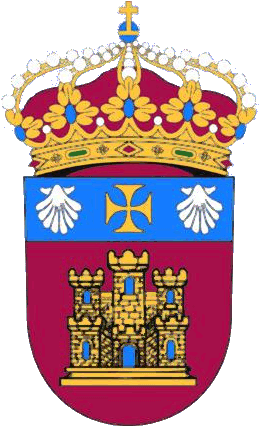
Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática



Sistemas Informáticos

Plugin de Refactorización 3.0

Anexo 4. Documentación Técnica

**Alumnos:**

Míryam Gómez San Martín

Íñigo Mediavilla Saiz

**Tutor:**

Raúl Marticorena Sánchez

**Burgos, Mayo de 2011**

# Índice de contenido

[Índice de contenido 2](#_Toc286658266)

[Índice de ilustraciones 3](#_Toc286658267)

[Índice de tablas 3](#_Toc286658268)

[Índice de fragmentos de código 3](#_Toc286658269)

[1. Lista de cambios 4](#_Toc286658270)

[1.1. Cambios 4](#_Toc286658271)

[2. Introducción 5](#_Toc286658272)

[2.1. Introduccion a la documentación técnica 5](#_Toc286658273)

[3. Documentación de las bibliotecas 6](#_Toc286658274)

[4. Código fuente 7](#_Toc286658275)

[5. Manual del programador 8](#_Toc286658276)

[5.1. Paso 1 – Limpiar 8](#_Toc286658277)

[5.2. paso 2 – Compilar 8](#_Toc286658278)

[5.3. Paso 3 – Generar la documentación 9](#_Toc286658279)

[5.4. Paso 4 – Empaquetado 9](#_Toc286658280)

[5.5. Paso 5 – Ejecutar los Tests de integración 9](#_Toc286658281)

[5.6. PASO 6 – Instalación 10](#_Toc286658282)

[5.7. Ciclo completo 10](#_Toc286658283)

[5.8. Artefactos generados tras la construcción del proyecto 13](#_Toc286658284)

[5.8.1. Artefactos generados en el directorio del plugin 13](#_Toc286658285)

[5.8.2. Artefactos generados el directorio de tests 14](#_Toc286658286)

[5.8.3. Generación del repositorio de instalación del plugin 14](#_Toc286658287)

[5.8.4. Instalación del plugin en el repositorio local de Maven 15](#_Toc286658288)

[6. Pruebas unitarias 16](#_Toc286658289)

[7. Bibliografía 17](#_Toc286658290)

## Índice de ilustraciones

[Ilustración 1. Portada de las métricas de un proyecto con Sonar 11](#_Toc286658291)

[Ilustración 2. Página de HotSpots 12](#_Toc286658292)

[Ilustración 3. Pantalla Time Machine en Sonar 13](#_Toc286658293)

## Índice de tablas

**No table of figures entries found.**

## Índice de fragmentos de código

[Código 1. Fichero xml de configuración de la “targetplatform”. 9](file:///C:\Users\Mediavillas\Documents\Memoria-PFC\docs\Anexo4-DocumentacionTecnica.docx#_Toc286658294)

# Lista de cambios

## Cambios

|  |  |  |  |
| --- | --- | --- | --- |
| Número | Fecha | Descripción | Autor/es |
| 0 | 8/02/11 | Creada sección del manual del programador. | Míryam Gómez San Martín  Íñigo Mediavilla Saiz |
| 1 | 22/02/2011 | Correcciones de formato. Explicación de instalación de Maven. | Míryam Gómez San Martín  Íñigo Mediavilla Saiz |
| 2 | 28/02/2011 | Añadidos prerrequisitos para la construcción y lista de artefactos generados por el proceso. | Míryam Gómez San Martín  Íñigo Mediavilla Saiz |

# Introducción

## Introduccion a la documentación técnica

# Documentación de las bibliotecas

# Código fuente

# Manual del programador

La construcción del producto se lleva a cabo mediante la herramienta Apache Maven ya descrita en el apartado de “Técnicas y herramientas” de la memoria. A continuación se hará una referencia a los prerrequisitos necesarios para llevar a efectuar la construcción para después listar los comandos necesarios para construir el software y las funciones de cada uno:

## Prerequisito - Instalación Maven 3

Para instalar Maven 3 es necesario disponer de un Java Runtime Environment (JRE) instalado en el equipo. La variable JAVA\_HOME debe estar correctamente configurada apuntando al directorio en el que el JRE está instalado y JAVA\_HOME/bin debe pertenecer a la variable “Path” del sistema para hacer posible la ejecución de Java.

### Pasos comunes

Ir a la página de downloads de maven [ref: <http://maven.apache.org/download.html>] y descargarse cualquiera de los paquetes binarios de entre los distintos formatos de ficheros comprimidos que aparecen.

Descomprimir el fichero descargado en el directorio deseado, a partir de ahora nos referiremos a dicho directorio como “M2\_HOME” que es el nombre de la variable que se utiliza para referenciar dicho directorio y que tendremos que crear a continuación. Ahora los pasos se van a explicar para la instalación de Maven3 en sistemas operativos UNIX y en Windows basados en los propios pasos explicados en la web de Maven [ref: http://maven.apache.org/download.html]:

### Instalación en Windows 2000/XP

* Añadir la variable “M2\_HOME” a las propiedades del sistema. Para acceder a la configuración de las variables de entorno del sistema se puede proceder pulsando la tecla de Windows + Pausa, seleccionando la pestaña “Avanzadas” y el botón “Variables de entorno”. Nota: Se debe omitir cualquier comilla incluso si la ruta contiene espacios.
* En el mismo diálogo se debe añadir al variable de entorno “M2” con el valor “%M2\_HOME%\bin”.
* Actualizar (o crear si no existe) la variable de entorno “Path” dentro de las variables de usuario y añadir el valor “%M2%” para hacer el comando “mvn” accesible desde la línea de comandos.
* Abrir una ventana de comandos pulsando “Tecla de Windows + R” y luego tecleando “cmd”. A continuación ejecutar “mvn –version” para verificar que “Maven” está correctamente instalado.

## Sistemas operativos basados en Unix (Linux, Solaris and Mac OS X)

* Desde una terminal de comandos agregar la variable de entorno “M2\_HOME”. Ejemplo:

export M2\_HOME=/usr/local/apache-maven/apache-maven-3.0.2

* Agregar la variable de entorno “M2”:

export M2=$M2\_HOME/bin

* Añadir la variable M2 al Path:

export PATH=$M2:$PATH

* Ejecutar “mvn –version” para verificar que “Maven” está correctamente instalado.

## Paso 1 – Limpiar

mvn clean

Borra los productos generados para poder construir el software desde cero. Se queda exclusivamente con lo necesario para construir el producto borrando los restos de procesos de construcción previos.

## paso 2 – Compilar

mvn compile

Si ha habido cambios en el código fuente (si no los ha habido la compilación sería innecesaria) compila el código fuente de todos los paquetes. Este paso es especial para un proyecto como el nuestro de un plugin de Eclipse, porque el plugin a construir no sólo tiene dependencias en bibliotecas disponibles en ficheros JAR en el propio proyecto, sino que también tiene dependencias en otros paquetes OSGI (que son un tipo especial de fichero JAR. Ver (OSGI)). Para poder compilar las clases es necesario recoger esas dependencias de algún lado. El lugar del que esas dependencias se recogen es conocido como “target platform” (plataforma objetivo) que es el almacen de plugins sobre la que ese plugin necesita correr para poder encontrar sus dependencias. La plataforma objetivo se puede definir apuntando a un directorio local (habitualmente una instalación de Eclipse) o gracias al plugin para Maven conocido como tycho como un repositorio de internet desde el que se pueden descargar los paquetes. En el fichero de configuración del plugin la mayoría de los paquetes los descargamos del sitio (EUpdates). Esta configuración está definida dentro de la carpeta dynamicrefactoring.targetplatform en el fichero dynamicrefactoring.target que se puede editar desde la interfaz gráfica con Eclipse pero cuyo contenido se muestra a continuación en formato de texto plano:

Código 1. Fichero xml de configuración de la “targetplatform”.

**<?xml version**="1.0"**encoding**="UTF-8"**standalone**="no"?>  
<?**pde version**="3.6"?>  
  
<target **name**="dynamicrefactoring">  
<locations>  
<location **includeAllPlatforms**="false" **includeMode**="slicer" **type**="InstallableUnit">  
<unit **id**="org.eclipse.sdk.ide" **version**="3.6.1.M20100909-0800"/>  
<repository **location**="<http://download.eclipse.org/eclipse/updates/3.6>"/>  
</location>  
</locations>  
</target>

Esto nos permite hacer el proceso de construcción totalmente independiente del equipo en el que se ejecuta. Cualquier equipo con conexión a Internet sólo necesita tener Maven 3 instalado y ejecutar la fase de compilación para compilar el proyecto. El propio Maven se ocupa de descargar las dependencias la primera vez que se ejecuta y almacenarlas en el repositorio local para luego ser utilizadas para compilar y ejecutar de las pruebas.

## Paso 3 – Generar la documentación

mvn javadoc:javadoc

Genera la documentación del API del proyecto sobre la carpeta target/javadoc de cada uno de los subproyectos de que el plugin se compone: el subproyecto del plugin y el de los tests. Además el fichero pom.xml está configurado para ejecutar el doccheck que genera un informe en el que identifica las clases o métodos sin comentarios y otras omisiones o irregularidades en la documentación de las clases.

## Paso 4 – Empaquetado

mvn package

Toma el código fuente compilado y lo empaqueta en el formato de fichero distribuible más adecuado. Para nuestro proyecto será el formato JAR. En otros tipos de proyectos este formato cambiará. Por ejemplo, para proyectos de páginas web dinámicas el formato por defecto es el de WAR.

## Paso 5 – Ejecutar los Tests de integración

mvn integration-tests

Es una fase posterior al compilado y empaquetado por tanto ejecuta ambas previamente a ejecutarse a sí misma. Por sí sola se encarga de ejecutar los tests de integración, es decir, los incluidos en el fragmento de pruebas del plugin. Este comando genera un entorno OSGI con los paquetes de Eclipse y desde este entorno se ejecutan las pruebas definidas en el fragmento. De este modo se permite hacer las pruebas en un entorno igual al de producción tal y como indicaba uno de los principios de la integración continua [ref apartado integr. Continua]. Si alguno de los tests falla provoca el fin del proceso de construcción para asegurar que fases posteriores como podría ser el despliegue no se llevan a cabo.

## PASO 6 – Instalación

mvn install

Ejecuta todos los pasos anteriormente descritos excepto el de javadoc y además copia el paquete al repositorio local de Maven.

## Ciclo completo

mvn clean install sonar:sonar

Comando que sirve para ejecutar el proceso de construcción completo. Las dos primeras fases ya han sido explicadas. El último objetivo se encarga de generar mediante (Sonar) una página web dinámica en forma de informe en el que se agrupan comprobaciones estáticas de “indicadores de errores” en el código generadas con (PMD), (CheckStyle) y (FindBugs), métricas de complejidad del código, informes de la cobertura de las pruebas, número de tests ejecutados, número de líneas de código del proyecto, estadísticas de la documentación y un pequeño grafo resumen con una estimación orientativa de la mantenibilidad del proyecto.

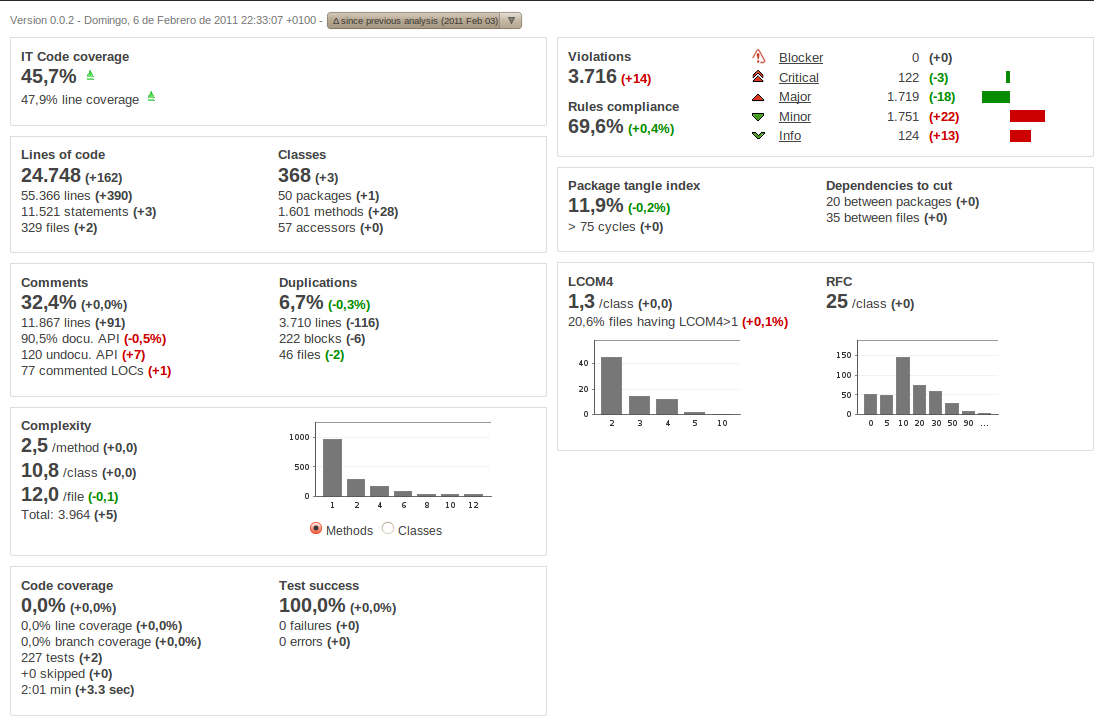


Ilustración 1. Portada de las métricas de un proyecto con Sonar

Además genera una página con el nombre de “Hotspots” (puntos calientes) en la que se muestra una lista de los defectos más importantes que presenta el proyecto agrupados por categorías tales como: lista de reglas más violadas, tests que tardan más en ejecutarse, clases más complejas, etc.

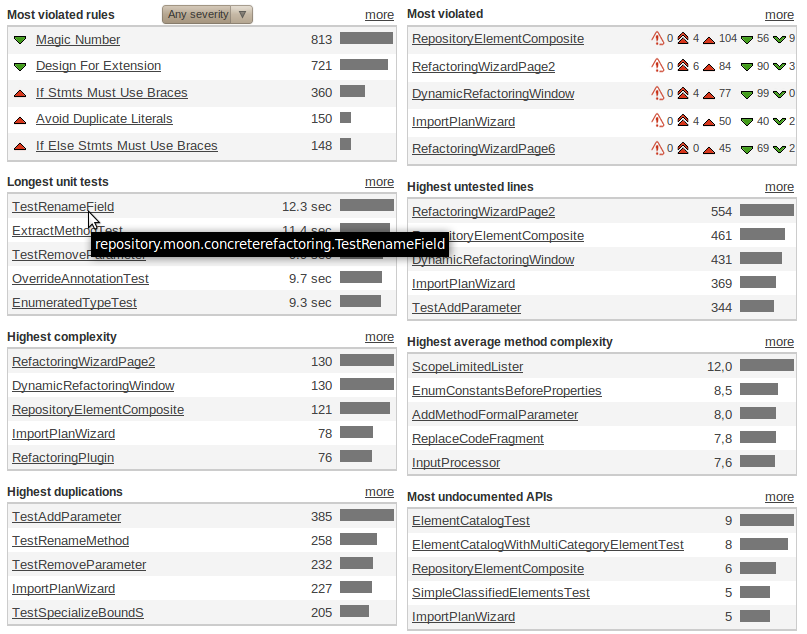


Ilustración 2. Página de HotSpots

También permite hacer retrospectivas de la evolución de las métricas en el proyecto tanto en la página principal como en el apartado “Time machine” (máquina del tiempo). En la página principal la última versión dispone de un combo que permite mostrar junto con cada medida la diferencia con los valores tomados para el proyecto en la última medición o en las mediciones realizadas hace una semana o hace un mes. Las variaciones positivas son marcadas con flechas verdes y las negativas con flechas rojas para indicar desviaciones en el código.

La máquina del tiempo por su parte permite mostrar la evolución en el tiempo de cualquiera de las métricas que se deseen en un gráfico. Esta es una herramienta muy útil para comprobar la evolución de los valores de ciertas métricas a lo largo de la evolución del proyecto. El gráfico puede ser configurado para reflejar las medidas que el usuario considera de mayor interés.

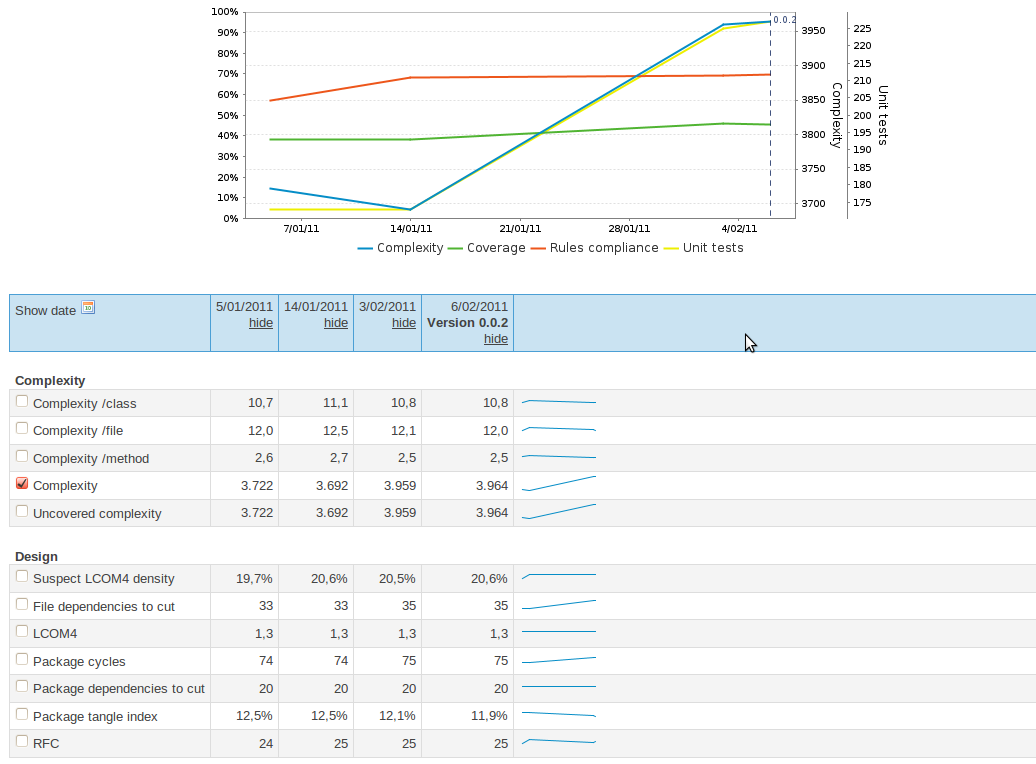


Ilustración 3. Pantalla Time Machine en Sonar

## Artefactos generados tras la construcción del proyecto

Como consecuencia del proceso de construcción del plugin se genera una gran cantidad de ficheros. A continuación se mencionarán aquellos que son de especial interés para el programador.

Si se ejecuta el proceso de generación de la documentación (ref doc) dentro de la carpeta "target/javadoc" del directorio del plugin ("dynamicrefactoring.plugin") se encontrará la documentación en formato javadoc del código fuente del plugin.

Si se ejecuta el proceso normal de construcción del plugin marcado en el ciclo completo se generará los siguientes artefactos:

### Artefactos generados en el directorio del plugin

Dentro de la carpeta "target" un "JAR" que contiene el plugin. Este "JAR" puede ser utilizado para instalar el plugin directamente copiándolo en una instalación de "Eclipse" en el directorio "dropins". Al arrancar "Eclipse" lo reconocerá y lo instalará. Esta es sólo una más de las distintas maneras de instalar el plugin en "Eclipse" para información sobre la forma de instalación recomendada acudir a la sección de instalación del plugin en el manual de usuario (ref manual de usuario).

### Artefactos generados el directorio de tests

En la carpeta "target/sonar" se genera una lista de ficheros "xml" con los resultados de las comprobaciones de estilo y de métricas del código. Estos ficheros son utilizados por "Sonar" para generar la página web en la que se muestran todas las medidas obtenidas del proceso de construcción del proyecto. Los ficheros "xml" pueden ser de interés en caso de que el desarrollador quisiera obtener informes personalizados de dichas mediciones más allá de las proporcionadas por la página web de "Sonar".

En la carpeta "target/surefire-reports" se genera dos ficheros por cada test ejecutado en el proceso de construcción. Ambos ficheros son una representación en texto plano y en formato "xml" respectivamente de la información sobre la ejecución de la prueba. Si la clase de prueba se ha ejecutado sin fallo la información que aparece se corresponde con los nombres de los tests ejecutados en la clase y el tiempo de ejecución de los mismos. Si ha habido errores los tests que fallaron aparecen indicados.

En la carpeta "target/cobertura" se genera un informe "html" con las líneas de código y porcentaje de ramas cubiertas por la ejecución de los tests. Este informe es generado al correr los tests bajo el control de la librería "JaCoCo" (ref) que monitoriza cada una de las instrucciones del proyecto ejecutadas al correr los tests.

### Generación del repositorio de instalación del plugin

El módulo de "Maven" del repositorio "P2" contenido en la carpeta "dynamicrefactoring.p2repository" genera un repositorio que el usuario del plugin puede configurar en Eclipse para instalar el plugin y para obtener actualizaciones del mismo. Ese repositorio ha sido configurado en un servidor web montado en una instancia de una máquina virtual del servicio de "Amazon" "Elastic Cloud" (EC) lo que permite a cualquier usuario obtener la última versión del plugin (ref manual de instalación). La última versión se genera tras ejecutar la construcción del plugin programada de forma diaria a partir del contenido del repositorio de código fuente siempre y cuando todos los tests se ejecuten correctamente.

### Instalación del plugin en el repositorio local de Maven

Al ejecutar la fase de instalación (ref instalacion) el "JAR" del plugin es almacenado en el repositorio local de "Maven" en el PC. La razón de ser de este proceder es que de esta manera se hace disponible el plugin a otros proyectos que pudieran tener dependencias en dicho plugin. Así, si para otro proyecto que se estuviera desarrollando en el mismo equipo de forma independiente se decidiera utilizar alguna de las funcionalidades proporcionadas por el plugin de refactorización, todo lo que se tendría que hacer sería incluir el plugin de refactorización como una de sus dependencias en su fichero "POM". Con dicha mínima configuración "Maven" se encargaría de tomar el "JAR" del plugin de refactorización del repositorio local y añadirlo al classpath cuando ese otro proyecto se fuera a compilar.

# Pruebas unitarias

# Bibliografía