Calidad Estatica con Sonar

==========

Table of Contents

1. Calidad estática del código

1.1. Calidad del Código

Decimos que un código tiene calidad, cuando tenemos facilidad de mantenimiento y de desarrollo.

¿Como podemos hacer que nuestro código tenga mas calidad? Consiguiendo que nuestro código no tenga partes que hagan que:

- Se reduzca el rendimiento.
- Se provoquen errores en el software.
- Se compliquen los flujos de datos.
- Lo hagan mas complejo.
- Supongan un problema en la seguridad.

Tendremos dos técnicas para mejorar el código fuente de nuestra aplicación y, con ello, el software que utilizan los usuarios como producto final:

- Test. Son una serie de procesos que permiten verificar y comprobar que el software cumple con los objetivos y con las exigencias para las que fue creado.
- Análisis estático del código. Proceso de evaluar el software sin ejecutarlo.

1.2. Análisis Estático del Código

Es una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo.

La idea es que, en base a ese código fuente, podamos obtener información que nos permita mejorar la base de código manteniendo la semántica original.

Esta información nos vendrá dada en forma de sugerencias para mejorar el código.

Emplearemos herramientas que incluyen

- Analizadores léxicos y sintácticos que procesan el código fuente.
- Conjunto de reglas que aplicar sobre determinadas estructuras.

Si nuestro código fuente posee una estructura concreta que el analizador considere como "mejorable" en base a sus reglas nos lo indicará y nos sugerirá una mejora.

Se deberian realizar análisis estáticos del código cada vez que se crea una nueva funcionalidad, así como cuando el desarrollo se complica, nos cuesta implementar algo que supuestamente debe ser sencillo.

1.3. PMD

Detecta patrones de posibles errores que pueden aparecer en tiempo de ejecución, por ejemplo

- Código que no se puede ejecutar nunca porque no hay manera de llegar a él.
- Código que puede ser optimizado.
- Expresiones lógicas que puedan ser simplificadas.
- Malos usos del lenguaje, etc
- Tambien incluye detección de CopyPaste (CPD)

La pagina de referencia aquí

Los patrones que se emplean se encuentran catalogados en distintas categorías, se pueden consultar aquí

Se pueden añadir nuevas reglas o configurar las que ya se incluyen en caso de que esto fuera necesario.

Se dipone de un plugin de Maven para la generacion de reportes

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-pmd-plugin</artifactId>
            <version>3.6</version>
            <configuration>
                <linkXref>true</linkXref>
            </configuration>
        </plugin>
    </plugins>
</reporting>
```

Este plugin tambien puede ser configurado como plugin de la fase de Test

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-pmd-plugin</artifactId>
            <version>3.6</version>
            <configuration>
                <failOnViolation>true</failOnViolation>
                <failurePriority>2</failurePriority>
                <minimumPriority>5</minimumPriority>
            </configuration>
            <executions>
                <execution>
                    <phase>test</phase>
                    <goals>
                        <goal>pmd</goal>
                        <goal>cpd</goal>
                        <goal>cpd-check</goal>
                        <goal>check</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Con los parametros

- **failOnViolation** → le indicamos que si hay fallos, haga que la fase de Test falle, supeditamos el exito de los Test al analisis de PMD
- **failurePriority** → le indicamos a partir de que prioridad se considera fallo, las prioridades van de 1 a 5, siendo 1 la maxima y 5 la menor, si se define por ejemplo 2, solo se consideran las reglas con prioridad 1 y 2.
- **minimumPriority** → Minima prioridad de las reglas a evaluar.

Y los goal

- **pmd** → Ejecuta las reglas de pmd
- **cpd** → Ejecuta las reglas de cpd
- **cpd-check** → Chequea los resultados de cpd
- **check** → Chequea los resultados de pmd

1.4. Checkstyle

Inicialmente se desarrolló con el objetivo de crear una herramienta que permitiese comprobar que el código de las aplicaciones se ajustase a los estándares dictados por **Sun Microsystems**.

Posteriormente se añadieron nuevas capacidades que han hecho que sea un producto muy similar a PMD. Es por ello que también busca patrones en el código que se ajustan a categorías muy similares a las de este analizador.

La página de referencia aquí

Dispone de un plugin de Maven

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>2.9.1</version>
        </plugin>
    </plugins>
</reporting>
```

1.5. Findbugs

Es un producto de la Universidad de Maryland que, como su nombre indica, está especializado en encontrar errores.

Tiene una serie de categorías que catalogan los errores

- malas prácticas
- mal uso del lenguaje
- internacionalización
- posibles vulnerabilidades
- mal uso de multihilo
- rendimiento
- seguridad, ...

La página de referencia aquí y la descripción de los bugs aquí

Hay un plugin de maven

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>findbugs-maven-plugin</artifactId>
            <version>2.5.2</version>
        </plugin>
    </plugins>
</reporting>
```

1.6. Cobertura

La Cobertura, representa la cantidad de codigo que cubren las pruebas realizadas sobre el codigo.

Existe un plugin de Maven que permite realizar la medición de la cobertura, presentando el resultado en informes **html** y **xml**.

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>cobertura-maven-plugin</artifactId>
            <version>2.5.2</version>
            <configuration>
                <formats>
                    <format>xml</format>
                    <format>html</format>
                </formats>
            </configuration>
        </plugin>
    </plugins>
</reporting>
```

1.7. Jacoco

Formado por las primeras silabas de **Java Code Coverage**, es otro plugin de cobertura.

La página de referencia se encuentra aquí

```
<build>
    <plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
```

```
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.7.5.201505241946</version>
            <executions>
                <execution>
                    <id>pre-unit-test</id>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                    <configuration>
                        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
                        <destFile>${project.build.directory}/jacoco-ut.exec</destFile>
                        <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas unitarias-->
                        <propertyName>surefireArgLine</propertyName>
                    </configuration>
                </execution>
                <execution>
                    <id>post-unit-test</id>
                    <phase>test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                    <configuration>
                        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
                        <dataFile>${project.build.directory}/jacoco-ut.exec</dataFile>
                        <!-- Establece la ruta donde se genera el reporte para pruebas
unitarias -->
                        <outputDirectory>${project.reporting.outputDirectory}/jacoco-
ut</outputDirectory>
                    </configuration>
                </execution>
                <execution>
                    <id>pre-integration-test</id>
                    <phase>pre-integration-test</phase>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                    <configuration>
                        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
                        <destFile>${project.build.directory}/jacoco-it.exec</destFile>
                        <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas de integracion -->
                        <propertyName>failsafeArgLine</propertyName>
                    </configuration>
                </execution>
```

```
                <execution>
                    <id>post-integration-test</id>
                    <phase>post-integration-test</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                    <configuration>
                        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
                        <dataFile>${project.build.directory}/jacoco-it.exec</dataFile>
                        <!-- Establece la ruta donde se genera el reporte para pruebas de
integracion -->
                        <outputDirectory>${project.reporting.outputDirectory}/jacoco-
it</outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Se ha de configurar igualmente que el agente sea ejecutado en las distintas fases de **test** e **integration-test**, indicando en el plugin con **argLine** la ubicacion del agente de **jacoco** que debera generar los datos del analisis.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.12.4</version>
            <configuration>
                <argLine>${surefireArgLine}</argLine>
                <excludes>
                    <exclude>**/integracion/*.java</exclude>
                </excludes>
                <includes>
                    <include>**/unitarias/*.java</include>
                </includes>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-failsafe-plugin</artifactId>
            <version>2.8</version>
            <configuration>
```

```
                <argLine>${failsafeArgLine}</argLine>
                <excludes>
                    <exclude>**/unitarias/*.java</exclude>
                </excludes>
                <includes>
                    <include>**/integracion/*.java</include>
                </includes>
            </configuration>
            <executions>
                <execution>
                    <id>pasar test integracion</id>
                    <phase>integration-test</phase>
                    <goals>
                        <goal>integration-test</goal>
                    </goals>
                </execution>
                <execution>
                    <id>validar pruebas integracion</id>
                    <phase>verify</phase>
                    <goals>
                        <goal>verify</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

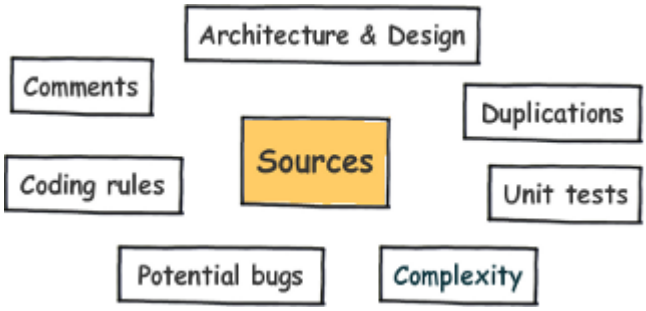
Finalmente se pueden añadir los resultados del analisis al sitio añadiendo

```
<reporting>
    </plugins>
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.7.5.201505241946</version>
        </plugin>
    </plugins>
</reporting>
```

2. Sonarqube

2.1. Introducción

Herramienta que centraliza otras herramientas que analizan la calidad estática del código de un proyecto. Cubre 7 ejes principales de la calidad del software



Ofrece información sobre

- Cobertura
- Complejidad ciclomatica.
- Buenas practicas.

A traves de herramientas como

- Checkstyle
- PMD
- FindBugs

Hay disponible una demo con APIs conocidas aquí

También hay un grupo español, que ofrece información aquí

Algunos de los plugins mas interesantes de Sonar

- PDF Export. Plugin que permite generqar pdf con la info de Sonar
- Motion Chart. Pluginque permite mostrar graficos en movimineto con la evolucion de las metricas
- Timeline. Plugin que visualiza el historico de las metricas
- Sonargraph. Plugin enables you to check and measure the overall coupling and the level of cyclic dependencies
- Taglist. Plugin handles Checkstyle ToDoComment rule and Squid NoSonar rule and generates a report.

2.2. Instalación

Descargar la distribución de aquí

Configurar la base de datos en el fichero

```
SONAR_HOME/conf/sonar.properties
```

Estos son los posibles valores para mysql, de no configurarse se empleará una base de datos Derby, no recomendable para entornos de producción.

```
sonar.jdbc.url: jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8
sonar.jdbc.driverClassName: com.mysql.jdbc.Driver
sonar.jdbc.validationQuery: select 1
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
```

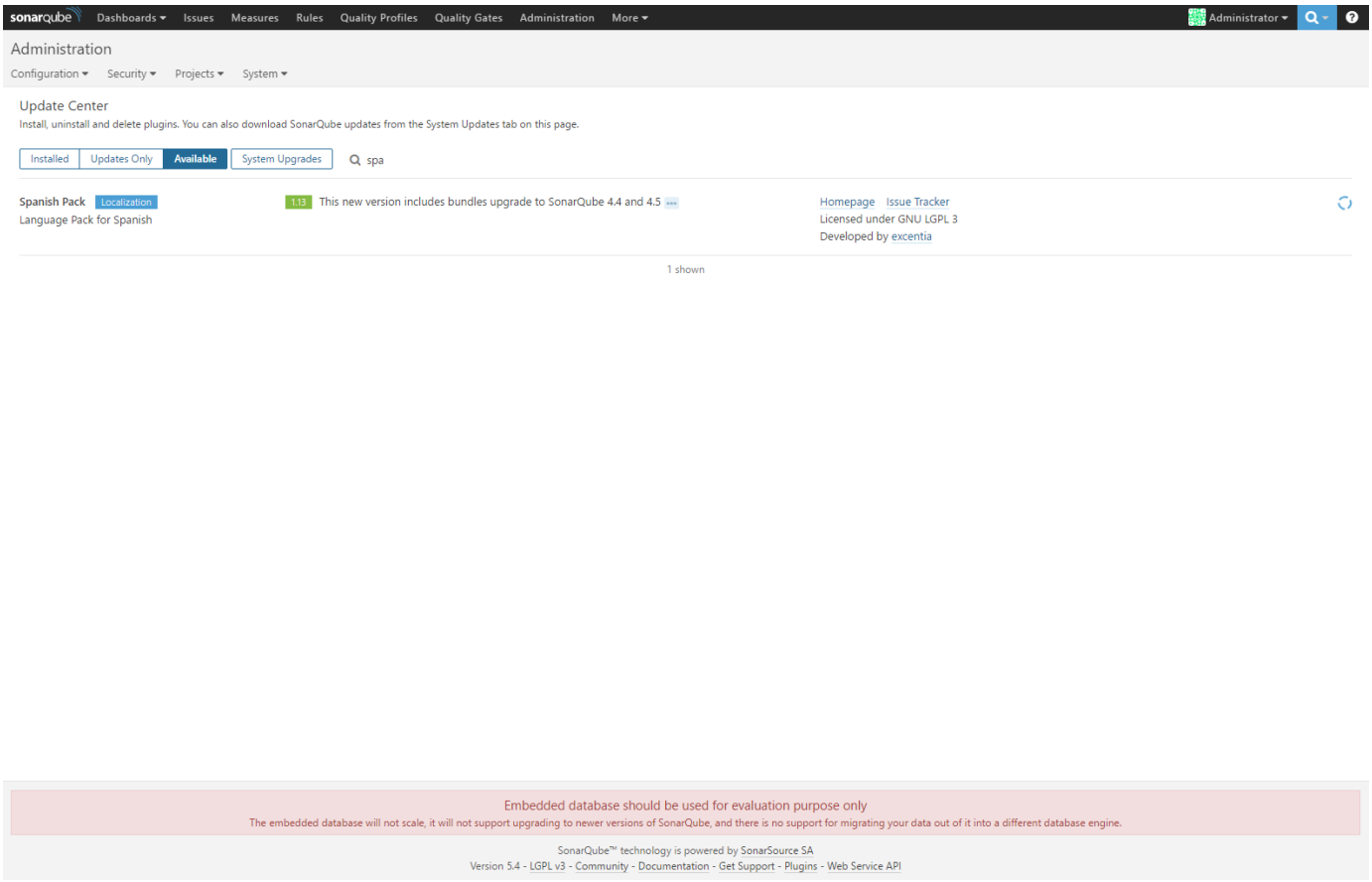
Arrancar el servidor con el comando para el sistema operativo correspondiente que se encuentra en

```
SONAR_HOME/bin/<sistema operativo>/<comando>
```

Esta opción arranca Sonar en el puerto **9000**, el puerto tambien se puede configurar en el anterior fichero de configuración.

Existe un usuario administrador creado por defecto con **admin/admin**

Se puede instalar un plugin para el idioma español, para ello acceder a **Administration/System/Update Center/Available Plugins** y buscar el **Spanish Pack**



2.3. Conceptos

Deuda tecnica: Es un calculo basado únicamente en **reglas** y **evidencias**. La deuda técnica en Sonar, se calcula con la metodología SQALE (Software Quality Assessment based on Lifecycle Expectations). Se mide en dias.

Reglas: Representan aquellos puntos que se desean vigilar en los proyectos. Se pueden definir con un perfil de calidad. Se pueden obtener mas reglas a partir de nuevos Plugins. Se pueden definir nuevas reglas basadas en plantillas.

Evidencias: Son los incumplimientos de las Reglas de calidad que se presentan en el código. Tendrán asociada una severidad. Con las evidencias se pued hacer: Comentar, Asignar, Planificar, Confirmar, Cambiar Severidad, Resolver y Falso Positivo. Todas estas tareas se pueden realizar de forma individual o conjunta. Se pueden definir evidencias manuales.

2.4. Organizacion

La interface web de sonar, se divide en tres partes

- Menu superior
- Menu lateral
- Zona de visualizacion de datos

En el menu superior aparecen los siguientes items

- Cuadros de mando para volver en cualquier momento a la página de inicio
- Proyectos para acceder al listado completo de proyectos, vistas, desarrolladores, etc. o para acceder de forma rápida a proyectos recientemente accedidos
- Medidas, permite definir consultas sobre las medidas, se pueden guardar para visualizarlas en un cuadro de mando.
- Evidencias para acceder al servicio de evidencias
- Reglas para acceder a la página de reglas
- Perfiles navegar y gestionar perfiles de calidad
- Configuración para acceder a la configuración del sistema (acceso restringido a administradores de sistema)
- Conectarse / <Nombre> para conectarse con tu usuario. Dependiendo de tus permisos de usuario, tendrás acceso a diferentes servicios y cuadros de mando. Autenticarte en el sistema te permitirá tener acceso a tu propia interfaz web personalizada. Desde aquí puedes modificar tu perfil y desconectarte.
- Buscar un componente: proyecto, fichero, vista, desarrollador, etc. para acceder rápidamente a él. Pulsa 's' para acceso directo a la caja de búsqueda.

El menú lateral ira cambiando sus opciones dependiendo del área en la que nos encuentremos, de los permisos de usuario y de las extensiones que se hayan incorporado en la instalación. Proporciona acceso a diferentes cuadros de mando y servicios.

2.5. Carga de datos

Para cargar los datos de un proyecto, se puede hacer de varias formas, la mas habituales son

- A través de un plugin de Maven.

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>2.7</version>
</plugin>
```

Y su goal

```
mvn sonar:sonar
```

Para la seleccion de un perfil de Sonar a emplear, se ha de indicar el parametro **sonar.profile**

```
-Dsonar.profile="Mi Perfil"
```

Si se desea publicar la cobertura en Sonar, se ha de seguir los pasos que se pueden encontrar en **https://github.com/SonarSource/sonar-examples/blob/master/projects/languajes/java/code-coverage**

- A través del plugin de jenkins sonarqube.

2.6. Gestion de Usuarios y Seguridad

Se pueden añadir nuevos usuarios, grupos, definir permisos a nivel global o de proyecto, desde **Administration/Security/Users**

2.7. Cuadro de mando

Los cuadros de mando, son los componentes principales de Sonar, ya que son los que nos permiten configurar que información de que proyecto queremos visualizar y como visualizarlo.

Se organizan en columnas, pudiendo seleccionar entre 5 distribuciones distintas.

Se dividen en **Widget**, habiendo **Widget** orientados a distintos propositos, si se busca un Widget concreto se pueden filtrar los Widget mostrados por categorias.

Los Widget a parte de mostrar información, permiten acceder a vistas mas avanzadas, ya que en general los Widget ofrecen resumenes de la información.

Existen Widget que ofrecen información sobre

- Tamaño de los ficheros
- Bloques duplicados
- Mala distribución de la complejidad
- Codigo Spaguetti
- Falta de pruebas unitarias
- Cumplimiento de estandares y defectos potenciales
- Contabilizacion de comentarios
- Eventos en cuanto a la calidad.
- Treemap
- Evidencias y deuda técnica.
- Piramide de deuda técnica. Muestra la deuda tecnica organizada de abajo arriba por prioridad en su resolución.

- Resumen de deuda tecnica. Ofrece un Ratio entre lo que se necesita invertir para solventar la deuda tecnica y lo que se necesita invertir para crear el proyecto desde cero.