conclusiones

Este artículo se centra en una fase muy concreta del ciclo de vida, la construcción. En él se presenta el uso de la herramienta PMD como solución para evaluar de una manera objetiva la calidad de los productos de software entregados.

El artículo ha comenzado haciendo una breve introducción de PMD, enfocada a sus objetivos y características principales. Tras eso, se ha presentado el proceso seguido por el grupo de autores para realizar su adaptación y adecuación al entorno real de los proyectos con los que comúnmente trabajan.

Después de la presentación de la adaptación, se explicado cómo se ha implantado en los proyectos y los costes en formación que esto ha supuesto. Finalmente, se han presentado los trabajos relacionados.

Cabe decir que, tras las experiencia real, y una vez formado el equipo de desarrollo, PMD no es un herramienta que encarezca los proyectos. Si los equipos de programadores conocen las reglas de antemano, poco a poco van adquiriendo aptitudes que los orientan a escribir el código, incluyendo esos aspectos de calidad.

Por otro lado, nuestro trabajo de adaptación no ha terminado aún. Si bien el conjunto de reglas que se propone para los proyectos está ya muy depurado, con cada nueva aplicación empírica encontramos una nueva fuente de inspiración para la revisión. En la actualidad se aplica a 23 proyectos reales y a proyectos fin de carrera que desarrollamos en el seno del grupo de investigación. No obstante, está previsto que este número crezca gracias a las colaboraciones con empresas que mantenemos.

Por último, hemos de indicar que la aplicación de PMD es solo una parte del trabajo de calidad que proponemos desde el grupo. En él tenemos definida toda una metodología de desarrollo en la que existen inspectores y medidores de calidad en todas las fases y que también aplicamos sobre estos proyectos. Actualmente estamos en plena fase de certificación, bajo la norma ISO 9001, de todo el sistema de aseguramiento de la calidad que tenemos definido.

Agradecimientos

La elaboración de este artículo ha contado con el apoyo del subproyecto QSimTest del proyecto Cycit (TIN2007-67843-C06_03) y de la red de investigación RePRIS (TIN2007-30391-E) del Ministerio de Educación y Ciencia.

Referencias

[1] Armesto, A., Loniewski, G., Carlos, A. y Llorens, V., “Incorporando el análisis estático de código en un proceso de integración continua”, en: Vos, T. J. (ed.), *Proceedings de las*

IV Jornadas de Testeo de Software 2007. Valencia (España), 3-4 de mayo de 2007, pp. 65-78, 2007.

[2] Rutar, N., Almazan, C. B. y Foster, J.S., "Comparison of Bug Finding Tools for Java", en: *Proceedings of the 15th International Symposium on Software Reliability Engineering. Saint-Malo (France), 2-5 de noviembre de 2004*, pp. 245- 256, 2004.

Apéndice A

JDeveloper: http://www.oracle.com/technology/products/jdev

Eclipse: http://www.eclipse.org/

jEdit: http://www.jedit.org/

JBuilder: http://www.codegear.com/products/jbuilder

BlueJ: http://www.bluej.org/

CodeGuide: http://www.omnicore.com/

NetBeans: http://www.netbeans.org/

IDEA: http://www.jetbrains.com/idea/

Textpad: http://www.textpad.com

Maven: http://maven.apache.org/

Ant: http://ant.apache.org/

Gel: http://www.gexperts.com/gel

JCreator: http://www.jcreator.com/

Emacs: http://www.gnu.org/software/emacs/