PRÁCTICA 6: MÉTRICAS DE CALIDAD

1. Introducción

La actividad tiene como objetivo realizar un análisis de calidad interna de producto software apoyados en una de las herramientas más utilizadas actualmente, SonarQube. Este software realiza análisis estático del código fuente, detectando aspectos como código clonado, concordancia con estándares de programación, búsqueda de bugs, vulnerabilidades, documentación, estructura o cobertura de pruebas. A cada uno de estos aspectos detectados, le asigna un valor de tiempo llamado "deuda técnica", que podríamos definir como el coste y los intereses a pagar por hacer mal las cosas (el sobre esfuerzo a pagar por mantener un software mal hecho). Hay que tener en cuenta que sólo una parte de esta deuda es intrínseca al código y podemos medirla mediante analizadores, y que también puede venir asociada a aspectos relacionados con el proceso, arquitectura, diseño, tecnologías, etc.

2. Objetivos

(1) Aprender a analizar varios aspectos referentes a la calidad de producto software.
(2) Aprender a utilizar una herramienta software que permita el análisis de calidad de producto.
(3) Aprender a realizar un plan de mejora de la calidad de producto que priorice la resolución de las incidencias más importantes, resolver dichas incidencias y observar la mejora obtenida.

3. Taller

En esta etapa analizaremos el código fuente de un proyecto de ejemplo. En el análisis observaremos distintas métricas que sacarán a la luz evidencias referentes a errores muy comunes de desarrollo, que son perjudiciales para la calidad de nuestro proyecto e incrementan la deuda técnica.

Seguiremos el siguiente procedimiento, que tiene como objetivo conocer las medidas más importantes que nos proporciona Sonarqube.

Carlos Blanco

Vista general

- Quality Gate
- Incidencias y deuda técnica
- Clasificación SQALE
    - Confiabilidad (Bugs)
    - Seguridad (Vulnerabilidades)
    - Mantenimiento (Code Smells)
- Cobertura
- Duplicados

Evidencias

- Listado de evidencias
- Observar una determinada evidencia (deuda técnica que aporta, problema detectado y posible solución, lugar del código donde se encuentra).
- Filtros:
    - Evidencias por tipo: bug, vulnerabilidad, code smell.
    - Evidencias por severidad: bloqueante, crítica, mayor, menor, info.
    - Evidencias por etiqueta.
    - Evidencias por regla.
    - Evidencias que afectan a un determinado fichero.
    - Etc.
- Filtros combinados:
    - Permiten resolver preguntas del tipo, ¿en qué fichero se concentran los code smells más severos?

Security hotspots (añadida en la última versión)

- Riesgos de seguridad potenciales identificados según OWASP Top 10, CWE Top 25 2019 y 2020, y CWE.

Medidas

- Hotspots. Calificación SQALE, bugs detectados, deuda técnica asociada y qué ficheros concentran los principales problemas.
    - Confiabilidad (reliability) → bugs
        - Lugares donde podemos obtener comportamiento no esperado
    - Seguridad (security) → vulnerabilidades
        - Puntos que pueden explotarse para realizarnos ataques
    - Revisión de seguridad (security review) → riesgos potenciales
    - Mantenibilidad (mantenibility) → code smells
        - Aspectos que hacen que el código sea más difícilmente actualizable
- Código duplicado. Porcentaje, líneas, bloques y archivos.
- Medidas referentes al tamaño (ficheros, clases, funciones, líneas de código).
- Cálculo de complejidades ciclomática y cognitiva.

4. Actividades a realizar

Una vez conocidas las principales funcionalidades de Sonarqube, en esta etapa prepararemos el entorno de trabajo y analizaremos un proyecto real siguiendo los pasos que se detallan a continuación.

4.1 Instalación de herramientas

SonarQube presenta una arquitectura cliente servidor y ofrece numerosos plugins para su integración con otras herramientas de integración continua (Travis, Jenkins), control de versiones (Git, GitHub), IDEs (Eclipse, Visual Studio), etc.

Servidor SonarQube

En primer lugar vamos a instalar un servidor local en nuestro equipo.

- Entramos en la web oficial de sonarqube http://www.sonarqube.org
- Descargamos la Community Edition: Versión 8.9 LTS (mayo de 2021)
- No hace falta instalarlo sino que obtendremos un fichero .zip que debemos descomprimir y en encontraremos dentro de la carpeta bin, carpetas con los ejecutables para distintos sistemas operativos.
    - En el caso de Linux, ejecutar el archivo sonar.sh ./sonar.sh start
    - En el caso de Mac, ejecutar el archivo sonar.sh ./sonar.sh start Posiblemente saltarán varias alertas para pedir que concedáis permisos de seguridad al wrapper y una librería, lo cuál habrá que autorizar en ajustes del sistema -> seguridad.
    - En el caso de Windows, ejecutar el archivo StartSonar.bat
    - Dejaremos la ventana minimizada ya que el algún sistema operativo al cerrarla podemos matar el proceso. También tenemos comandos adicionales para comprobar el estado (sonar.sh status) o detener el servidor con (sonar.sh stop).
- Si da un error lo más probable es que sea por la versión de Java (requiere versión 11).
- Tardará un rato en levantar el servidor y podemos comprobar que está todo correcto accediendo desde un navegador web a la URL http://localhost:9000

  Os pedirá cambiar la contraseña de admin, lo cuál deberemos hacer y apuntar la nueva ya que la vamos a necesitar más adelante para cargar proyectos.

Cargar análisis en SonarQube

Para lanzar los análisis de nuestros proyectos y cargarlos en el servidor deberemos:

- Ejecutar en el proyecto que queremos analizar, **mvn sonar:sonar**
- Añadir la siguiente configuración al fichero **settings.xml** (en la carpeta mvn/conf)

```
<pluginGroups>
   <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
</pluginGroups>
<profiles>
 <profile>
   <id>sonar</id>
   <activation> <activeByDefault>true</activeByDefault></activation>
   <properties>
    <sonar.host.url>http://localhost:9000</sonar.host.url>
    <sonar.login>admin</sonar.login>
    <sonar.password>vuestra contraseña</sonar.password>
   </properties>
 </profile>
</profiles>
```

- Y la siguiente al fichero **pom.xml** del proyecto

  Hay que añadir el plugin Jacoco para que pase los informes de pruebas a sonar y la identificación del proyecto (key y name).

```
<build>
 <plugins>
  <plugin>
   <groupId>org.jacoco</groupId>
   <artifactId>jacoco-maven-plugin</artifactId>
   <version>0.8.7</version>
   <executions>
    <execution>
     <goals><goal>prepare-agent</goal></goals>
    </execution>
   <execution>
    <id>report</id><phase>test</phase>
    <goals><goal>report</goal></goals>
   </execution>
   </executions>
  </plugin>
 </plugins>
</build>
<properties>
 <sonar.projectKey>Carlos-Practica5-SinRefactorizar</sonar.projectKey>
 <sonar.projectName>Carlos-Practica5-SinRefactorizar</sonar.projectName>
</properties>
```

4.2 Análisis de la mejora obtenida en la práctica de refactorización

El proyecto a analizar será el correspondiente a la práctica de refactorización. En primer lugar cargaremos en el servidor de SonarQube el proyecto sin refactorizar y el refactorizado, es decir, los proyectos inicial y final de la práctica anterior.

Una vez cargados los dos proyectos en SonarQube pasaremos a observar la mejora obtenida con la refactorización. En este punto se realiza un **análisis más superficial** en el que se deberá observar principalmente el cuadro de mando resumen de ambos proyectos y las medidas que nos ha calculado (como las de complejidad).

4.3 Análisis de calidad del código refactorizado

Centrándonos en el proyecto refactorizado, daremos un paso más en su mejora centrándonos en las incidencias de calidad y su resolución. En esta ocasión habrá que realizar un **análisis más profundo**, navegando por las distintas opciones y aplicando filtros de forma que detectemos dónde están los mayores problemas de calidad del código. Podremos observar los tipos de incidencias, su severidad, a qué fichero pertenecen, que regla se incumple más, cuánta deuda técnica genera cada aspecto, etc. y combinarlos, por ejemplo, para saber por el incumplimiento de qué regla se están generando más incidencias de mantenibilidad del tipo crítico.

Este análisis no consiste en copiar y pegar sin sentido todas las pantallas de sonar, sino que se pide analizar lo que está ocurriendo para presentar el resultado del análisis de forma que se identifiquen dónde están los problemas principales que afectan a la calidad del código.

4.4 Mejora de calidad

En proyectos de cierta entidad no se dispondría de tiempo suficiente para resolver todas las evidencias y habrá que centrarse en las que tengan una mayor relación entre coste de realización (tiempo de quitar la incidencia) y beneficio obtenido (reducción de deuda técnica, eliminar vulnerabilidades críticas, etc.).

En este apartado habrá que definir un **plan de mejora de la calidad del código** en el que se propongan una lista de acciones correctivas a realizar y se razone por qué se ha optado por ellas. Se pide que esta lista esté priorizada de más importante y prioritaria a menos.

A continuación se procederá a **resolver** varias de estas incidencias sobre el propio código (no hace falta resolverlas todas y además, si alguna es especialmente complicada de resolver por la naturaleza del código, se puede pasar a otra).

Finalmente, se realizará un **análisis final**, para observar la mejora obtenida. Dicho análisis es superficial, centrado en el cuadro de mando y las medidas que proporciona.

5. Criterios de Evaluación y Aclaraciones

Se deberá realizar un informe a modo de memoria y entregarlo en la correspondiente tarea de Moodle. El informe incluirá:

- Un análisis de la mejora obtenida en la refactorización realizada en la práctica anterior en cuanto a las medidas que arroja SonarQube al respecto.
- Un análisis de la situación del código de partida (el ya refactorizado) en cuanto a la calidad de producto. Este análisis no consiste en copiar y pegar sin sentido todas las pantallas de sonar, sino que se pide analizar lo que está ocurriendo para identificar dónde están los problemas principales que afectan a la calidad del código.
- La mejora del código de partida. Consistente en:
    - El plan de mejora de la calidad del código propuesto, indicando la lista priorizada de acciones correctivas a realizar y el razonamiento de por qué se ha optado por ellas.
    - Qué incidencias se han resuelto y cómo.
    - El análisis de la situación final del código en cuanto a calidad de producto.

Anexo

En casos excepcionales, que haya problemas irresolubles para ejecutar el servidor de sonar en vuestra máquina local, se podría utilizar el servidor sonarcloud de 4º.

Se lanzará de igual forma (mvn sonar:sonar) pero la dirección a la que se subirá es https://sonarcloud.io/organizations/isuc/projects

La configuración necesaria es la siguiente:

- Fichero **pom.xml** del proyecto según las instrucciones iniciales
- Fichero **settings.xml** (en la carpeta mvn/conf)

```
<pluginGroups>
   <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
</pluginGroups>
<profiles>
 <profile>
   <id>sonar</id>
   <activation> <activeByDefault>true</activeByDefault></activation>
   <properties>
    <sonar.host.url>https://sonarcloud.io</sonar.host.url>
    <sonar.organization>isuc</sonar.organization>
    <sonar.login>120537998e2c122476f30cade8d4a25865210fa6</sonar.login>
   </properties>
 </profile>
</profiles>
```