



Universidad de Valladolid

E.T.S. de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención Tecnologías de la Información

Estudio de PMD, Integración y Extensibilidad

Autor:

Diego Herrero Paniagua

Tutora:

Yania Crespo González-Carvajal

*Out of the night that covers me,
Black as the pit from pole to pole,
I thank whatever gods may be
For my unconquerable soul.*

*In the fell clutch of circumstance
I have not winced nor cried aloud.
Under the bludgeonings of chance
My head is bloody, but unbowed.*

*Beyond this place of wrath and tears
Looms but the horror of the shade,
And yet the menace of the years
Finds and shall find me unafraid.*

*It matters not how strait the gate,
How charged with punishments the scroll,
I am the master of my fate:
I am the captain of my soul.*

William Ernest Henley

AGRADECIMIENTOS

A mis padres Rosamary y Jesús, que me han apoyado todos y cada uno de los largos días de este largo camino que empezó hace 19 años, sin ellos nunca hubiera llegado hasta aquí.

A mis amigos, que me han dado todo el apoyo que necesitaba en las horas más duras, sacándome una sonrisa cuando más las necesitaba, ellos forman parte de mi como yo formo parte de ellos.

A mis compañeros de trabajo, que me han asesorado y animado a seguir adelante.

A mi tutora Yania por confiar en mis capacidades.

A todos ellos y a muchos más,
gracias por todo.

RESUMEN

Este proyecto consiste en el estudio de la herramienta de análisis estático de código PMD, la cual es usada ampliamente a la hora de detectar malas prácticas y vulnerabilidades a la hora de desarrollar aplicaciones.

Esta herramienta admite como entradas varios tipos de lenguajes, principalmente Java, y el resultado de su análisis se puede obtener en distintos formatos para su posterior análisis o presentación.

Además de contar con un catálogo muy amplio de reglas, PMD permite la creación de nuevas reglas para así satisfacer las necesidades del usuario.

ABSTRACT

This project involves the study of static source code analyzer PMD, which is widely used for detecting malpractices and vulnerabilities when developing applications.

This tool accepts as input various types of languages , especially Java, and the result of analysis can be obtained in different formats for further analysis or presentation.

Besides having a very wide range of rules, PMD allows the creation of new rules to satisfy user needs.

Índice

CAPÍTULO I: INTRODUCCIÓN	15
Visión general	17
Objetivos	17
Aplicaciones similares.....	17
CAPÍTULO II: PLANIFICACIÓN INICIAL	21
Organización del Proyecto	23
Roles y Responsabilidades.....	23
Estimaciones de Tiempos	23
Plan de Proyecto.....	23
Calendario del Proyecto.....	24
Recursos del Proyecto.....	26
Recursos Humanos	26
Recursos Hardware	26
Gestión de Riesgos.....	27
Plan de Riesgos	27
Planificación de Fases de Desarrollo	29
Estimación de Costes.....	31
CAPÍTULO III: DESCRIPCIÓN	33
Descripción	35
Funcionalidades.....	36
CAPÍTULO IV: INTEGRACIÓN Y USO	41
Línea de Comandos	43
PMD.....	44
CPD.....	47
Integración con Eclipse	48
Integración con Jenkins	53
CAPÍTULO V: CREACIÓN DE NUEVAS REGLAS	57
Funcionamiento Interno de PMD	59
Uso de Reglas Personalizadas	64
Reglas XPath	66
Definición de objetivos.....	66
Desarrollo	66
Reglas Java	68
Definición de objetivos.....	68

Desarrollo.....	68
CAPÍTULO VI: CONCLUSIONES	71
Conclusiones.....	73
Trabajo futuro	73
CAPÍTULO VII: BIBLIOGRAFÍA	75
CAPÍTULO VIII: ANEXOS	79
Código fuente de la clase Java ALotOfFinalStaticVariables	81
Estructura de la librería custom.jar.....	82
Manual de Pruebas.....	82
Contenido del CD.....	83

Índice de Tablas

Tabla 1. Roles y Responsabilidades	23
Tabla 2. Fases del Proyecto	23
Tabla 3. Fases e Hitos	23
Tabla 4. Calendario del proyecto	24
Tabla 5. Recursos del Proyecto	26
Tabla 6. Características del Ordenador de Desarrollo 1	26
Tabla 7. Características del Ordenador de Desarrollo 2	26
Tabla 8. R001	27
Tabla 9. R002	27
Tabla 10. R003	28
Tabla 11. R004	28
Tabla 12. R005	28
Tabla 13. Fases de Creación de Regla XPath	29
Tabla 14. Fases de Creación de Regla Java	29
Tabla 15. Coste del hardware	31
Tabla 16. Coste del software.....	31
Tabla 17. Coste de los recursos humanos	31
Tabla 18. Coste total.....	31
Tabla 19. Correspondencia de código de un bucle <code>while</code> con su representación en AST	61
Tabla 20. Correspondencia de código de un bucle <code>while</code> con llaves con su representación en AST	61
Tabla 21. Representación AST de una estructura <code>switch</code>	66
Tabla 22. Representación AST de la definición de constantes en una clase.....	68

Índice de ilustraciones

Ilustración 1. Detalle del uso de Checkstyle en Eclipse.....	18
Ilustración 2. Ejemplo de Findbugs, mostrando la descripción del bug hallado	19
Ilustración 3. Diagrama de Gantt del proyecto	25
Ilustración 4. Diagrama de Gantt de la Creación de Regla XPath	29
Ilustración 5. Diagrama de Gantt de la Creación de Regla Java	30
Ilustración 6. Logo de PMD.....	35
Ilustración 7. Carpeta raíz de PMD	43
Ilustración 8. Contenido de la carpeta bin.....	44
Ilustración 9. Ejemplo de ejecución del comando <code>pmd.bat</code>	44
Ilustración 10. Ejemplo de redirección de salida.....	45
Ilustración 11. Ejemplo de las salidas <code>xml</code> y <code>html</code>	45
Ilustración 12. Salida <code>textcolor</code>	45
Ilustración 13. Salida <code>xml</code>	46
Ilustración 14. Salida <code>html</code>	46
Ilustración 15. Descripción de regla en el sitio web de PMD.....	46
Ilustración 16. Ejemplo de ejecución del comando <code>cpd.bat</code>	47
Ilustración 17. Eclipse Marketplace.....	48
Ilustración 18. Opción Check Code	49
Ilustración 19. Fallos señalados en el código	49
Ilustración 20. Listado de fallos	50
Ilustración 21. Estadísticas de fallos.....	50
Ilustración 22. Opción Generate Reports	51
Ilustración 23. Carpeta reports	51
Ilustración 24. Opciones de informes del plugin PMD en Eclipse	52
Ilustración 25. Pantalla principal de Jenkins	53
Ilustración 26. Presentación de los resultados de un determinado proyecto.....	53
Ilustración 27. Detalle de resultado de análisis PMD agrupado por paquete	54
Ilustración 28. Detalle de resultado de análisis PMD agrupado por categoría	55
Ilustración 29. Detalle de resultado de análisis PMD agrupado por regla	55
Ilustración 30. Descripción del fallo detectado.....	56
Ilustración 31. Utilidad de PMD para la generación de AST	59
Ilustración 32. Ejemplo de AST	60
Ilustración 33. Evaluación de reglas en la herramienta <code>bgastviewer.bat</code>	62
Ilustración 34. Contenido del CD.....	83

CAPÍTULO I: INTRODUCCIÓN

Visión general

Cuando se desarrolla código para una nueva aplicación o un nuevo proyecto, no se obtiene, o no se suele, obtener el resultado idóneo a la primera. Esto puede provocar que en nuestras aplicaciones queden fragmentos de código que no se usan, bugs, agujeros de seguridad, etc...

Y esto ocurre por el simple hecho de que, aunque seamos profesionales de la programación, somos humanos, y siempre puede quedar algo que corregir en el tintero.

PMD surge de esta situación, de la realidad de que existen fallos o fragmentos inútiles en el código, para ayudar a localizarlos y corregirlos.

A título personal, durante la asignatura Prácticas de Empresa me he servido de este analizador de código para corregir fallos y vulnerabilidades de grandes proyectos, como puede ser la aplicación de captura de la solicitud única de la Política Agraria Común (PAC).

Por tanto, el objetivo final de este proyecto es conocer a fondo la herramienta PMD, cómo funciona y cómo se puede utilizar para satisfacer las necesidades específicas de un usuario.

Objetivos

Los objetivos de este proyecto se pueden dividir en tres campos:

- Conocimiento de la herramienta PMD. Sus orígenes y motivaciones, para comprender realmente cuál es su verdadera utilidad.
- Integración con otras aplicaciones. Debe ser posible interactuar con diversas aplicaciones para poder analizar el código, ya que no todo el mundo lo desarrolla de la misma manera.
- Extensibilidad. Al ser un proyecto desarrollado con Licencia Pública General Reducida de GNU, PMD está en constante evolución con la adición de nuevas funcionalidades, así que es necesario saber cómo poder aportar para que siga creciendo.

Aplicaciones similares

Como suele ser habitual en algunos casos, no siempre existe una sola solución para un determinado problema. Así, a lo largo de los años han surgido otros proyectos con ideas u objetivos similares que pueden desempeñar la misma función que PMD.

Por otro lado, muchas de estas herramientas coexisten, y un correcto uso combinado de algunas de ellas permiten el desarrollo de código de calidad.

La primera herramienta que vamos a tratar es Checkstyle, que ayuda a seguir un estándar de codificación a la hora de programar en Java, que pese a existir, siempre puede haber quien no lo cumpla, ya que no todo el mundo programa de la misma forma. Gracias a esta herramienta, integrable en algunos de los entornos de desarrollo más utilizados, como Eclipse, se puede generar código fácilmente interpretable por cualquier programador. El inconveniente, como se puede deducir, es que esta herramienta solo se limita a la presentación del código.^{[5][6]}

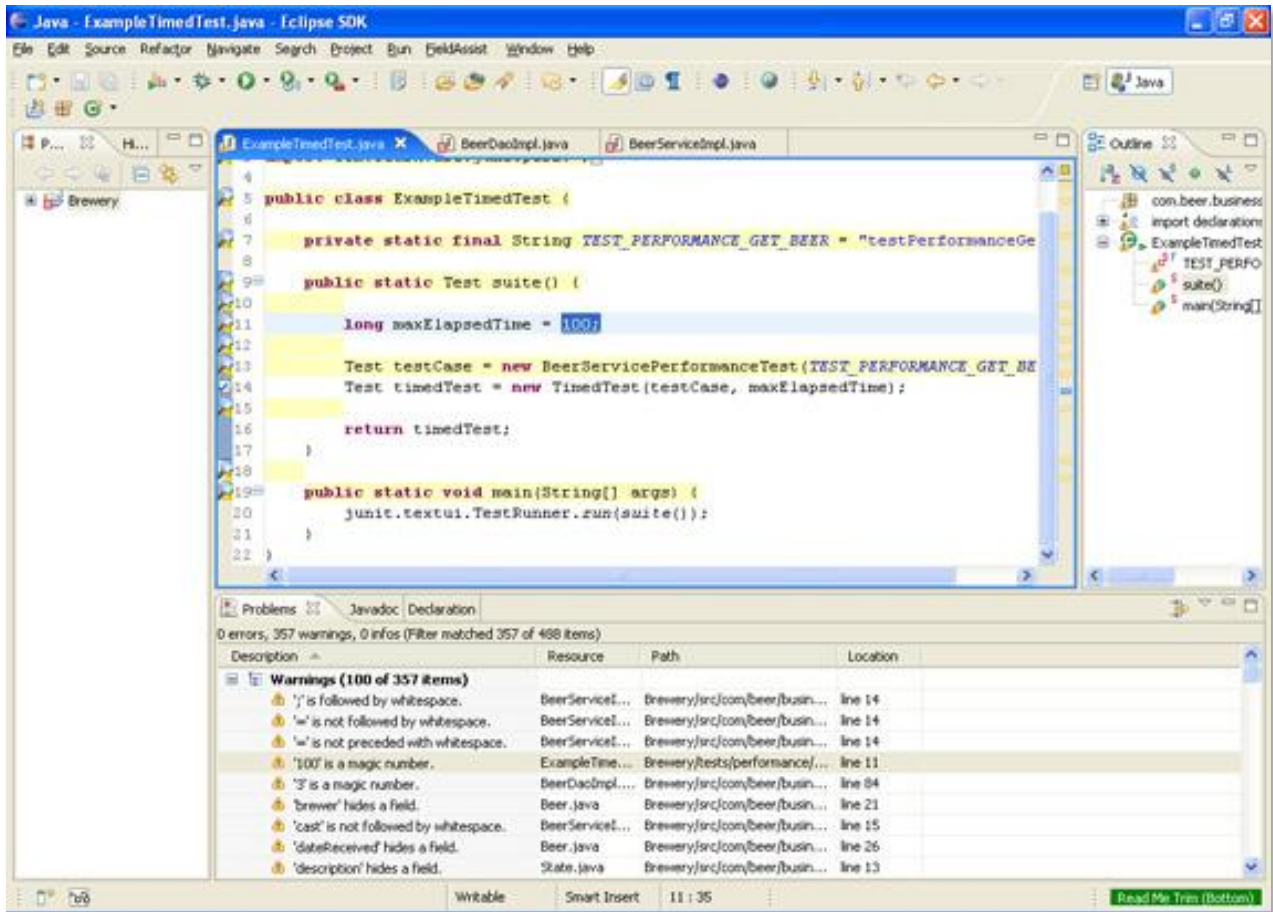


Ilustración 1. Detalle del uso de Checkstyle en Eclipse

La siguiente herramienta que vamos a presentar es Findbugs. Esta herramienta no vigila el aspecto del código, sino que lo analiza en busca de algunos bugs ya conocidos y muy comunes, como pueden ser posibles excepciones por puntero nulo o no almacenar el valor de retorno de un método en una variable. Su mayor inconveniente es que para ejecutar su análisis, es preciso la compilación previa del código a analizar.^{[7][8]}



Ilustración 2. Ejemplo de Findbugs, mostrando la descripción del bug hallado

En el caso de PMD, se puede decir que abarca parte del campo de acción de estas herramientas, añadiendo, además, nuevas funcionalidades que serán desarrolladas más adelante.^[9]

Como he dicho antes, el uso combinado de herramientas de este tipo nos permitirá desarrollar código limpio y de calidad. Algunos ejemplos de software que permiten dicha combinación son Eclipse o Jenkins, que serán tratados más adelante para el caso específico de PMD.

CAPÍTULO II: PLANIFICACIÓN INICIAL

Organización del Proyecto

El proyecto ha sido realizado por una persona, el alumno Diego Herrero Paniagua, quién se ha encargado de todos los roles durante el desarrollo del proyecto. Además, ha contado para su realización, con la ayuda de su tutora Doña Yania Crespo González-Carvajal.

Roles y Responsabilidades

Rol	Responsabilidades	Persona encargada
Gestor del Proyecto	Planificar, organizar, gestionar, dirigir y controlar el proyecto	Diego Herrero Paniagua
Planificador	Planificar y organizar las fases del proyecto	Diego Herrero Paniagua
Analista	Adquirir todo el conocimiento necesario para el desarrollo correcto y óptimo de nuevas reglas	Diego Herrero Paniagua
Diseñador	Desarrollador y persona encargada de comprobar el buen funcionamiento de las reglas	Diego Herrero Paniagua

Tabla 1. Roles y Responsabilidades

Estimaciones de Tiempos

A la hora de realizar la estimación del tiempo requerido para la realización del presente proyecto, se han tenido en cuenta la experiencia previa obtenida tanto en otros proyectos desarrollados durante la carrera, así como en las prácticas en empresa.

Plan de Proyecto

Al ser este un proyecto que en su mayor parte se trata del estudio de la técnica usada, se ha estimado oportuno dividir la planificación del proyecto en fases compuestas por cada una de las partes que lo componen. No obstante, a la hora de crear nuevas reglas, se ha tomado parte del Proceso Unificado para definir sus fases.

Fase	Fecha de inicio	Fecha de fin	Duración
Inicio	jue 10/12/15	mié 06/01/16	
Descripción	jue 07/01/16	mié 10/02/16	
Integración y Uso	jue 11/02/16	mié 16/03/16	
Creación de Nuevas Reglas	jue 17/03/16	mié 11/05/16	
Transición	jue 12/05/16	mié 08/06/16	

Tabla 2. Fases del Proyecto

Cada fase estará marcada por un hito, mostrados en la Tabla 3.

Fase	Hito
Inicio	Se aborda el proyecto en general, definiendo los objetivos generales, así como una planificación
Descripción	Desarrollo del estudio acerca de PMD, dando respuesta a cuestiones sobre su origen, uso y necesidad
Integración y uso	Análisis de las posibles entradas que puede tener el analizador de código PMD, así como el abanico de salidas para su posterior tratamiento
Creación de Nuevas Reglas	Adquisición de conocimiento en la elaboración de nuevas reglas para su posterior puesta en práctica
Transición	Composición y revisión del documento

Tabla 3. Fases e Hitos

Calendario del Proyecto

Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos
Comienzo del proyecto	0 días	jue 10/12/15	jue 10/12/15	Diego Herrero Paniagua
Inicio	20 días	jue 10/12/15	mié 06/01/16	
Definición de objetivos	5 días	jue 10/12/15	mié 16/12/15	Diego Herrero Paniagua
Planificación	15 días	jue 17/12/15	mié 06/01/16	Diego Herrero Paniagua
Descripción	25 días	jue 07/01/16	mié 10/02/16	
Descripción	10 días	jue 07/01/16	mié 20/01/16	Diego Herrero Paniagua
Funcionalidades	15 días	jue 21/01/16	mié 10/02/16	Diego Herrero Paniagua
Integración y uso	25 días	jue 11/02/16	mié 16/03/16	
Entradas	12,5 días	jue 11/02/16	lun 29/02/16	Diego Herrero Paniagua
Salidas	12,5 días	lun 29/02/16	mié 16/03/16	Diego Herrero Paniagua
Creación de nuevas reglas	40 días	jue 17/03/16	mié 11/05/16	
Regla XPath	20 días	jue 17/03/16	mié 13/04/16	Diego Herrero Paniagua
Java	20 días	jue 14/04/16	mié 11/05/16	Diego Herrero Paniagua
Transición	20 días	jue 12/05/16	mié 08/06/16	
Composición del documento	12 días	jue 12/05/16	vie 27/05/16	Diego Herrero Paniagua
Revisión del documento	8 días	lun 30/05/16	mié 08/06/16	Diego Herrero Paniagua
Fin del proyecto	0 días	jue 09/06/16	jue 09/06/16	Diego Herrero Paniagua

Tabla 4. Calendario del proyecto

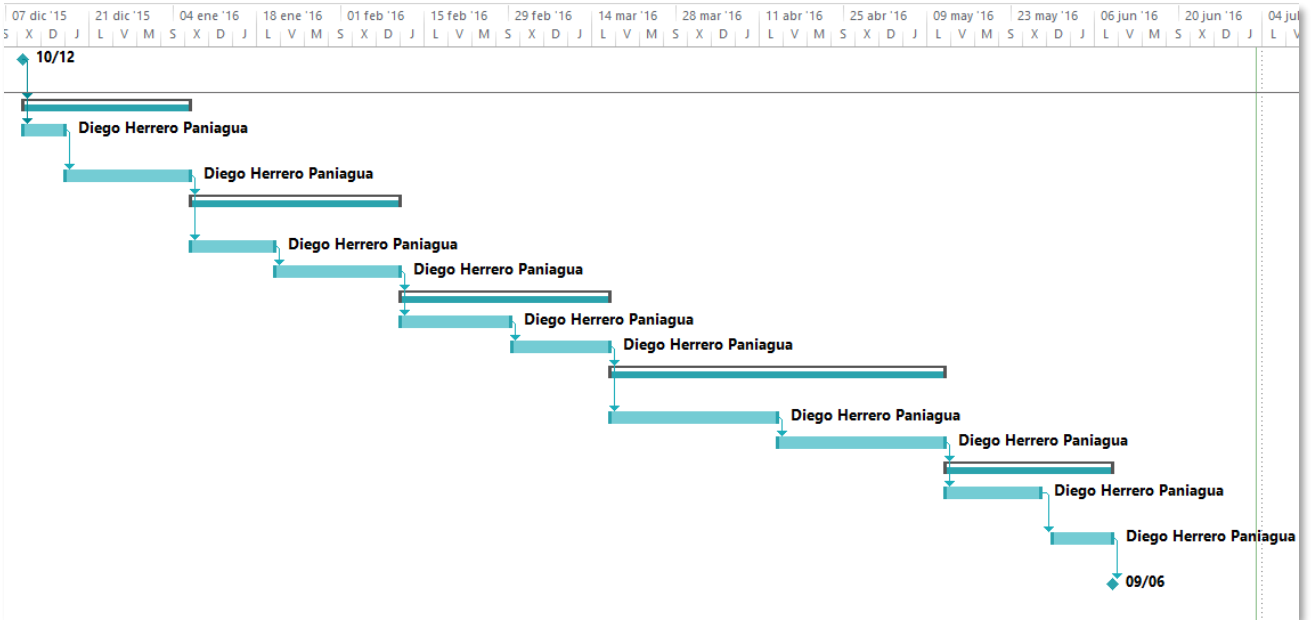


Ilustración 3. Diagrama de Gantt del proyecto

Recursos del Proyecto

Fase	Recursos Humanos	Recursos Hardware	Recursos Software
Inicio	Diego Herrero Paniagua	Ordenador de desarrollo 1	Microsoft Office 2016, Microsoft Project 2013
Descripción	Diego Herrero Paniagua	Ordenador de desarrollo 1, Ordenador de desarrollo 2	Microsoft Office 2016
Integración y uso	Diego Herrero Paniagua	Ordenador de desarrollo 2	Microsoft Office 2016
Creación de Nuevas Reglas	Diego Herrero Paniagua	Ordenador de desarrollo 2	Microsoft Office 2016, Eclipse Kepler, PMD
Transición	Diego Herrero Paniagua	Ordenador de desarrollo 2	Microsoft Office 2016

Tabla 5. Recursos del Proyecto

Recursos Humanos

Se cuenta únicamente con el trabajo de Diego Herrero Paniagua, encargado de la realización de todas y cada una de las fases.

Recursos Hardware

El único recurso hardware necesario es el ordenador de desarrollo del alumno.

Ordenador de desarrollo 1 – Acer Aspire 5750	
CPU	Intel Core i5-2410M
GPU	Intel HD Graphics 3000 (Integrada)
RAM	4 GB RAM DDR3
Resolución de pantalla	15.6 pulgadas con una resolución de 1.366 x 768 px
Disco duro	500 GB
Sistema Operativo	Windows 10 Pro

Tabla 6. Características del Ordenador de Desarrollo 1

Debido a la avería del primer ordenador de desarrollo, fue necesaria la adquisición de un segundo equipo.

Ordenador de desarrollo 2 – Asus X555LJ	
CPU	Intel Core i7-5500U
GPU	NVIDIA GeForce GT 920M
RAM	4 GB RAM DDR3
Resolución de pantalla	15.6 pulgadas con una resolución de 1.366 x 768 px
Disco duro	1 TB
Sistema Operativo	Windows 10 Pro

Tabla 7. Características del Ordenador de Desarrollo 2

Gestión de Riesgos

Para incrementar la calidad del proyecto y sus posibilidades de éxito, es necesario llevar a cabo un análisis de los riesgos que pueden surgir durante la realización del mismo, así como una posible solución. [10]

Plan de Riesgos

A continuación, se detallan algunos de los riesgos considerados para este proyecto.

R001 Falta de experiencia en la planificación de proyectos	
Descripción	El alumno tiene poca experiencia en la planificación de proyectos de esta envergadura, ya que es el primer proyecto de este tipo que realiza
Efecto	Retrasos en las fases y entregas
Contexto del riesgo	Fase de inicio
Probabilidad	Alta
Consecuencia	Alta
Estrategia correctora	Realizar una estimación inicial de los tiempos con amplios márgenes que puedan contener los posibles retrasos del proyecto, así como tomar ejemplo de otros proyectos para aproximarse a la realización de una mejor planificación
Plan de contingencia	En caso de retrasos en los plazos, se aumentará la cantidad de trabajo realizada cada día, así como establecer horas extras para cumplir con los plazos

Tabla 8. R001

R002 Circunstancias propias del alumno	
Descripción	El alumno puede encontrarse ante situaciones imprevistas, tales como viajes, enfermedades, horas extras en el desempeño profesional de su trabajo
Efecto	Retrasos en las fases y entregas
Contexto del riesgo	En cualquier fase
Probabilidad	Baja
Consecuencia	Alta
Estrategia correctora	Tener en cuenta en la planificación estas circunstancias, ampliando los plazos si es necesario
Plan de contingencia	En caso de retrasos en los plazos, se aumentará la cantidad de trabajo realizada cada día, así como realizar horas extras para cumplir con los plazos

Tabla 9. R002

R003 Falta de motivación	
Descripción	El alumno puede verse falto de motivación por el contenido del proyecto o por el desempeño simultáneo de su actividad laboral
Efecto	Retrasos en las fases y entregas, y en caso extremo, abandono del proyecto
Contexto del riesgo	En cualquier fase
Probabilidad	Baja
Consecuencia	Muy alta
Estrategia correctora	En casos en los que el alumno desempeñe una actividad laboral, efectuar descansos entre una y otra actividad para poder cambiar la mentalidad
Plan de contingencia	Aumentar el ritmo de trabajo, realizar el proyecto en momentos más relajados como fines de semana o días libres

Tabla 10. R003

R004 Disponibilidad de los recursos hardware del proyecto	
Descripción	El equipo de desarrollo puede sufrir problemas de funcionamiento o averías
Efecto	Retrasos en las fases y entregas
Contexto del riesgo	En cualquier fase
Probabilidad	Baja
Consecuencia	Muy alta
Estrategia correctora	Tener conocimiento de los dispositivos en uso y las posibles alternativas en caso de necesidad (préstamo de dispositivos de la ETSII, ayuda de amigos y familiares, compra de nuevos dispositivos)
Plan de contingencia	Obtener un nuevo dispositivo para realizar el proyecto

Tabla 11. R004

R005 Pérdida de datos	
Descripción	Por cualquier circunstancia, puede haber una pérdida del trabajo realizado, bien sea borrado accidental, avería u otras
Efecto	Retrasos en las fases y entregas
Contexto del riesgo	En cualquier fase
Probabilidad	Baja
Consecuencia	Muy alta
Estrategia correctora	Realizar copias de seguridad periódicamente, tanto en dispositivos físicos como en servicios online (OneDrive, Dropbox...)
Plan de contingencia	Recuperar la copia de seguridad almacenada

Tabla 12. R005

Planificación de Fases de Desarrollo

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Regla XPath	20 días	jue 17/03/16	mié 13/04/16		
Definición de objetivos	1 día	jue 17/03/16	jue 17/03/16		Diego Herrero Paniagua
Adquisición de conocimiento	7 días	vie 18/03/16	lun 28/03/16	2	Diego Herrero Paniagua
Desarrollo	7 días	mar 29/03/16	mié 06/04/16	3	Diego Herrero Paniagua
Pruebas	5 días	jue 07/04/16	mié 13/04/16	4	Diego Herrero Paniagua

Tabla 13. Fases de Creación de Regla XPath

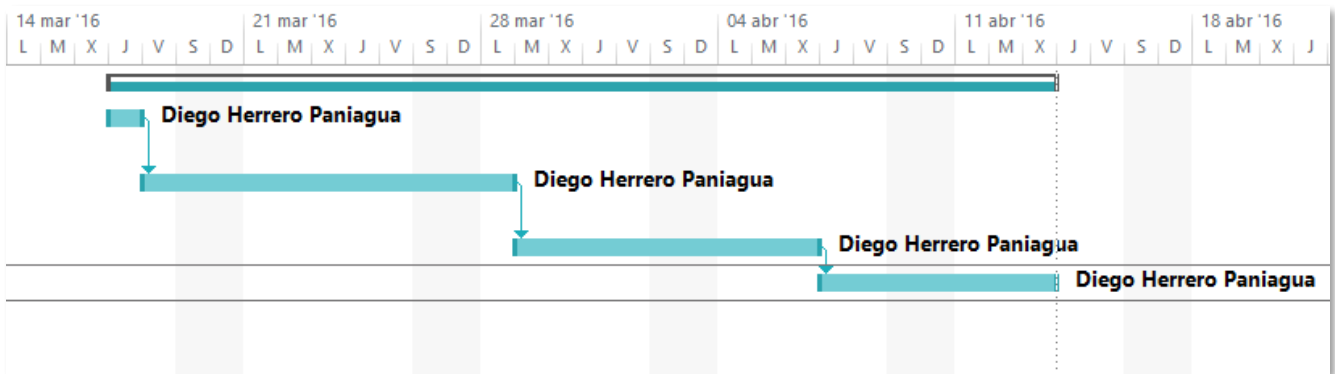


Ilustración 4. Diagrama de Gantt de la Creación de Regla XPath

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Regla Java	20 días	jue 17/03/16	mié 13/04/16		
Definición de objetivos	1 día	jue 17/03/16	jue 17/03/16		Diego Herrero Paniagua
Adquisición de conocimiento	7 días	vie 18/03/16	lun 28/03/16	2	Diego Herrero Paniagua
Desarrollo	7 días	mar 29/03/16	mié 06/04/16	3	Diego Herrero Paniagua
Pruebas	5 días	jue 07/04/16	mié 13/04/16	4	Diego Herrero Paniagua

Tabla 14. Fases de Creación de Regla Java

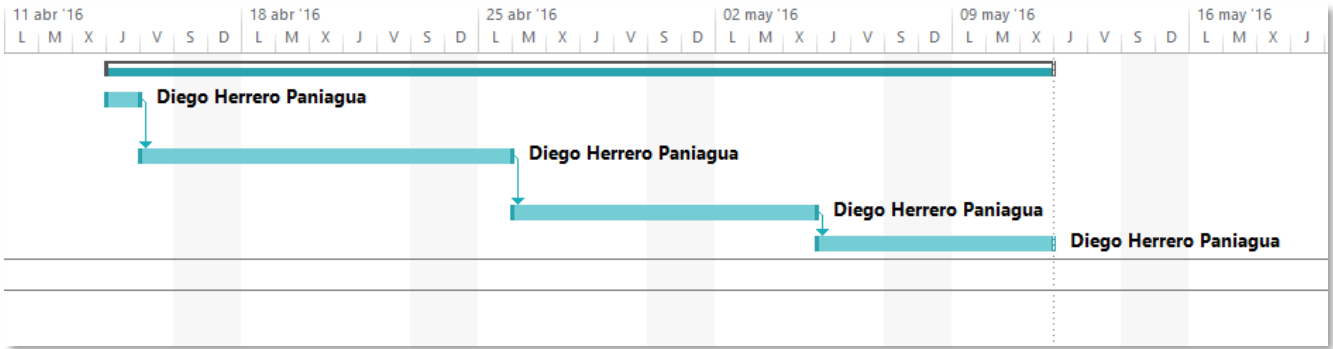


Ilustración 5. Diagrama de Gantt de la Creación de Regla Java

Estimación de Costes

El coste de un proyecto como este es una de las estimaciones más importantes a realizar, ya que es una aproximación a la valoración real del trabajo realizado por el alumno, así como del software y el hardware que ha tenido que usar.

Coste del hardware			
Hardware	Coste unitario (€)	Cantidad	Coste total
Acer Aspire 5750	499 €	1	499 €
Asus X555LJ	715 €	1	715 €
Total			1214 €

Tabla 15. Coste del hardware

El software utilizado es en su totalidad gratuito, gracias a las licencias educativas proporcionadas por la Universidad de Valladolid, a licencias libres y a la posesión previa de dicho software. La relación completa del software utilizado se puede ver en la Tabla 16.

Coste del software			
Hardware	Coste unitario (€)	Cantidad	Coste total
Windows 10 Pro	0 €	1	0 €
Microsoft Office 2016	0 €	1	0 €
Microsoft Project 2013	0 €	1	0 €
Eclipse Kepler	0 €	1	0 €
PMD	0 €	1	0 €
Total			0 €

Tabla 16. Coste del software

Coste de los recursos humanos			
Hardware	Coste por hora (€)	Cantidad (horas x días)	Coste total
Gestor del proyecto	30 €	1x90	2700 €
Analista	25 €	1x90	2250 €
Desarrollador	20 €	3x40	2400 €
Subtotal			7350 €
Ajuste del 20% para gastos inesperados			1470 €
Total			8820 €

Tabla 17. Coste de los recursos humanos

Coste total	
Hardware	1214 €
Software	0 €
Recursos humanos	8820 €
Total	10034 €

Tabla 18. Coste total

El presupuesto total del proyecto asciende a **diez mil treinta y cuatro euros**.

CAPÍTULO III: DESCRIPCIÓN

Descripción

La herramienta PMD nació en junio del año 2002 de forma muy modesta y con el objetivo de facilitar la actividad del desarrollador de código (en una primera instancia, en lenguaje Java) para prevenir posibles bugs y vulnerabilidades, así como adecuar el estilo del código desarrollado a un estándar.^[1]



Ilustración 6. Logo de PMD

Con una primera versión muy sencilla, limitada a detectar, por ejemplo, cuándo se dejaba vacía la captura de una excepción, y mostrando los resultados del análisis en texto plano, PMD ha ido creciendo hasta abarcar numerosas funcionalidades y lenguajes.^[2]

El fundamento de PMD es el análisis del código fuente mediante un conjunto de reglas, que abarcan desde la detección de posibles bugs hasta el descubrimiento de fragmentos de código no usados, pasando por el análisis de la complejidad de una clase.

No obstante, PMD solo es capaz de analizar un fichero de código fuente a la vez, por lo que, por ejemplo, ver si una clase es usada en algún punto de un proyecto actualmente es imposible, aunque si es una de las líneas rojas de su equipo de desarrollo.^[3]

Aun así, al ser un proyecto de Licencia Pública General Reducida de GNU, las aportaciones al crecimiento del mismo pueden venir, además del propio equipo de desarrollo, de cualquiera que esté interesado en mejorarlo, tanto con la creación de nuevas reglas como dando soporte a otros lenguajes de programación.^[4]

Como curiosidad, cabe destacar que ni el propio equipo de desarrollo da un significado concreto a las siglas PMD. Sin embargo, la versión más aceptada es *Programming Mistake Detector* (Detector de Errores en la Programación).

Funcionalidades

Para el caso de estudio de este proyecto, el uso de PMD para analizar código Java, nos encontramos que el catálogo de reglas para el mismo. No obstante, y actualmente, PMD cuenta con soporte para un gran número de lenguajes, como pueden ser JavaScript, C++, C#, PL/SQL, PHP, Python... Para la mayoría de ellos solo está disponible un módulo de reciente creación de detección de copia de código (CPD, se detallará más adelante), aunque para alguno de ellos, como PL/SQL, sí que existe un catálogo de reglas, si bien es más limitado, ya que el objetivo principal del proyecto PMD es el lenguaje Java.

Además, en la definición de cada regla viene establecida una prioridad, que representa la gravedad del fallo o vulnerabilidad detectado

Y, volviendo a Java, a continuación, se detallarán las categorías en las que se divide su catálogo de reglas con un ejemplo de cada una de ellas, que, si bien es muy amplio, a la hora de ejecutar un análisis se pueden definir qué reglas queremos ejecutar y cuáles no:

- **Android**: orientadas específicamente al desarrollo de aplicaciones móviles, sobre todo en la aplicación de buenas prácticas en su generación.
Un ejemplo de ello es la regla `DoNotHardCodeSDCard`, en la que básicamente se detecta el uso, al nombrar un directorio, de la nomenclatura `/sdcard`, cuando es más recomendable utilizar el método `Environment.getExternalStorageDirectory()` para obtener la ruta al almacenamiento externo del dispositivo.
- **Basic**: esta categoría comprende reglas que deberían ser de conocimiento básico para todo programador, aunque bien es cierto que el mejor de los programadores puede tener un desliz.
Por ejemplo, para el uso de condicionales como `if`, se tienen en cuenta criterios como que no se declare un `if` cuando se tiene la certeza de que la condición siempre va a ser verdadera o falsa (`UnconditionalIfStatement`), o la sucesión de dos o más `if` cuando podrían estar comprendidos en uno solo mediante el uso de operadores condicionales (`CollapsibleIfStatements`).
- **Braces**: tiene en cuenta la presencia o no de llaves en el bloque de ejecución de un `if` (`IfStmtsMustUseBraces`) o un `while` (`WhileLoopsMustUseBraces`), entre otros.
- **Clone Implementation**: recoge consideraciones acerca del uso de método `clone()`, como que la clase que lo utiliza deba implementar la clase `Cloneable` (`CloneMethodMustImplementCloneable`).
- **Code Size**: para esta categoría, las reglas atienden a la longitud de los métodos (`ExcessiveMethodLength`), las clases (`ExcessiveClassLength`), o incluso la cantidad de parámetros que se le pasa a un método (`ExcessiveParameterList`). En la implementación actual, se considera excesivo 100 líneas, 1000 líneas y 10 parámetros respectivamente.

- **Comments**: las reglas pertenecientes a esta categoría se fijan en los comentarios escritos a lo largo del código, fijándose tanto en su longitud (`CommentSize`) como en su contenido (`CommentContent`). Para esta última, en su implementación se han definido una serie de palabras para que los comentarios sean políticamente correctos.
- **Controversial**: son reglas que, básicamente, están pensadas para servir de apoyo o complementar a los paquetes de reglas personalizados por el usuario. Algunos ejemplos de ellas detectan los paréntesis innecesarios (`UnnecessaryParentheses`) o la declaración de varias variables en una sola línea (`OneDeclarationPerLine`).
- **Coupling**: las reglas contenidas en esta categoría tienen en cuenta el acoplamiento entre clases, por ejemplo, cuando se hacen importaciones de muchas clases contenidas en un mismo paquete, cuando se podía haber importado directamente el paquete (`ExcessiveImports`).
- **Design**: en este caso, se buscan implementaciones que sean susceptibles de ser optimizadas, como tener un gran número de `if` anidados (`AvoidDeeplyNestedIfStmts`) o la ausencia del `default` en la declaración de un `switch` (`SwitchStmtsShouldHaveDefault`).
- **Empty Code**: como su propio nombre indica, esta categoría contiene reglas que buscan bloques de código entre llaves vacío, ya sean bucles (`EmptyWhileStmt`), condiciones (`EmptyIfStmt`), etc...
- **Finalizer**: detecta posibles problemas que pueden surgir a la hora de usar métodos `finalize`, como el paso de parámetros a dicho método (`FinalizeOverloaded`) o cualquier otra declaración distinta de `protected` al implementarlo (`FinalizeShouldBeProtected`).
- **Import Statements**: las reglas contenidas en esta categoría analizan las clases y paquetes importados en una determinada clase, viendo así si se usan a lo largo de ella o no (`UnusedImports`), o si están duplicados (`DuplicateImports`).
- **J2EE**: a esta categoría pertenecen reglas especialmente diseñadas para implementaciones de la plataforma Java Enterprise Edition, como la detección del uso de hilos (`DoNotUseThreads`), considerada una mala práctica en esta plataforma.
- **Jakarta Commons Logging**: se trata de otra categoría específica de código desarrollado bajo el framework del proyecto Jakarta, desarrollador de código abierto para Java, y como indica el nombre de la categoría, atiende al uso correcto del logging específico de este proyecto (`UseCorrectExceptionLogging`).^[11]
- **JavaBeans**: estas reglas detectan malas implementaciones de JavaBeans, estructuras diseñadas para encapsular varios objetos dentro de otro objeto, como, por ejemplo, que deben implementar la interfaz `Serializable` (`BeanMembersShouldSerialize`).

- Java Logging: a diferencia de la categoría relacionada con el proyecto Jakarta, las reglas de esta clase se fijan en las líneas de log comunes de Java. Así, una línea de log implementada mediante un `System.Out.Println` es considerada una mala práctica, teniendo que usar en su lugar la salida estándar de log (`SystemPrintln`).
- JUnit: los test destinados a probar un determinado código también son susceptibles de ser analizados, y estos deben ser declarados de manera pública y estática (`JUnitStaticSuite`).
- Migration: esta categoría contiene un conjunto de reglas destinadas a detectar código migrado desde versiones anteriores de Java, para así usar la nueva versión. Así, en vez de usar un `Vector` se usa una `List` (`ReplaceVectorWithList`), o en lugar de utilizar objetos de tipo `Hashtable`, utilizarles de tipo `Map` (`ReplaceHashtableWithMap`).
- Naming: como su propio nombre indica, las reglas de esta categoría analizan la nomenclatura usada en variables, métodos, etc..., para que sea acorde al estándar (`camelCase`) y fácil de leer e identificar por el usuario (ni nombres muy cortos, ni demasiado largos).
- Optimization: estas reglas detectan optimizaciones genéricas en el código, como la declaración de una variable como `final` si se le asigna valor una sola vez (`LocalVariableCouldBeFinal`) o el uso de la clase `StringBuffer` para concatenar `Strings` (`UseStringBufferForStringAppends`).
- Security Code Guidelines: esta categoría recoge la implementación de una serie de reglas definidas por Oracle en su página web con el objetivo de producir código seguro, como puede ser el no devolver directamente el `Array` interno de una clase, sino una copia del mismo (`MethodReturnsInternalArray`).^[12]
- Strict Exceptions: las excepciones y su tratamiento son susceptibles de ser analizadas. Así, se puede detectar si una excepción capturada se vuelve a lanzar (`AvoidRethrowingException`) o si una excepción es lanzada desde el `finally` de un bloque `try/catch` (`DoNotThrowExceptionInFinally`).
- String and StringBuffer: todas las reglas contenidas en esta categoría están destinadas a analizar cómo se usan objetos de las clases `String`, `StringBuffer` y `StringBuilder`. Por ejemplo, el aplicar el método `toString()` a un `String` no es necesario, porque ya es un `String` (`StringToString`); otro ejemplo, el usar comillas simples en lugar de dobles a la hora de concatenar un carácter en un objeto de la clase `StringBuffer` (`AppendCharacterWithChar`).
- Type Resolution: esta categoría se refiere al uso correcto de las clases, para así evitar lanzar excepciones o crear objetos demasiado genéricos, que pueden dificultar la comprensión del código (`SignatureDeclareThrowsException`).

- Unnecessary: el objetivo de las reglas de esta categoría es, simplemente, detectar fragmentos de código innecesarios, como un `return` en un método `void` (`UnnecessaryReturn`), o declarar como `final` una variable cuando la clase que la contiene ya es `final` (`UnnecessaryFinalModifier`).
- Unused Code: similar a la categoría anterior, esta se encarga de encontrar fragmentos de código que no se usan, ya sean variables locales (`UnusedLocalVariable`) o métodos propios de la clase (`UnusedPrivateMethod`).

Como se ha podido apreciar, la nomenclatura de las reglas identifica claramente su propósito, por lo que en el resultado del análisis resulta fácil de identificar y resolver el fallo o vulnerabilidad detectado.

CAPÍTULO IV: INTEGRACIÓN Y USO

Existen numerosas formas de utilizar la herramienta PMD, desde la más simple, por línea de comandos, hasta su integración con otras aplicaciones, ya sean entornos de desarrollo como Eclipse, o software de integración continua como Jenkins. A lo largo de este capítulo se tratarán estos posibles usos.

Línea de Comandos

La manera más sencilla de usar la herramienta PMD es a través de una consola de comandos, bien en Windows, bien en Unix/Linux. Una vez descargado el código fuente de PMD, nos encontramos una serie de carpetas (como se puede ver en la Ilustración 7), entre ellas la carpeta `bin`, que contiene los comandos que se van a usar.

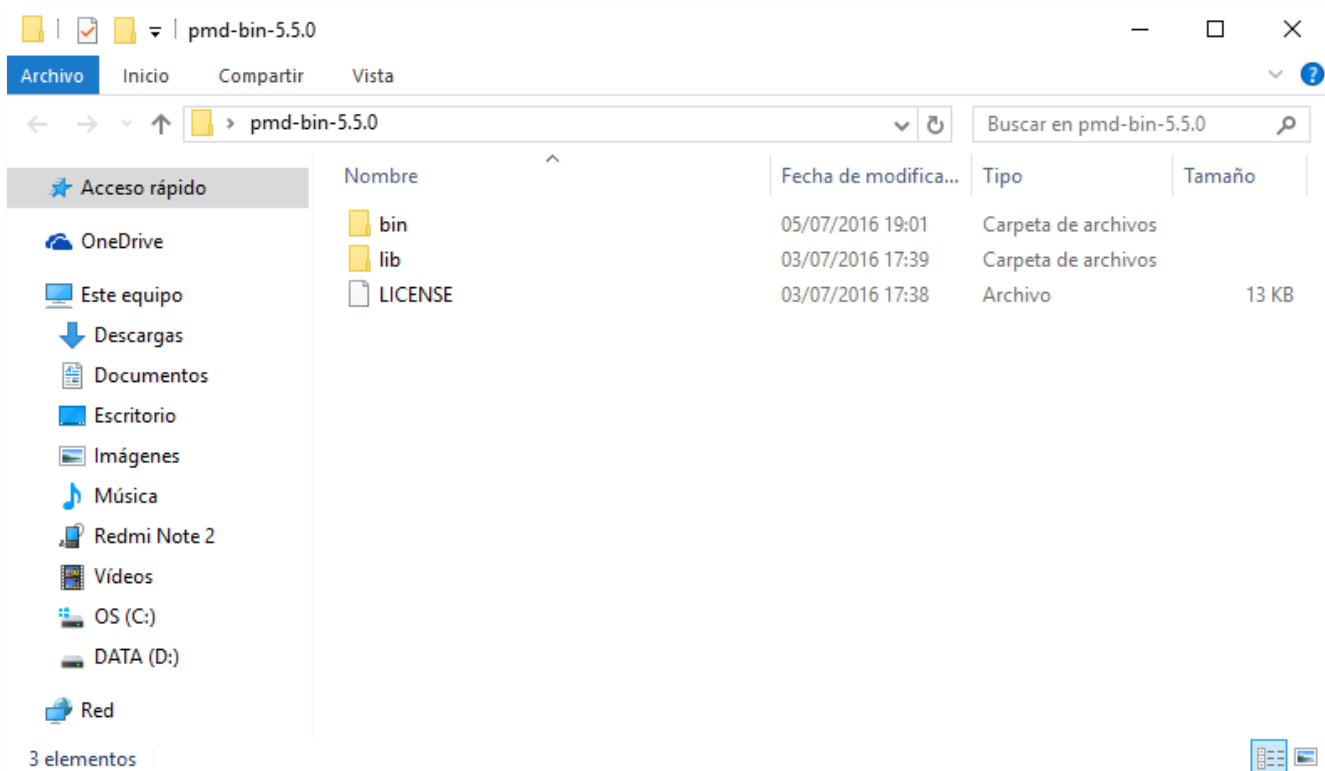


Ilustración 7. Carpeta raíz de PMD

Mientras que en Windows basta con invocar al fichero con extensión `.bat`, en Unix/Linux es necesario usar el script `run.sh` seguido del comando. En uno u otro caso, los parámetros que recibe son los mismos.

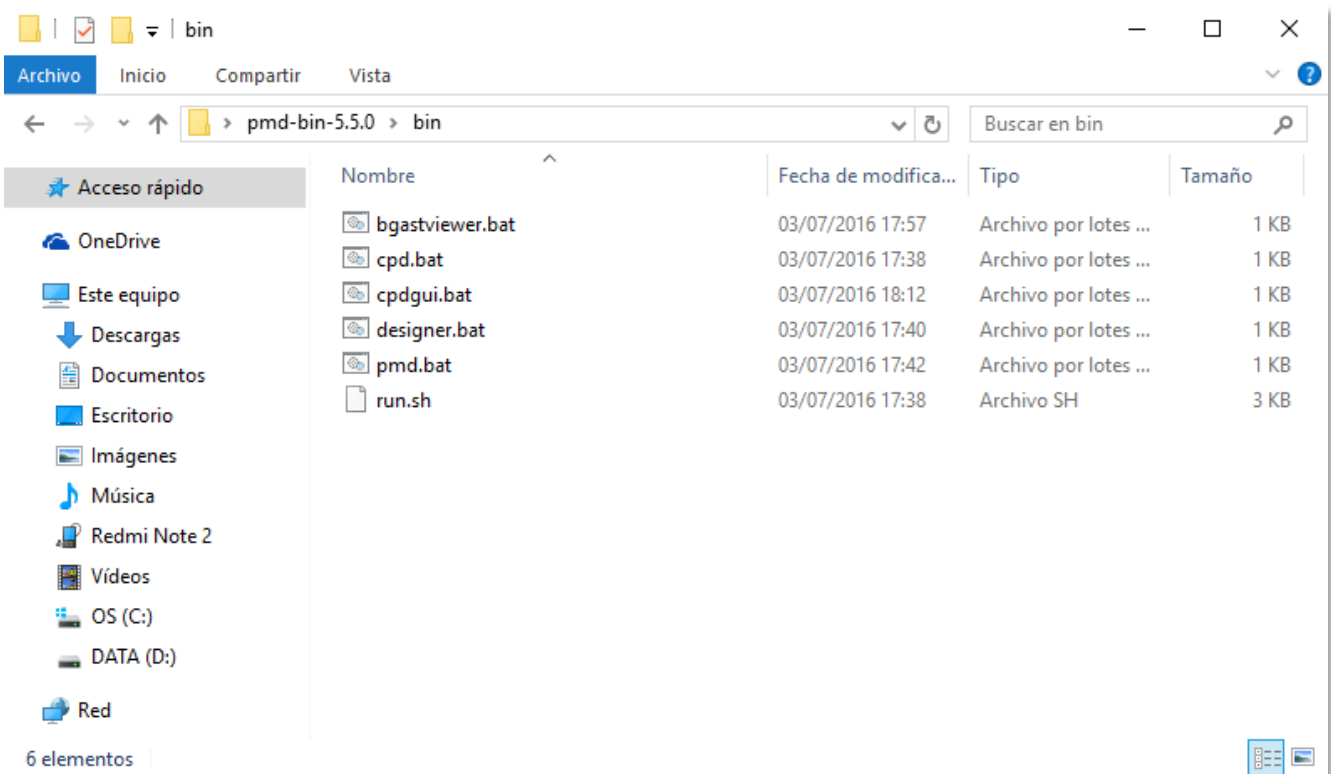


Ilustración 8. Contenido de la carpeta `bin`

Además, en la carpeta `bin` se encuentran utilidades que facilitan la creación de nuevas reglas, que serán comentadas en el próximo capítulo.

Por familiaridad con el sistema operativo, se describirá el uso de los comandos en Windows.

PMD

La sintaxis del comando `pmd.bat` es muy sencilla, y requiere de, al menos, tres parámetros:

- `-d/-dir`: para indicar el directorio o la ruta completa al fichero a analizar.
- `-f/-format`: para escoger el formato del informe.
- `-R/-rulesets`: para definir qué conjuntos de reglas (separados por comas) se van a utilizar en el análisis.

Además, cuenta con otras opciones, como, por ejemplo `-e/-encoding`, para definir la codificación de los ficheros de entrada, o `-filelist`, para especificar una serie de ficheros susceptibles de ser analizados dentro de un directorio.

```
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f text -R rulesets/java/braces.xml
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:44:  Avoid using if statements without curly braces
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:47:  Avoid using if statements without curly braces
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:51:  Avoid using if statements without curly braces
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:54:  Avoid using if statements without curly braces
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:59:  Avoid using if statements without curly braces
D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java:62:  Avoid using if statements without curly braces
```

Ilustración 9. Ejemplo de ejecución del comando `pmd.bat`

Por defecto, la salida se muestra por pantalla, pero si se quiere recoger en un fichero, se puede usar `-r/reportfile` para indicar el fichero de salida o bien redirigirla directamente a un fichero, como se puede ver en la Ilustración 10.

```
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f text -R rulesets/java/braces.xml -reportfile prueba.txt
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f text -R rulesets/java/braces.xml > prueba2.txt
```

Ilustración 10. Ejemplo de redirección de salida

Los formatos de salida son muy diversos, desde el texto plano hasta una página web, pasando por texto separado por comas, importable en Excel, o un fichero XML listo para ser parseado. Para escoger el formato de la salida, basta con modificar el valor del parámetro `-f`, que puede tomar valores como `text`, `textcolor`, `xml`, `html`, `csv`...

```
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f html -R rulesets/java/braces.xml -reportfile prueba.html
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f xml -R rulesets/java/braces.xml -reportfile prueba.xml
```

Ilustración 11. Ejemplo de las salidas `xml` y `html`

```
D:\Escritorio\pmd-bin-5.5.0\bin>pmd.bat -d ..\pruebas -f textcolor -R rulesets/java/braces.xml
* file: D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java
  src: maquina_enigma.java:44:44
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(P[1]<0)P[1]+=26;

  src: maquina_enigma.java:47:47
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(P[0]<0)P[0]+=26;

  src: maquina_enigma.java:51:51
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(X>25)X-=26;

  src: maquina_enigma.java:54:54
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(X<0)X+=26;

  src: maquina_enigma.java:59:59
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(X>25)X-=26;

  src: maquina_enigma.java:62:62
  rule: IfStmtsMustUseBraces
  msg: Avoid using if statements without curly braces
  code: if(X<0)X+=26;

Summary:
: 6
* warnings: 6
```

Ilustración 12. Salida `textcolor`

```

<?xml version="1.0" encoding="UTF-8"?>
- <pmd timestamp="2016-07-05T19:40:01.761" version="5.5.0">
  - <file name="D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java">
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="51" begincolumn="33" endline="44" beginline="44"> Avoid using if statements
      without curly braces </violation>
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="43" begincolumn="25" endline="47" beginline="47"> Avoid using if statements
      without curly braces </violation>
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="46" begincolumn="33" endline="51" beginline="51"> Avoid using if statements
      without curly braces </violation>
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="45" begincolumn="33" endline="54" beginline="54"> Avoid using if statements
      without curly braces </violation>
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="46" begincolumn="33" endline="59" beginline="59"> Avoid using if statements
      without curly braces </violation>
    <violation priority="3" externalInfoUrl="https://pmd.github.io/pmd-5.5.0/pmd-java/rules/java/braces.html#IfStmtsMustUseBraces" method="enigma"
      class="maquina_enigma" ruleset="Braces" rule="IfStmtsMustUseBraces" endcolumn="45" begincolumn="33" endline="62" beginline="62"> Avoid using if statements
      without curly braces </violation>
  </file>
</pmd>

```

Ilustración 13. Salida `xml`

#	File	Line	Problem
1	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	44	Avoid using if statements without curly braces
2	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	47	Avoid using if statements without curly braces
3	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	51	Avoid using if statements without curly braces
4	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	54	Avoid using if statements without curly braces
5	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	59	Avoid using if statements without curly braces
6	D:\Escritorio\pmd-bin-5.5.0\pruebas\maquina_enigma.java	62	Avoid using if statements without curly braces

Ilustración 14. Salida `html`

Cabe destacar que, si se usa un conjunto de reglas predefinido, la página web generada contendrá enlaces directos a la descripción de la regla que ha detectado el fallo en el sitio web del proyecto PMD, como se puede ver en las Ilustraciones 14 y 15.

The Braces ruleset contains rules regarding the use and placement of braces.

IfStmtsMustUseBraces

Since: PMD 1.0
Priority: 3

Avoid using if statements without using braces to surround the code block. If the code formatting or indentation is lost then it becomes difficult to separate the code being controlled from the rest.

```
//IfStatement[count(*) < 3][not(Statement/Block)]
```

Example(s):

```
if (foo)    // not recommended
    x++;

if (foo) {  // preferred approach
    x++;
}
```

Ilustración 15. Descripción de regla en el sitio web de PMD

CPD

Como caso particular, y debido a que se ha incorporado en las últimas versiones de la implementación de la herramienta PMD, se hará una pequeña mención al módulo de detección de copia CPD.

Este comando requiere como mínimo, en este caso, dos parámetros:

- `--minimum-tokens`: indica la cantidad mínima de elementos sobre la que queremos encontrar duplicidades.
- `--files`: establece el directorio en el que se quieren buscar las duplicidades.

```
D:\Escritorio\pmd-bin-5.5.0\bin>cpd.bat --minimum-tokens 100 --files ..\pruebas
Found a 18 line (138 tokens) duplication in the following files:
Starting at line 120 of D:\Escritorio\pmd-bin-5.5.0\bin\..\pruebas\maquina_enigma.java
Starting at line 8 of D:\Escritorio\pmd-bin-5.5.0\bin\..\pruebas\otroFicheroQueNoEsLaMaquinaEnigma.java

    public static int[] anum (String a){
        int b[] = new int[a.length()];
        for(int i =0;i<a.length();i++){
            b[i] = a.charAt(i)-65;}
        return b;
    }
//Método para comprobar que la palabra criba y el texto a desencriptar cumple con las condiciones preestablecidas
public static void Comprobacion (String a){
    Scanner lector=new Scanner(System.in);
    for (int i =0;i<a.length();i++){
        while (a.charAt(i)<65 || a.charAt(i)>90){
            System.out.println("Deben ser letras en mayuscula de la A a la Z. Introdúcelo otra vez: ");
            a = lector.nextLine();
            i=0;
        }
    }
}
```

Ilustración 16. Ejemplo de ejecución del comando `cpd.bat`

Como se puede observar en la Ilustración 16, la salida de este comando nos indica la cantidad de líneas duplicadas encontrada, los ficheros afectados y el número de línea de cada fichero a partir del cual se ha detectado el código copiado, además del código duplicado en sí mismo.

La salida estándar se muestra por pantalla en formato `text`, pero de la misma forma que con el comando `pmd.bat`, se puede redirigir la salida a un fichero y especificar el formato, siendo menos amplio el abanico de posibilidades en este caso (`text`, `csv` y `xml`).

Integración con Eclipse

Una de las herramientas básicas de un programador informático es el entorno de desarrollo (IDE a partir de ahora). Gracias a él, es posible tener un acceso rápido a las funciones y clases de un proyecto, detectar fácilmente errores en el código, desarrollar en multitud de lenguajes con las ayudas propias de cada uno de ellos...

Entre los disponibles en el mercado podemos encontrar Eclipse como uno de los más populares. Además de las funciones ya descritas de los IDEs, Eclipse cuenta con un amplísimo catálogo de *plugins* que facilitan aún más la labor del desarrollador. Y en ese catálogo se encuentra también uno de la herramienta PMD. Para instalarlo en Eclipse, basta con abrir el *Eclipse Marketplace* desde el menú Ayuda. Una vez allí, solo tenemos que buscarlo e instalarlo.^{[13][14]}

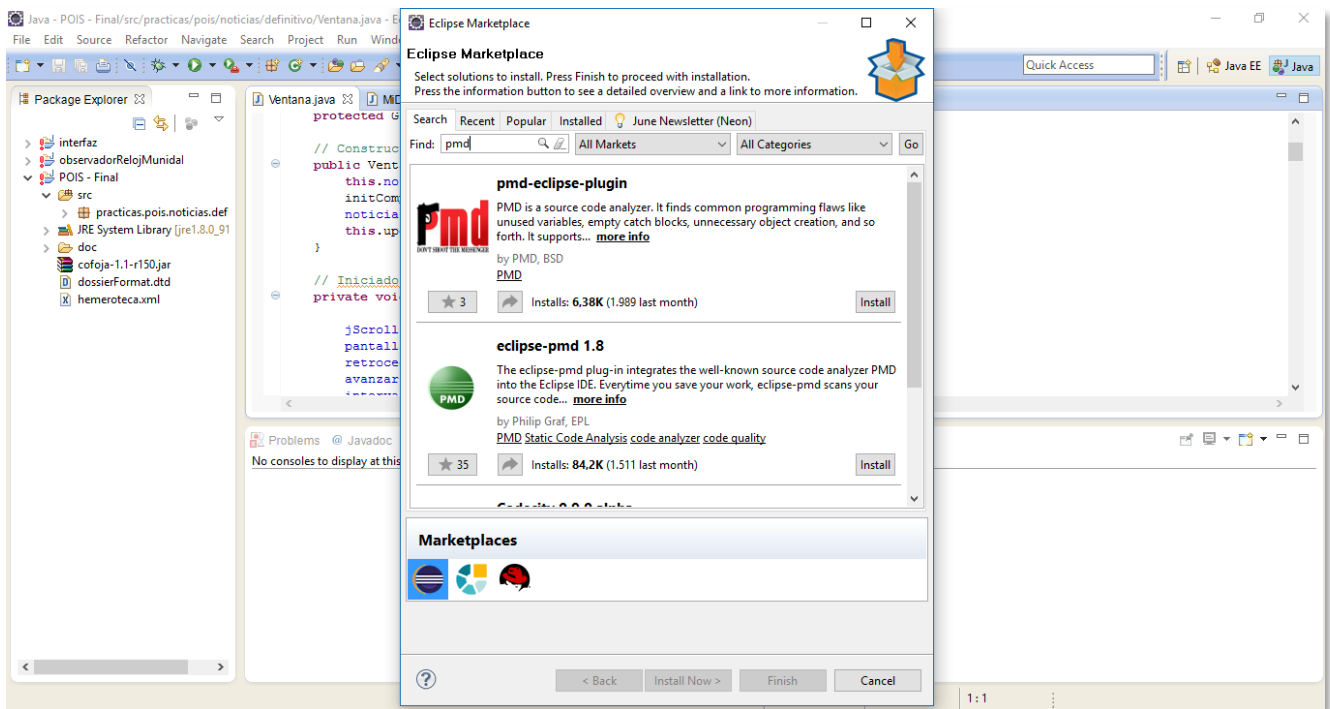


Ilustración 17. Eclipse Marketplace

Una vez instalado, y con el fin de analizar nuestro código, simplemente se presionaría el botón derecho del ratón para ver el menú contextual, localizar el menú de PMD y seleccionar la opción *Check Code*. Es posible, también, realizar la misma acción sobre el proyecto en vez de sobre un único fichero, analizando así todo el código fuente del mismo.

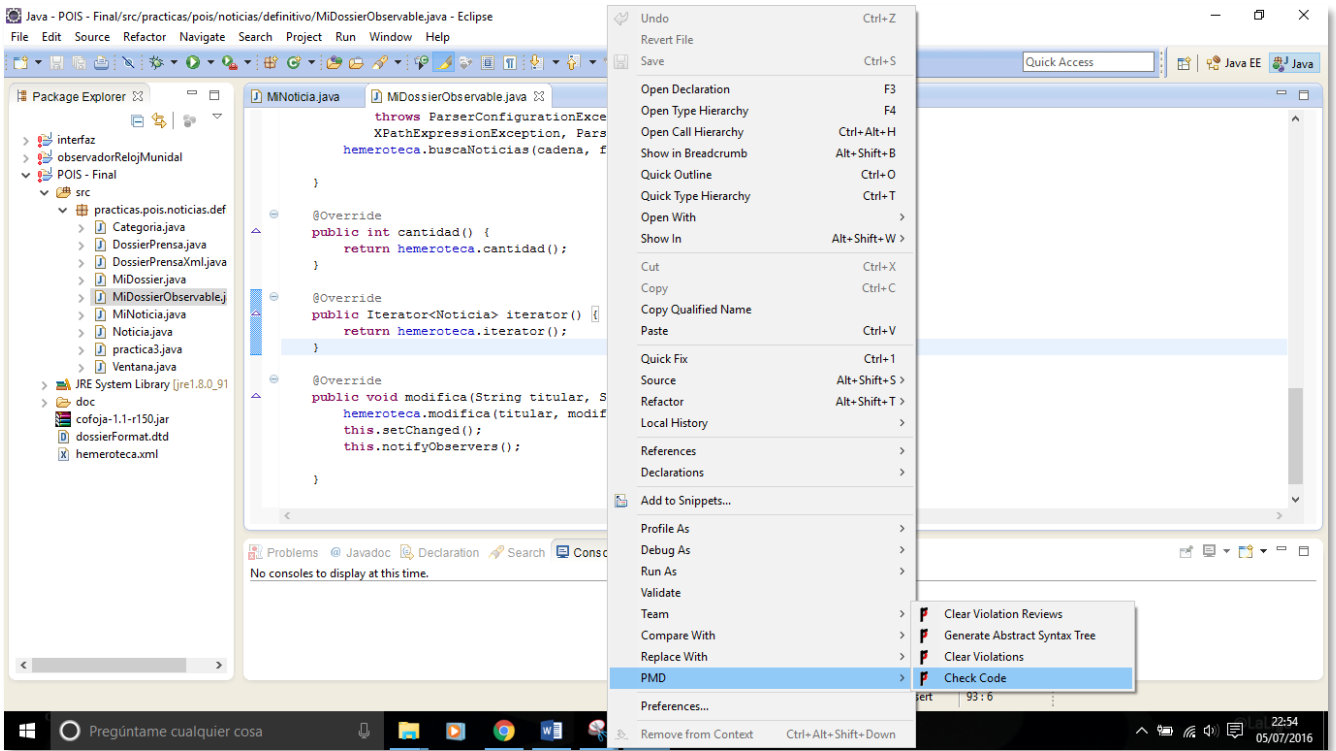


Ilustración 18. Opción Check Code

Tras unos pocos segundos de análisis, se abrirá una nueva perspectiva específica para PMD en la que podremos ver:

- En el código, las líneas en las que se incumplen las reglas predefinidas en PMD.

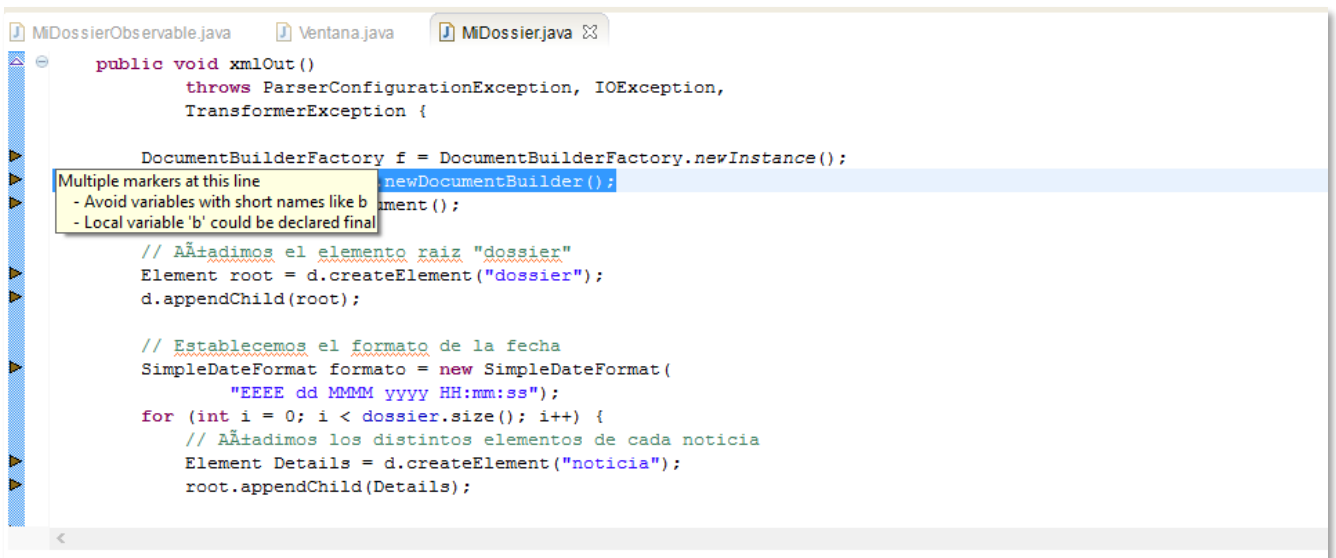


Ilustración 19. Fallos señalados en el código

- En la ventana *Violations Outline*, un listado de todos los fallos detectados en el fichero seleccionado, junto con el mensaje asociado y la línea en la que se han descubierto, así como su prioridad.

P	Line	created	Rule	Error Message
▶	302	Tue Jul 05 22:56:33 ...	VariableNamingConventions	Variables should :
▶	205	Tue Jul 05 22:56:33 ...	AvoidReassigningParameters	Avoid reassigning
▶	105	Tue Jul 05 22:56:33 ...	SystemPrintln	System.out.printl
▶	205	Tue Jul 05 22:56:33 ...	AvoidReassigningParameters	Avoid reassigning
▶	206	Tue Jul 05 22:56:33 ...	AvoidReassigningParameters	Avoid reassigning
▶	247	Tue Jul 05 22:56:33 ...	SystemPrintln	System.out.printl
▶	290	Tue Jul 05 22:56:33 ...	ShortVariable	Avoid variables w
▶	307	Tue Jul 05 22:56:33 ...	LawOfDemeter	Potential violatio
▶	166	Tue Jul 05 22:56:33 ...	AvoidDuplicateLiterals	The String literal
▶	330	Tue Jul 05 22:56:33 ...	ShortVariable	Avoid variables w
▶	187	Tue Jul 05 22:56:33 ...	MethodArgumentCouldBeFinal	Parameter 'fecha
▶	285	Tue Jul 05 22:56:33 ...	CommentRequired	publicMethodCo
▶	236	Tue Jul 05 22:56:33 ...	LawOfDemeter	Potential violatio
▶	331	Tue Jul 05 22:56:33 ...	ShortVariable	Avoid variables w
▶	306	Tue Jul 05 22:56:33 ...	LawOfDemeter	Potential violatio
▶	176	Tue Jul 05 22:56:33 ...	AvoidInstantiatingObjectsInLoo...	Avoid instantiatir
▶	236	Tue Jul 05 22:56:33 ...	LawOfDemeter	Potential violatio
▶	291	Tue Jul 05 22:56:33 ...	ShortVariable	Avoid variables w
▶	122	Tue Jul 05 22:56:33 ...	ShortVariable	Avoid variables w

Ilustración 20. Listado de fallos

- En la ventana *Violations Overview*, una serie de estadísticas relacionadas con el número de fallos detectados por fichero y por regla incumplida. Si el análisis se ha ejecutado sobre el proyecto al completo, las estadísticas que se mostrarán incluirán todos los ficheros del mismo.

Element	# Violations	# Violations/KLOC	# Violations/Method	Project
practicas.pois.noticias.definitivo	551	991.0	8.22	POIS - Final
> DossierPrensa.java	5	312.5	0.71	POIS - Final
> DossierPrensaXml.java	4	444.4	1.00	POIS - Final
> MiDossier.java	241	1158.7	18.54	POIS - Final
> MiDossierObservable.java	38	690.9	2.92	POIS - Final
> MiNoticia.java	41	953.5	4.10	POIS - Final
> Noticia.java	3	272.7	0.38	POIS - Final
▶ CommentRequired	1	90.9	0.12	POIS - Final
▶ CommentSize	1	90.9	0.12	POIS - Final
▶ ShortVariable	1	90.9	0.12	POIS - Final
> practica3.java	35	760.9	35.00	POIS - Final
> Ventana.java	184	1101.8	16.73	POIS - Final

Ilustración 21. Estadísticas de fallos

Si queremos generar un informe con los resultados del análisis del proyecto completo, basta con abrir el menú contextual del mismo, y dentro del menú de PMD, seleccionar la opción *Generate Reports*.

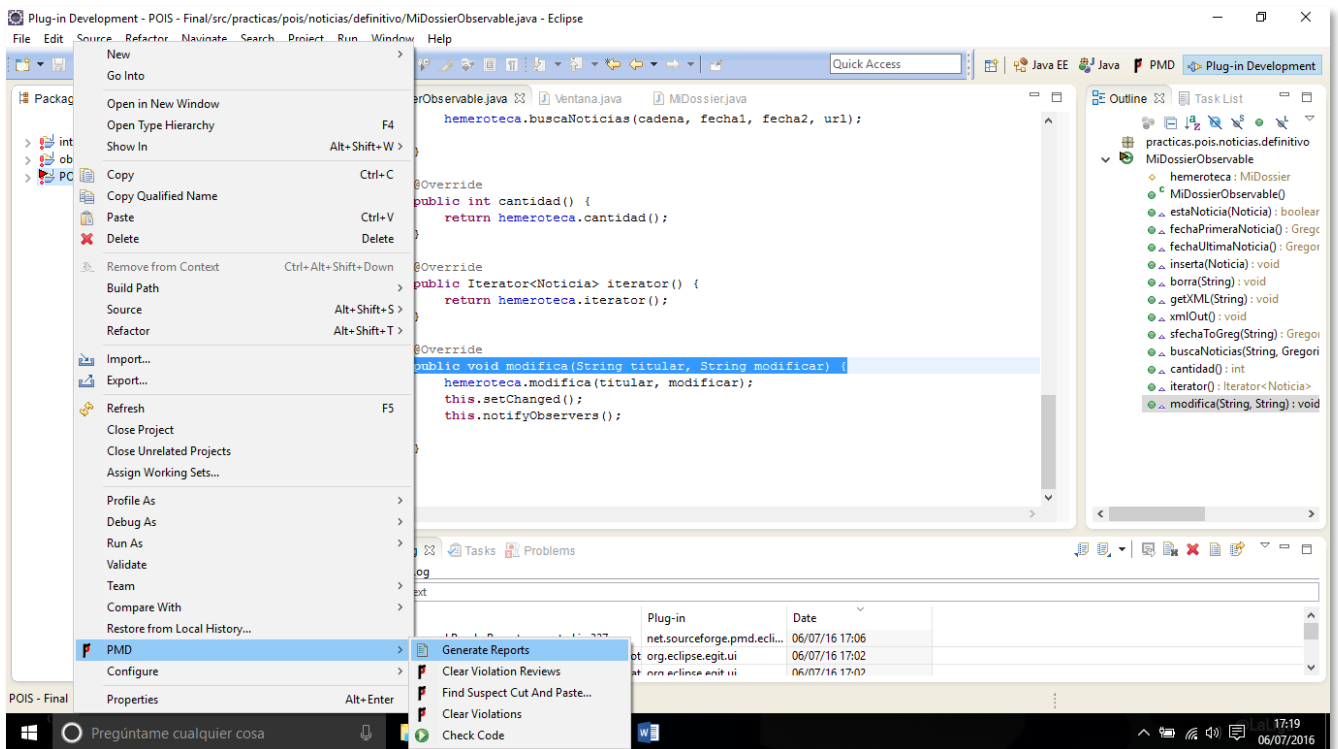


Ilustración 22. Opción *Generate Reports*

Tras su ejecución, se creará una carpeta dentro de nuestro proyecto, denominada `reports`, con los informes generados. En ella, encontraremos los resultados del análisis en diversos formatos, dependiendo las opciones seleccionadas en las preferencias del plugin de PMD, como se puede apreciar en la Ilustración 24.

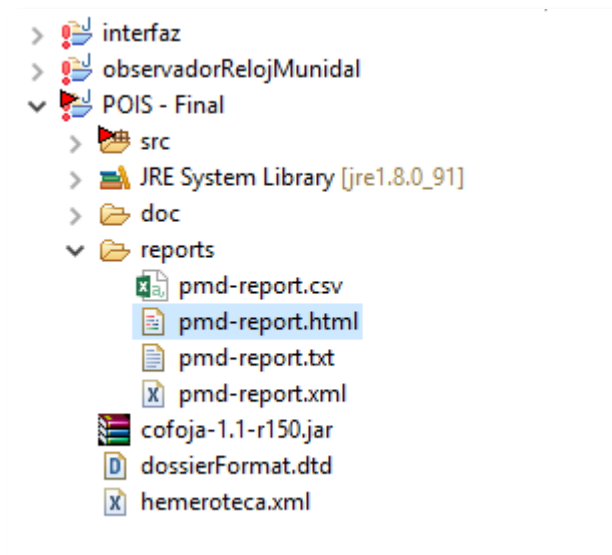


Ilustración 23. Carpeta `reports`

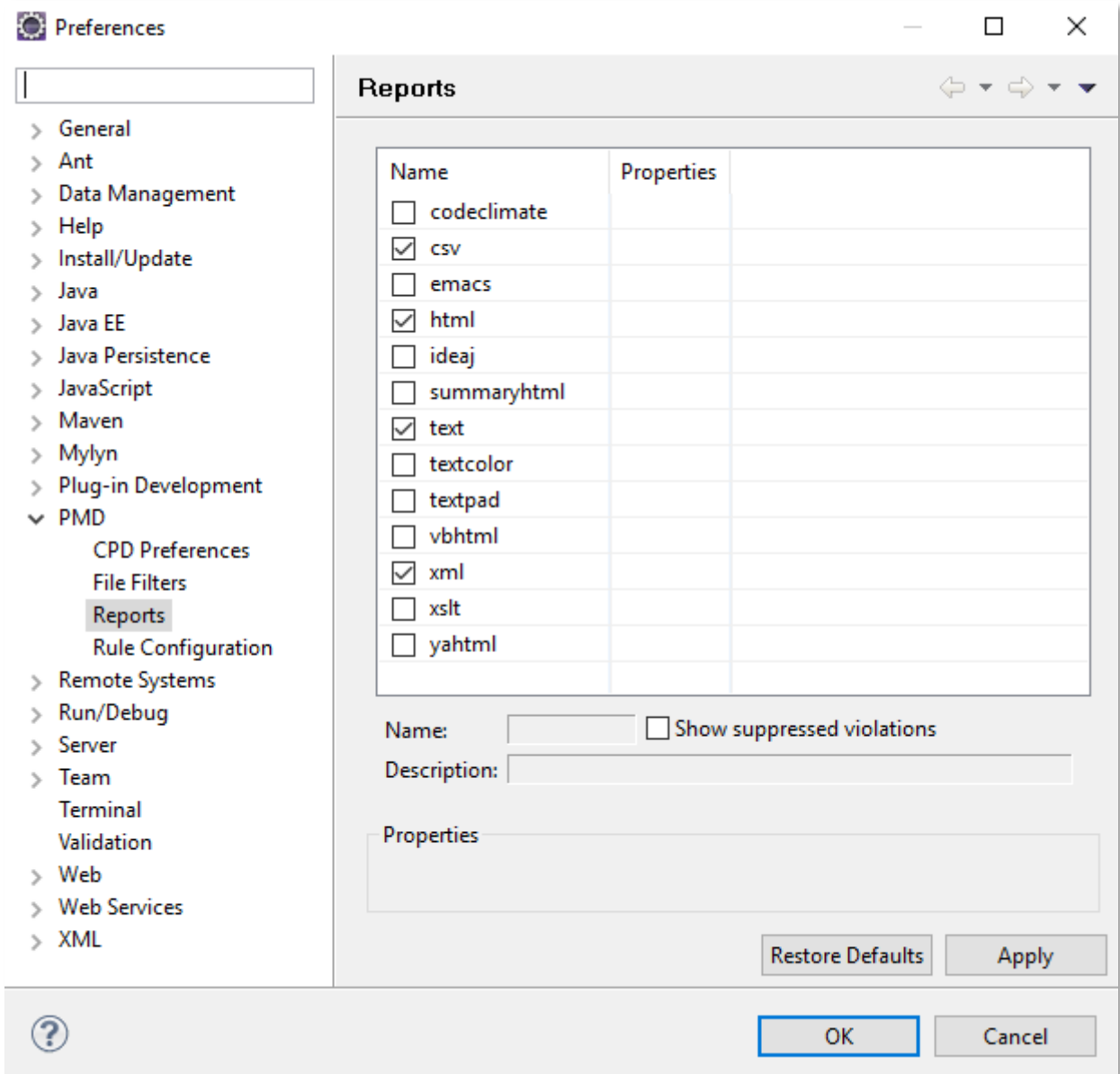


Ilustración 24. Opciones de informes del plugin PMD en Eclipse

Integración con Jenkins

Jenkins es un software de integración continua mediante el cual se pueden realizar compilaciones y ejecutar pruebas con el fin de detectar fallos lo antes posible. Dicho proceso puede estar programado para llevarse a cabo cada cierto tiempo o cada vez que se sube un cambio al servidor de control de versiones, como puede ser Git o Subversion.^{[15][16]}

Para realizar una labor más completa, Jenkins ofrece la posibilidad de incorporar plugins de otras herramientas, entre ellas PMD.

The screenshot shows the Jenkins main dashboard. At the top, it says 'Jenkins' and 'DIEGO HERRERO PANIAGUA'. Below that, there's a search bar and a 'Todo' button. The main content is a table of jobs:

S	W	Nombre ↓	Último Éxito	Último Fallo	Última Duración
●	☀	pa15	4 días 2 Hor - #909	5 días 21 Hor - #693	2 Min 39 Seg
●	☀	pa16	22 Hor - #706	5 días 21 Hor - #688	3 Min 41 Seg
●	☀	pe15	12 días - #1127	N/D	5 Min 7 Seg
●	☀	pe16	21 Hor - #2085	N/D	9 Min 31 Seg

Below the table, there are icons for 'S', 'M', and 'L'. On the left side, there are sections for 'Trabajos en la cola' (No hay trabajos en la cola) and 'En ejecución' (1 Inactivo, 2 Inactivo, 3 Inactivo).

Ilustración 25. Pantalla principal de Jenkins

Nada más entrar en Jenkins, nos encontramos con un listado de todos los proyectos a los que tenemos acceso, así como su estado (un círculo de color verde si la última compilación ha sido correcta, rojo en caso contrario) y su evolución (desde un icono tormentoso a uno de sol, en función de la evolución de los fallos de compilación observados en ejecuciones pasadas). Además, en la parte izquierda, se recogen los proyectos que se encuentran analizándose.

The screenshot shows the Jenkins project page for 'pe16'. On the left, there's a sidebar with navigation options like 'Volver al Panel de Control', 'Estado Actual', 'Zona de Trabajo', 'FindBugs Warnings', 'PMD Warnings', 'Java Warnings', and 'Open Tasks'. Below that is a 'Historia de tareas' table:

Estado	ID	Fecha y Hora
●	#2085	04-jul-2016 14:37
●	#2084	04-jul-2016 13:25
●	#2083	04-jul-2016 12:24
●	#2082	04-jul-2016 11:23
●	#2081	04-jul-2016 9:19
●	#2080	04-jul-2016 9:10
●	#2079	01-jul-2016 13:11
●	#2078	30-jun-2016 11:31
●	#2077	30-jun-2016 10:25
●	#2076	29-jun-2016 15:00
●	#2075	29-jun-2016 14:53
●	#2074	29-jun-2016 14:42
●	#2073	29-jun-2016 13:50
●	#2072	29-jun-2016 13:52

The main content area is titled 'Proyecto pe16' and includes links for 'Espacio de trabajo', 'Última Ejecución Exitosa' (pe16.jar, 1,40 MB), and 'Cambios recientes'. Below that are 'Enlaces permanentes' with links to the latest execution, stable execution, and correct execution. On the right, there are three trend charts: 'FindBugs Trend', 'PMD Trend', and 'Java Warnings Trend', each showing a count over time from #2044 to #2084.

Ilustración 26. Presentación de los resultados de un determinado proyecto

En la vista general de un proyecto, mostrada en la Ilustración 26, además de tener la información de su estado actual, se tiene acceso a un histórico de ejecuciones, a través de los cuales se pueden ver los resultados de las ejecuciones pasadas. Para tener una visión general de la evolución de un proyecto, Jenkins cuenta con gráficas que nos proporcionan esa información, con un código de colores referente a la prioridad de corrección del fallo, siendo rojo de prioridad alta, amarillo de prioridad normal y gris de prioridad baja.

Cuando se realiza un análisis, la primera acción que realiza Jenkins es la compilación del código fuente. Si este proceso no falla, se llevan a cabo los siguientes análisis, para, al finalizar, generar los informes pertinentes.

Los resultados de los análisis, en el caso de PMD, pueden mostrarse de diversas maneras en función de la forma en la que se quieran agrupar, siendo común en todas ellas el mostrar el dato de la cantidad de fallos hallados, además de presentarlos de forma visual según el código de colores anteriormente descrito.

Las distintas formas de agrupación son las siguientes:

- Por paquete (Ilustración 27)
- Por fichero
- Por categoría de la regla (Ilustración 28)
- Por regla (Ilustración 29)
- Por prioridad

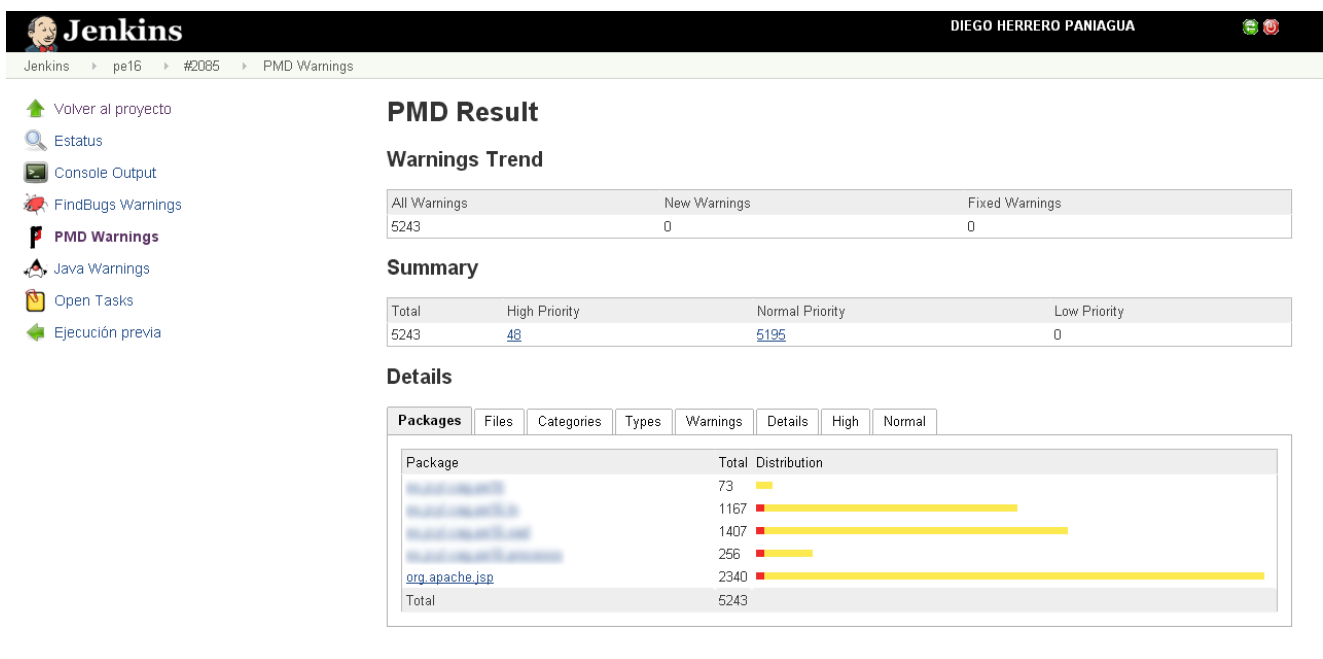


Ilustración 27. Detalle de resultado de análisis PMD agrupado por paquete

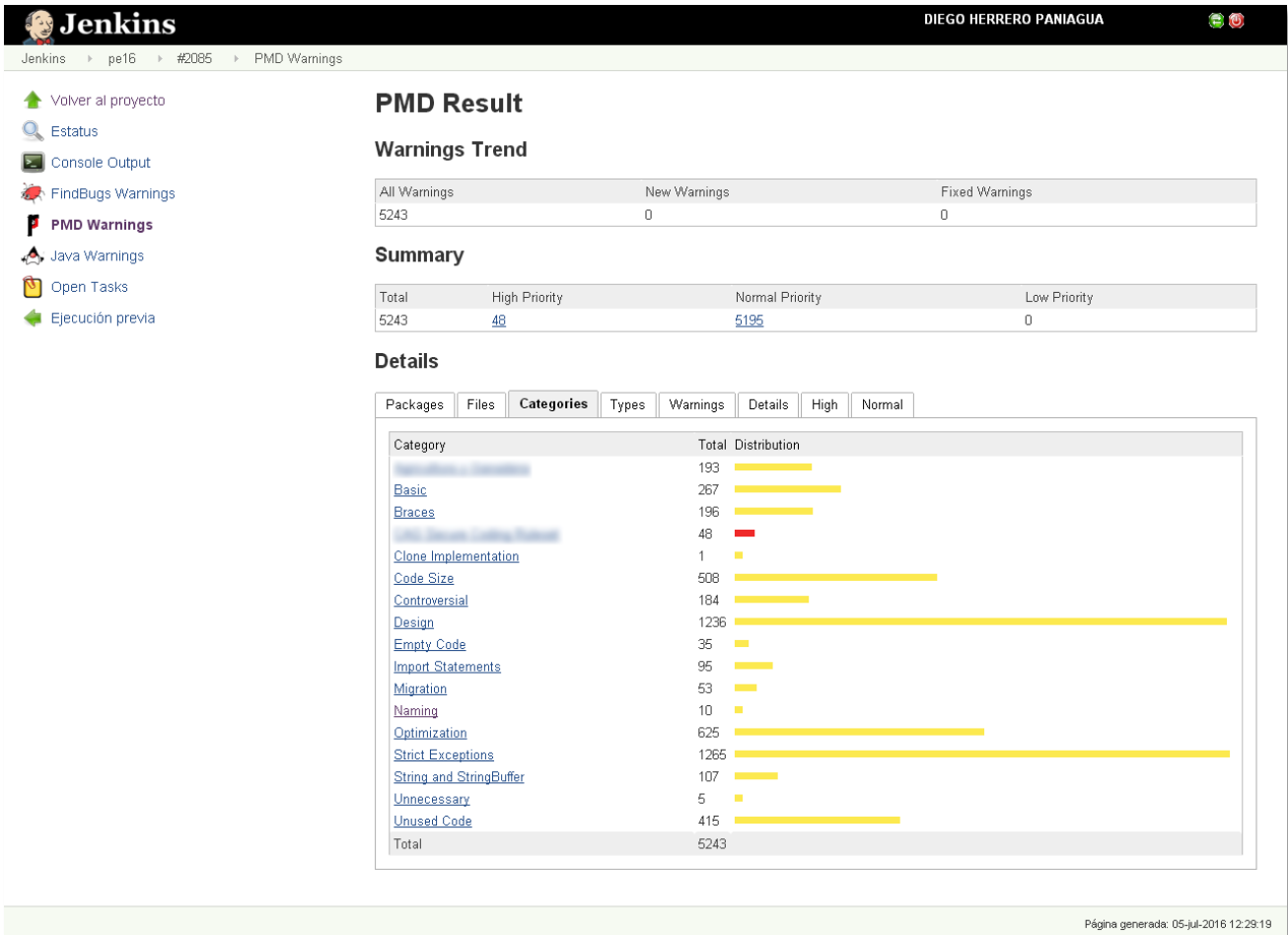


Ilustración 28. Detalle de resultado de análisis PMD agrupado por categoría

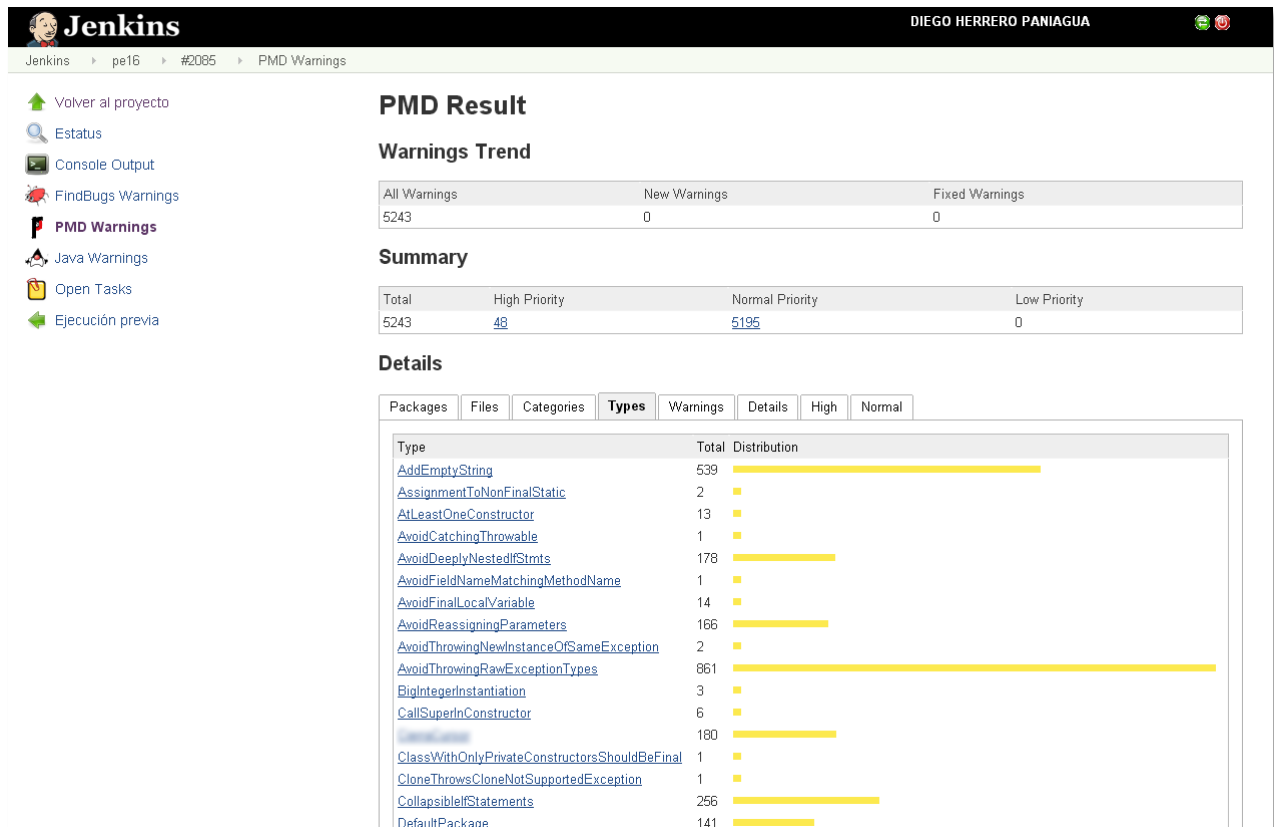


Ilustración 29. Detalle de resultado de análisis PMD agrupado por regla

Por último, cuando se accede a la información de un determinado fallo en un fichero concreto, se nos muestra la descripción detallada del fallo, así como un ejemplo genérico del mismo. Además, al seleccionar el enlace del fichero, este nos sería mostrado en la línea concreta en la que se ha detectado el fallo, para así poder solucionarlo de una forma más rápida.

The screenshot shows the Jenkins interface for PMD Warnings. The top navigation bar includes the Jenkins logo, the user name 'DIEGO HERRERO PANIAGUA', and a status indicator. The breadcrumb trail is 'Jenkins > pe16 > #2085 > PMD Warnings'. On the left sidebar, there are links for 'Volver al proyecto', 'Estatus', 'Console Output', 'FindBugs Warnings', 'PMD Warnings' (highlighted), 'Java Warnings', 'Open Tasks', and 'Ejecución previa'. The main content area is titled 'PMD Result' and contains a 'Warnings Trend' table, a 'Summary' table, and a 'Details' section. The 'Warnings Trend' table shows 5243 All Warnings, 0 New Warnings, and 0 Fixed Warnings. The 'Summary' table shows 5243 Total warnings, with 48 High Priority, 5195 Normal Priority, and 0 Low Priority. The 'Details' section has tabs for 'Packages', 'Files', 'Categories', 'Types', 'Warnings', 'Details' (selected), 'High', and 'Normal'. The selected 'Details' tab shows a warning for 'java.123, ExceptionAsFlowControl, Priority: Normal'. The warning text reads: 'Avoid using exceptions as flow control. Using Exceptions as form of flow control is not recommended as they obscure true exceptions when debugging. Either add the necessary validation or use an alternate control structure.' Below the text is a code snippet for a 'bar()' method that demonstrates the use of exceptions for flow control.

All Warnings	New Warnings	Fixed Warnings
5243	0	0

Total	High Priority	Normal Priority	Low Priority
5243	48	5195	0

Details

Packages Files Categories Types Warnings **Details** High Normal

java.123, ExceptionAsFlowControl, Priority: Normal

Avoid using exceptions as flow control.
Using Exceptions as form of flow control is not recommended as they obscure true exceptions when debugging. Either add the necessary validation or use an alternate control structure.

```
public void bar() {
    try {
        try {
        } catch (Exception e) {
            throw new WrapperException(e);
            // this is essentially a GOTO to the WrapperException catch block
        }
        } catch (WrapperException e) {
            // do some more stuff
        }
    }
}
```

Ilustración 30. Descripción del fallo detectado

CAPÍTULO V: CREACIÓN DE NUEVAS REGLAS

Hasta ahora, hemos visto qué es PMD, sus funciones y las distintas maneras de usarlo. Además, para adaptarse a las necesidades específicas de cada usuario, PMD permite la creación de nuevas reglas personalizadas, acordes a nuestras necesidades. No obstante, antes de crear una regla, es necesario conocer qué es lo que analiza concretamente PMD, esto es, su funcionamiento interno.

Funcionamiento Interno de PMD

PMD no analiza directamente el código fuente de una aplicación. En su lugar, para un fichero concreto de una clase, genera el árbol de sintaxis abstracta (*Abstract Syntax Tree*, AST). Un AST es la representación de la estructura sintáctica del código fuente, donde cada nodo representa un elemento del mismo.^{[19][20]}

La herramienta PMD cuenta con una herramienta que nos permite generar dicho árbol a partir de un código fuente dado, llamada `bgastviewer.bat`, localizada en la misma carpeta que el resto de ejecutables de PMD.

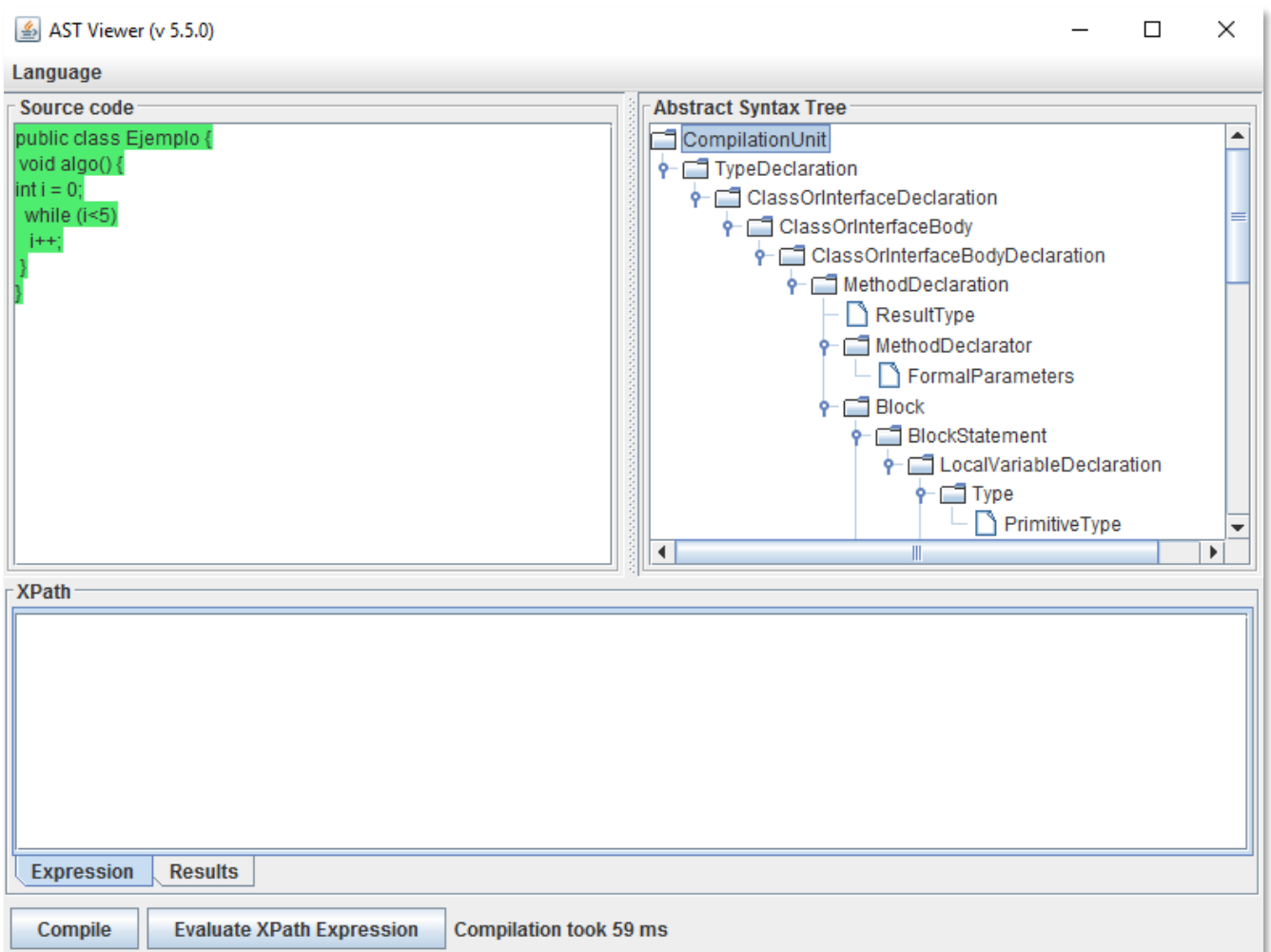


Ilustración 31. Utilidad de PMD para la generación de AST

Su uso es muy sencillo, basta con escribir nuestro código en el apartado *Source Code* y pulsar el botón *Compile*. Acto seguido, veremos en el apartado *Abstract Syntax Tree* el árbol generado, que para el código de ejemplo mostrado en la ilustración anterior es el siguiente:

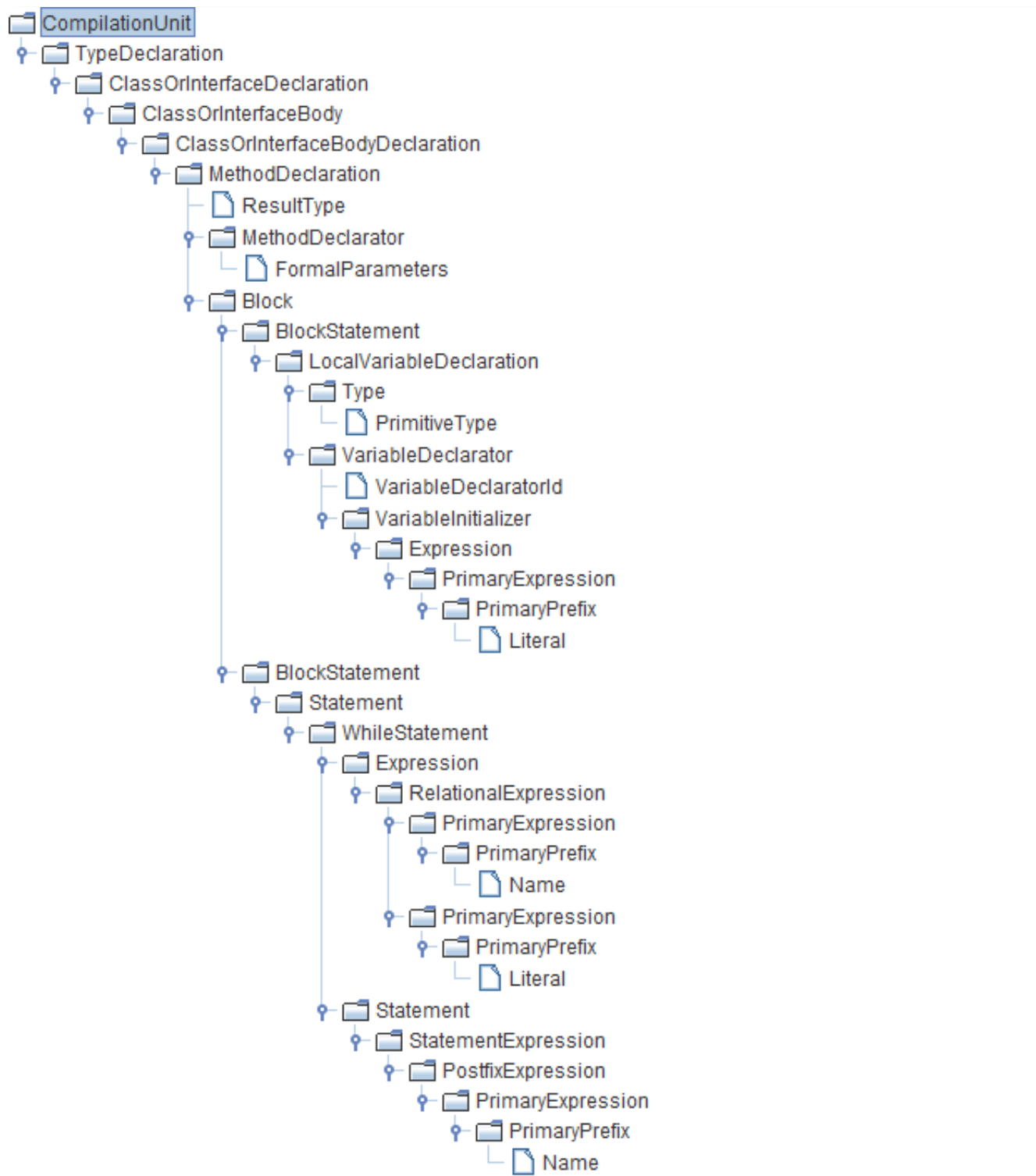


Ilustración 32. Ejemplo de AST

Para comprender mejor el significado de cada nodo, tomaremos parte del código anterior y su correspondencia en el AST.

Código	AST
<pre>while (i<5) i++;</pre>	<pre>WhileStatement Expression RelationalExpression PrimaryExpression PrimaryPrefix Name PrimaryExpression PrimaryPrefix Literal Statement StatementExpression PostfixExpression PrimaryExpression PrimaryPrefix Name</pre>

Tabla 19. Correspondencia de código de un bucle *while* con su representación en AST

Como se puede apreciar en la Tabla 19, un sencillo bucle *while* se compone de un nodo *WhileStatement* con dos hijos, la condición (*Expression*) y su acción (*Statement*). A su vez, la condición del bucle es un nodo *RelationalExpression* (esto es, una comparación), y se compone de dos *PrimaryExpression*, uno que contiene la variable *i* (*Name*) y otro que contiene el número 5 (*Literal*). Por otro lado, la acción es un nodo *StatementExpression* que contiene una operación que se aplica a la variable *i* (*Name*) tras su escritura (*PostfixExpression*).

Una de las categorías de las reglas de PMD está relacionada con el uso de las llaves de apertura y cierre, así que veamos la representación del mismo bucle, pero esta vez con dichas llaves.

Código	AST
<pre>while (i<5){ i++; }</pre>	<pre>WhileStatement Expression RelationalExpression PrimaryExpression PrimaryPrefix Name PrimaryExpression PrimaryPrefix Literal Statement Block BlockStatement StatementExpression PostfixExpression PrimaryExpression PrimaryPrefix Name</pre>

Tabla 20. Correspondencia de código de un bucle *while* con llaves con su representación en AST

La única diferencia con la representación anterior es la aparición de dos nuevos nodos, *Block* y *BlockStatement*. Por tanto, una regla que busque bucles *while* sin llaves de apertura y cierre debe fijarse en la ausencia de estos dos nodos.

Para abordar la creación de esta y cualquier otra regla, PMD ofrece dos alternativas:[17][18]

- El uso de expresiones regulares para detectar la condición de la regla, es decir, utilizar XPath.[21]
- Desarrollar una clase Java cuya función sea, al detectar un nodo de un tipo determinado, evaluar si cumple o no la condición de la regla.

En el ejemplo que estamos tratando, encontrar bucles `while` sin llaves, definir una regla XPath es casi inmediato, ya que debemos encontrar cualquier nodo padre del tipo `WhileStatement` que no tenga nodos hijo del tipo `Statement/Block`.

```
//WhileStatement[not(Statement/Block)]
```

Con `//WhileStatement` obtenemos todos los nodos de ese tipo. En el interior de los corchetes se establece qué tipo de nodos hijo estamos buscando, `Statement/Block`, y precisamente estamos buscando su ausencia, lo conseguimos con el `not`.

Cabe destacar que con la herramienta `bgastviewer.bat` es posible desarrollar y probar las reglas escritas en XPath. Para ello, basta con escribir la regla en la pestaña `Expression` del apartado `XPath` y pulsar el botón `Evaluate XPath Expression`. Tras ello, en la pestaña `Results` se mostrará el resultado del análisis, en el que, si seleccionamos una de las expresiones listadas, se marcará el fragmento de código afectado por el incumplimiento de la regla. Esto se muestra en la Ilustración 33.

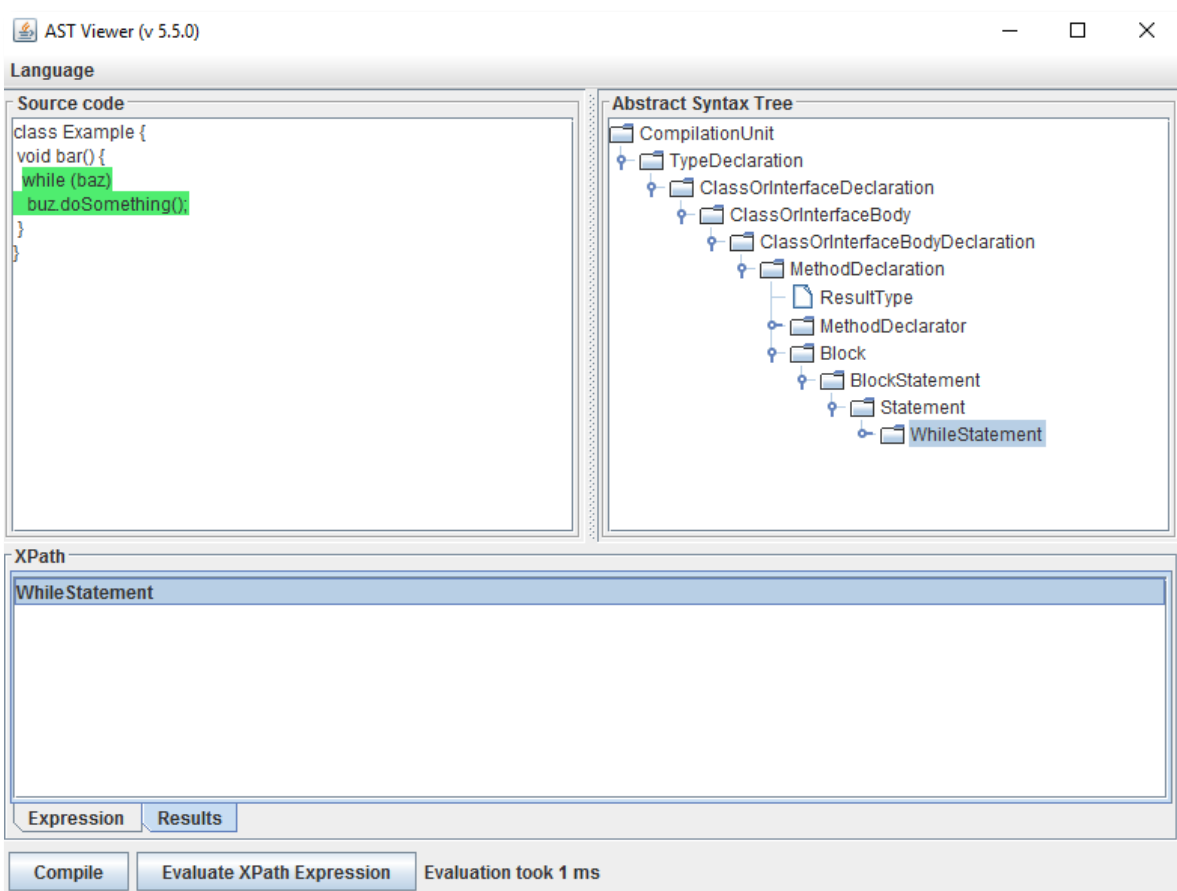


Ilustración 33. Evaluación de reglas en la herramienta `bgastviewer.bat`

En Java, la definición de esta misma regla es bastante más compleja. Cada clase que define una regla dispone de un método `visit()` que se ejecuta cada vez que se encuentra un nodo del tipo definido.

```
public Object visit(ASTWhileStatement node, Object data)
```

A partir de ese nodo, se extrae el segundo nodo hijo, que contiene el nodo `Statement`, dentro del cual se debería encontrar un nodo `Block`.

```
Node firstStmt = node.jjtGetChild(1)
```

Una vez conseguido el nodo `Statement`, se comprueba si tiene nodos hijos, y si los tiene, se verifica si el primer hijo es un nodo `Block`.

```
node.jjtGetNumChildren() != 0 && (node.jjtGetChild(0) instanceof ASTBlock)
```

En caso negativo, la regla habrá detectado un fallo y lo agregará al informe final. La implementación completa es la siguiente:

```
public class WhileLoopsMustUseBracesRule extends AbstractJavaRule {
    public Object visit(ASTWhileStatement node, Object data) {
        Node firstStmt = node.jjtGetChild(1);
        if (!hasBlockAsFirstChild(firstStmt)) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
    private boolean hasBlockAsFirstChild(Node node) {
        return (node.jjtGetNumChildren() != 0 && (node.jjtGetChild(0)
instanceof ASTBlock));
    }
}
```

Uso de Reglas Personalizadas

Una vez hemos conseguido crear reglas adaptadas a nuestras necesidades, el siguiente paso es utilizarlas. La forma de realizarlo es integrarlas en un conjunto de reglas o `ruleset`, un fichero `xml` en el que se declaran todas las reglas que queremos que se ejecuten en un mismo análisis. Además, hay que aportar el fichero `.class` de las reglas que hayamos definido en lenguaje Java. Tanto el `ruleset` como los `.class` se comprimirán en un fichero `.jar`, esto es, una librería, que se agregará a las ya existentes en el directorio `lib` del código fuente de PMD.

Si queremos ejecutar el análisis, no tenemos que realizar ninguna especificación a mayores. El comando `pmd.bat` toma todas las librerías de su directorio `lib`, por lo que solo hay que pasar como parámetro nuestro `ruleset` personalizado en la opción `-R` del comando.

La estructura del fichero `xml` es la siguiente:

```
<?xml version="1.0"?>
<ruleset name="My custom rules"
  xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0
http://pmd.sourceforge.net/ruleset_2_0_0.xsd">
  <rule name="WhileLoopsMustUseBracesRule"
    message="Avoid using 'while' statements without curly braces"
    class="WhileLoopsMustUseBracesRule">
    <description>
      Avoid using 'while' statements without using curly braces
    </description>
    <priority>3</priority>
    <example>
      <![CDATA[
        public void doSomething() {
          while (true)
            x++;
        }
      ]]>
    </example>
  </rule>
</ruleset>
```


Como se puede ver, un `ruleset` se compone de una cabecera en la que se define el esquema a seguir por el fichero `xml`, y la definición de cada una de las reglas. Dicha definición se compone de:

- `rule name`, el nombre de la regla.
- `message`, el mensaje a mostrar cuando una regla es violada.
- `class`, la clase en la que está contenida la regla en caso de estar escrita en Java. En caso de que la regla esté confeccionada mediante una expresión XPath, el campo `class` debe contener la definición del uso de dichas reglas, esto es, `net.sourceforge.pmd.lang.rule.XPathRule`.
- `description`, una descripción más detallada que se muestra en algunos casos, como al mostrar el detalle de una vulnerabilidad detectada en Jenkins.
- `priority`, la gravedad de la vulnerabilidad.
- `example`, un ejemplo del código que hace saltar la regla, también se muestra en el detalle de la vulnerabilidad.

Para el caso de una regla XPath, además, podemos encontrar un campo más

```
<properties>
  <property name="xpath">
    <value>
      <![CDATA[
        //EqualityExpression/PrimaryExpression
        /PrimaryPrefix/Literal/BooleanLiteral
      ]]>
    </value>
  </property>
</properties>
```

Como una expresión XPath no tiene una clase que contenga la regla, se define directamente en el `ruleset`.

Reglas XPath

Definición de objetivos

Se desea estudiar la posibilidad de crear una regla XPath para detectar estructuras de tipo `switch` con el mismo número de `cases` a lo largo del código fuente de un fichero.

Desarrollo

Antes de empezar, es necesario conocer el árbol AST de una estructura `switch`. Para mejorar la comprensión del árbol, solo se mostrará detallada la estructura de un `case`, ya que es la misma tanto para cada `case` como para el `default`.

Código	AST
<pre> switch(i) { case 1: i+=1; break; case 2: i+=2; break; default: i+=1; break; } </pre>	<pre> SwitchStatement Expression PrimaryExpression PrimaryPrefix Name SwitchLabel Expression PrimaryExpression PrimaryPrefix Literal BlockStatement Statement StatementExpression PrimaryExpression PrimaryPrefix Name AssignmentOperator Expression PrimaryExpression PrimaryPrefix Literal BlockStatement Statement BreakStatement SwitchLabel BlockStatement BlockStatement SwitchLabel BlockStatement BlockStatement </pre>

Tabla 21. Representación AST de una estructura `switch`

En la representación AST que se puede ver en la Tabla 21 hay dos nodos representativos de una estructura `switch`, que son los que tendremos que tener en cuenta para crear la regla:

- `SwitchStatement`, es el nodo padre de una estructura `switch`, por tanto, hay que buscar en el código nodos de este tipo.

```
//SwitchStatement
```

- `SwitchLabel`, son los nodos hijos que representan tanto las sentencias de tipo `case` como la sentencia `default`. Nos interesa pues, saber el número de nodos de este tipo presentes en la estructura `switch`.

```
count(SwitchLabel)
```

La limitación que nos encontramos en las expresiones XPath es la imposibilidad de comparar las características de un nodo con las de otro. Por tanto, la creación de una regla XPath que busque estructuras de tipo switch con el mismo número de casos es inviable.

Actualmente, y mediante expresiones XPath, PMD tiene predefinidas una serie de reglas para analizar estructuras de tipo switch, por ejemplo, relativas a la ausencia de sentencias cases o default o al escaso número de casos que puedan tener.

Por último, como ejemplo de regla XPath de análisis de estructuras de tipo switch, se ha decidido crear una regla que busque estructuras de este tipo con un número excesivo de sentencias cases. Un ejemplo muy común de switch con muchos casos es, en función de un entero que represente el número de un mes del año, conseguir el nombre de dicho mes. En este caso, habrá doce casos para cada mes del año y un default para contemplar el caso de un número no correspondiente a un mes, 13 SwitchLabel en total. Por lo tanto, la regla creada se encargará de buscar estructuras de tipo switch que tengan más de 13 sentencias de tipo case junto con su sentencia default.

```
//SwitchStatement[count(SwitchLabel)>13]
```

Reglas Java

Definición de objetivos

Se desea crear una nueva regla que detecte clases con un número elevado de constantes, para sugerir su sustitución por un tipo enumerado.

Desarrollo

Para comenzar, es necesario conocer el árbol AST de la definición de una constante, esto es, una variable con modificadores `final` y `static`, en una clase. Para mejorar la comprensión del árbol, solo se mostrará detallada la representación de una de las constantes, ya que es la misma en todos los casos.

Código	AST
<pre>public class prueba{ final static String a1 = "dsadsa"; final static String a2 = "asdassadas"; final static String a3 = "asdas"; final static String a4 = "asdasdasdas"; }</pre>	<pre>ClassOrInterfaceBody ClassOrInterfaceBodyDeclaration FieldDeclaration Type ReferenceType ClassOrInterfaceType VariableDeclarator VariableDeclaratorId VariableInitializer Expression PrimaryExpression PrimaryPrefix Literal ClassOrInterfaceBodyDeclaration ClassOrInterfaceBodyDeclaration ClassOrInterfaceBodyDeclaration</pre>

Tabla 22. Representación AST de la definición de constantes en una clase

Para definir la regla Java, hay que decidir primero cuál es el nodo que queremos que se visite para analizar. Como lo que buscamos es una clase que tenga muchas constantes definidas, lo que se buscará es un nodo del tipo `ASTClassOrInterfaceBody`.

El siguiente paso es saber cuántos nodos hijo tiene el nodo padre `ASTClassOrInterfaceBody`. Para ello, dicha clase posee un método con el que podemos obtener ese dato.

```
int totalHijos = node.jjtGetNumChildren();
```

El nodo del que nos interesa sacar información es del tipo `FieldDeclaration`, por lo que hay que saber si los nodos hijos tienen a su vez nodos hijos y, como primer hijo, uno de dicho tipo.

```
Node child=node.jjtGetChild(contador);

(child.jjtGetNumChildren() != 0 && (child.jjtGetChild(0) instanceof
ASTFieldDeclaration));
```

Si se cumple la condición, se extrae el nodo. La clase `ASTFieldDeclaration` cuenta con los métodos `isStatic()` e `isFinal()` para determinar si dicho nodo cuenta con esos modificadores. En caso afirmativo, se incrementará un contador.

```
ASTFieldDeclaration child2 = (ASTFieldDeclaration) child.jjtGetChild(0);

(child2.isStatic() && child2.isFinal());

contadorConstantes++;
```

Por último, se comprueba el valor final de ese contador. Se ha determinado que a partir de dos constantes definidas en una clase sería recomendable usar un tipo enumerado. Por tanto, en caso de que el número de nodos del tipo `FieldDeclaration` sea mayor que dos, se añade una violación al total detectado.

```
if (contadorConstantes > 2) {  
    addViolation(data, node);  
}
```


CAPÍTULO VI: CONCLUSIONES

Conclusiones

Tras finalizar este proyecto, se puede concluir que se han abordado todos los temas referentes a la herramienta PMD, desde sus funcionalidades hasta la creación de nuevas reglas, pasando además por todas las funcionalidades que ofrece.

Además, se han conocido otras herramientas, como son Checkstyle y Findbugs, que realizan funciones similares, y que, un uso combinado de todas ellas mejora en gran medida el proceso de desarrollo de código.

A nivel personal, este proyecto me ha permitido conocer a fondo las bases y el funcionamiento de la herramienta con la que estuve lidiando en la asignatura Prácticas de Empresa. Espero, además, que esta memoria pueda ser tomada como una guía para que cualquier persona que utilice PMD en su entorno pueda entender, al menos de manera básica, cómo funciona. Además, al crear nuevas reglas en lenguaje XPath, he reforzado y refrescado el conocimiento sobre el mismo, ya que, en la actualidad, mi actividad se centra en el uso de Java, JavaScript y SQL.

Por otro lado, el conocimiento adquirido me será de utilidad en el desempeño de mi actividad laboral, para así desarrollar código limpio y de calidad.

Trabajo futuro

Un proyecto de estas características siempre puede ser ampliado. Algunas de las posibles vías de trabajo futuro comprenden desde el estudio comparativo con otras herramientas, como las ya citadas Checkstyle y Findbugs, hasta su uso en otras plataformas, entornos de desarrollo y sistemas operativos.

Sin embargo, lo que tiene más potencial de ser estudiado, analizado y desarrollado en el futuro, es la creación de nuevas reglas. Siempre pueden surgir necesidades específicas para un determinado usuario que no estén recogidas ya en PMD.

CAPÍTULO VII: BIBLIOGRAFÍA

- [1] PMD (software)", *Wikipedia*, 2016. [Online]. Disponible: [https://en.wikipedia.org/wiki/PMD_\(software\)](https://en.wikipedia.org/wiki/PMD_(software)). [Último acceso: 25 – May - 2016].
- [2] PMD – Old Changelog", *Pmd.sourceforge.net*, 2016. [Online]. Disponible: <http://pmd.sourceforge.net/snapshot/overview/changelog-old.html>. [Último acceso: 25 – May - 2016].
- [3] "PMD – Future directions", *Pmd.github.io*, 2016. [Online]. Disponible: <http://pmd.github.io/pmd-5.5.0/overview/future.html>. [Último acceso: 25 – May - 2016].
- [4] "GNU Lesser General Public License", *Wikipedia*, 2016. [Online]. Disponible: https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License. [Último acceso: 25 – May - 2016].
- [5] "Checkstyle", *Wikipedia*, 2016. [Online]. Disponible: <https://en.wikipedia.org/wiki/Checkstyle>. [Último acceso: 27 – May - 2016].
- [6] "checkstyle – Checkstyle 7.0", *Checkstyle.sourceforge.net*, 2016. [Online]. Disponible: <http://checkstyle.sourceforge.net/>. [Último acceso: 27 – May - 2016].
- [7] "FindBugs™ - Find Bugs in Java Programs", *Findbugs.sourceforge.net*, 2016. [Online]. Disponible: <http://findbugs.sourceforge.net/>. [Último acceso: 27 – May - 2016].
- [8] "FindBugs, Part 1: Improve the quality of your code", *Ibm.com*, 2016. [Online]. Disponible: <http://www.ibm.com/developerworks/java/library/j-findbug1/>. [Último acceso: 27 – May - 2016].
- [9] "Comparison of Static Code Analysis Tools for Java - Findbugs vs PMD vs Checkstyle - Software Engineering Candies", *Sw-engineering-candies.com*, 2016. [Online]. Disponible: <http://www.sw-engineering-candies.com/blog-1/comparison-of-findbugs-pmd-and-checkstyle>. [Último acceso: 28 – May - 2016].
- [10] I. Sommerville and M. Alfonso Galipienso, *Ingeniería del software*. Madrid: Pearson Addison-Wesley, 2005.
- [11] "Jakarta Project", *Es.wikipedia.org*, 2016. [Online]. Disponible: https://es.wikipedia.org/wiki/Jakarta_Project. [Último acceso: 04 – Jun - 2016].
- [12] "Secure Coding Guidelines for Java SE", *Oracle.com*, 2016. [Online]. Disponible: <http://www.oracle.com/technetwork/java/seccodeguide-139067.html#gcg>. [Último acceso: 04 – Jun - 2016].

- [13] "pmd-eclipse-plugin", 2016. [Online]. Disponible: <https://marketplace.eclipse.org/content/pmd-eclipse-plugin>.
[Último acceso: 05 - Jun - 2016].
- [14] "EclipseZone - Improving Code Quality with PMD and Eclipse", *Eclipsezone.com*, 2016. [Online]. Disponible: <http://www.eclipsezone.com/articles/pmd/>.
[Último acceso: 06 - Jun - 2016].
- [15] "Jenkins", *Es.wikipedia.org*, 2016. [Online]. Disponible: <https://es.wikipedia.org/wiki/Jenkins>.
[Último acceso: 05 - Jun - 2016].
- [16] "Jenkins", *Jenkins.io*, 2016. [Online]. Disponible: <https://jenkins.io/>.
[Último acceso: 05 - Jun - 2016].
- [17] "PMD - How to write a PMD rule", *Pmd.github.io*, 2016. [Online]. Disponible: <http://pmd.github.io/pmd-5.5.0/customizing/howtowritearule.html>.
[Último acceso: 20 - Jun - 2016].
- [18] T. Copeland, "PMD - How to write a PMD rule", *Pmd.sourceforge.net*, 2016. [Online]. Disponible: <http://pmd.sourceforge.net/pmd-4.3.0/howtowritearule.html>.
[Último acceso: 15 - Jun - 2016].
- [19] "Abstract syntax tree", *Wikipedia*, 2016. [Online]. Disponible: https://en.wikipedia.org/wiki/Abstract_syntax_tree.
[Último acceso: 12 - Jun - 2016].
- [20] I. Neamtiu, J. Foster and M. Hicks, "Understanding source code evolution using abstract syntax tree matching", *Proceedings of the 2005 international workshop on Mining software repositories - MSR '05*, 2005.
- [21] "XPath Expression Syntax", *Saxon.sourceforge.net*, 2016. [Online]. Disponible: <http://saxon.sourceforge.net/saxon6.5.3/expressions.html>.
[Último acceso: 15 - Jun - 2016].

CAPÍTULO VIII: ANEXOS

Código fuente de la clase Java ALotOfFinalStaticVariables

```
package net.sourceforge.pmd.lang.java.rule.custom;

import net.sourceforge.pmd.lang.ast.*;
import net.sourceforge.pmd.lang.java.ast.*;
import net.sourceforge.pmd.lang.java.rule.*;

public class ALotOfFinalStaticVariables extends AbstractJavaRule {
    public Object visit(ASTClassOrInterfaceBody node, Object data) {
        int totalHijos = node.jjtGetNumChildren();
        int contador = 0;
        int contadorConstantes=0;
        while (contador<totalHijos){
            Node child=node.jjtGetChild(contador);
            if(hasFieldDeclarationAsFirstChild(child)){
                ASTFieldDeclaration child2 = (ASTFieldDeclaration)
child.jjtGetChild(0);
                if(childIsFinalStatic(child2)){
                    contadorConstantes++;
                }
            }
            contador++;
        }
        if(contadorConstantes>2){
            addViolation(data, node);
        }
        return super.visit(node,data);
    }

    private boolean hasFieldDeclarationAsFirstChild(Node node) {
        return (node.jjtGetNumChildren() != 0 && (node.jjtGetChild(0)
instanceof ASTFieldDeclaration));
    }

    private boolean childIsFinalStatic(ASTFieldDeclaration node) {
        return (node.isStatic() && node.isFinal());
    }
}
```

Estructura de la librería custom.jar

- META-INF (autogenerada al crear el jar)
 - MANIFEST.MF
- net (carpeta que contiene las clases java, se ha mantenido la estructura de directorios del resto de reglas de PMD)
 - sourceforge/pmd/lang/java/rule/custom/ALotOfFinalStaticVariables.class
- rulesets (carpeta que contiene el fichero custom-ruleset.xml, se ha mantenido la estructura de directorios del resto de reglas de PMD)
 - java/custom-ruleset.xml

Manual de Pruebas

Para poder ejecutar un análisis con las reglas creadas, simplemente hay que acceder a la carpeta `pmd-bin-5.5.0/bin` y en ella abrir una consola de comandos para ejecutar el siguiente comando:

```
pmd.bat -d "..\..\Ficheros de Prueba" -f textcolor -R rulesets/java/custom-ruleset.xml
```

En caso de ejecutarlo en un entorno Unix/Linux, basta con cambiar el comando `pmd.bat` por `run.sh pmd`. La cadena de parámetros no sufriría cambios.

Se puede cambiar el parámetro `textcolor` por cualquiera de los siguientes:

- text
- csv
- html
- xml
- yahtml

Para poder ejecutar PMD, es necesario tener Java instalado, se puede descargar de la siguiente URL:

<https://www.java.com/es/download/>

Contenido del CD

El CD entregado cuenta con los siguientes directorios:

- Memoria: contiene esta memoria en formato PDF.
- Ficheros de prueba. Contiene una serie de ficheros escritos en lenguaje Java que cumplen o no las reglas creadas.
- Código Fuente. Contiene todo el código fuente desarrollado, así como una copia de la librería custom.jar, ya incluida en la carpeta de librerías del código fuente de PMD.
- pmd-bin-5.5.0. Contiene el código fuente propio de la herramienta PMD, necesario para ejecutar los comandos descritos en esta memoria.





 Código Fuente	09/07/2016 17:22	Carpeta de archivos
 Ficheros de Prueba	09/07/2016 17:13	Carpeta de archivos
 Memoria	09/07/2016 14:05	Carpeta de archivos
 pmd-bin-5.5.0	09/07/2016 17:13	Carpeta de archivos

Ilustración 34. Contenido del CD