

- Laboratorio de Desarrollo y Herramientas -

Práctica 3

Herramientas de Calidad del Producto Software y Documentación.

SonarQube, Maven y Doxygen

Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología

Grado: Ingeniería Informática

Asignatura: Laboratorio de Desarrollo y Herramientas

Alumna: Rebeca Rodríguez Rodríguez

Curso: 2023-24

Fecha: 28 oct 2023

Índice

<u>Introducción</u>	<u>2</u>
<u>Desarrollo de la práctica</u>	<u>3</u>
<u>Configuración de plugins de Maven</u>	<u>3</u>
<u>Análisis del proyecto</u>	<u>4</u>
<u>Checkstyle</u>	<u>4</u>
<u>PMD</u>	<u>5</u>
<u>OWASP Dependency-Check</u>	<u>5</u>
<u>SonarQube</u>	<u>6</u>
<u>Identificación y solución de problemas de seguridad</u>	<u>8</u>
<u>Security Hotspots</u>	<u>8</u>
<u>Bugs</u>	<u>9</u>
<u>Referencias</u>	<u>12</u>

Introducción

Esta práctica consiste en analizar y documentar un proyecto software dado [\[1\]](#) con las herramientas SonarQube, Maven y Doxygen.

SonarQube [\[2\]](#), como vimos en la práctica anterior, es un software de código abierto para la revisión y evaluación de la calidad del código fuente que se le proporcione. Esta herramienta proporciona diferentes medios para analizar el código, como análisis de código estático, detección de errores (bugs), detección de código duplicado, análisis de la complejidad del código, entre otros. Además, es una herramienta muy versátil, ya que es compatible con numerosos lenguajes de programación, como pueden ser Java, C, C#, C++, Python, y un largo etcétera.

Por otro lado, Maven [\[3\]](#) es una herramienta software para la construcción y configuración de proyectos software en lenguaje Java. Con Maven, el proceso de construcción del software es más sencillo, y además nos proporciona una estructura uniforme y predefinida para nuestro código, lo que resulta útil sobre todo para proyectos grandes. Cabe destacar el archivo pom.xml [\[4\]](#), que es la unidad básica de trabajo en Maven, pues contiene toda la información del proyecto, incluyendo dependencias, plugins y perfiles propios del mismo.

Por último, Doxygen [\[5\]](#) es una herramienta de generación de documentación para diferentes lenguajes de programación, entre los que se encuentra Java. Doxygen es capaz de generar esta documentación en línea o de forma local a partir de archivos de código fuente que estén debidamente documentados, aunque también se puede configurar para extraer la estructura del código de archivos no documentados, lo que resulta útil para navegar rápidamente en proyectos de gran tamaño. Además, Doxygen permite crear gráficos que muestran las relaciones entre los diferentes elementos del código de forma automática.

A continuación, vamos a configurar algunos de los plugins que soporta Maven, para posteriormente realizar un análisis del proyecto facilitado e identificar los principales problemas de seguridad que presenta.

Desarrollo de la práctica

Configuración de plugins de Maven

Para esta práctica he utilizado los siguientes plugins soportados por Maven:

- **Checkstyle**. Checkstyle es una herramienta de desarrollo de software que ayuda a escribir código Java sujeto a un determinado estilo de código. Con Maven, este plugin genera informes que muestran información sobre el proyecto software analizado, tal como las advertencias y errores detectados en el código que incumplen el conjunto de reglas de estilo elegido. [\[6\]](#)
- **PMD**. PMD es una herramienta de análisis de código que sirve para detectar defectos comunes de código, como declaración de variables no utilizadas. Con Maven, este plugin ejecuta esta herramienta y genera un informe de resultados. [\[7\]](#) [\[8\]](#)
- **OWASP Dependency-Check**. Dependency-Check es un software que permite escanear nuestro código e identificar posibles vulnerabilidades. Con Maven, comprueba las dependencias del proyecto y si estas tienen alguna vulnerabilidad conocida, y genera un informe de resultados. [\[9\]](#) [\[10\]](#)

Análisis del proyecto

Para llevar a cabo el análisis del proyecto, me he ayudado de los plugins de Maven mencionados en el apartado anterior y de la herramienta SonarQube.

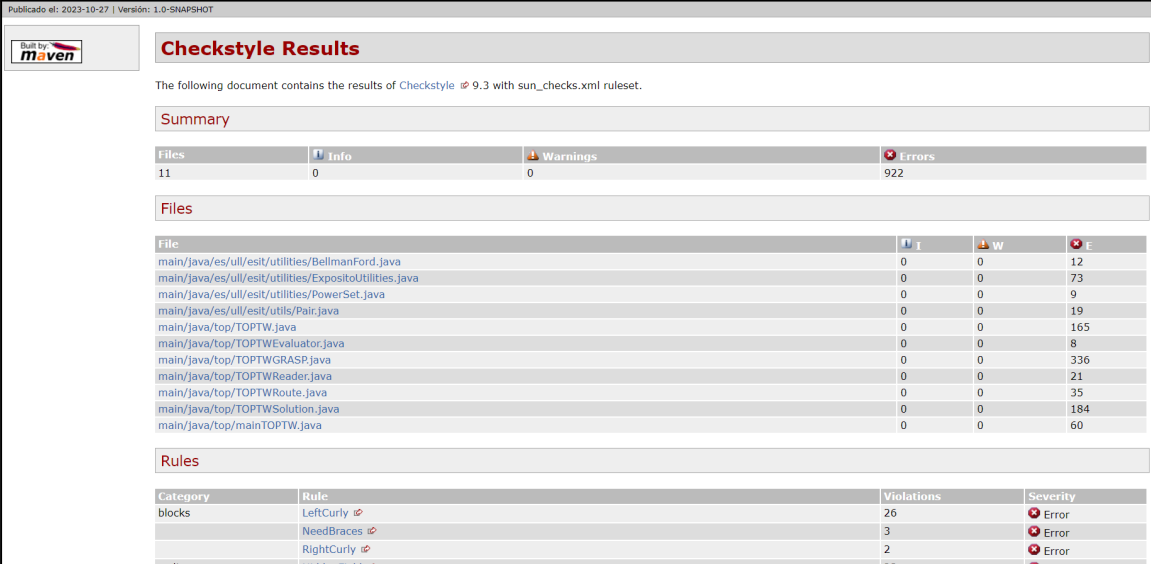
Cabe destacar que, antes de poder analizar el código, he creado y configurado el archivo pom.xml, fundamental para poder utilizar Maven sobre el proyecto, y también he trasladado todas las subcarpetas del proyecto a un directorio src.

A continuación se detalla el proceso realizado para analizar el proyecto con las diferentes herramientas mencionadas:

Checkstyle

En primer lugar, he ejecutado la orden “mvn checkstyle:checkstyle” en el directorio del proyecto. Con esto, se ha lanzado el plugin Checkstyle y se ha creado un informe de resultados en el subdirectorio target/site del proyecto.

Como podemos ver en la Figura 1, en este informe se ha analizado el código en base al conjunto de normas sun_check.xml, y que el proyecto contiene 11 ficheros y un total de 922 errores. Esto quiere decir que hay 922 casos en los que el código no cumple con las normas de estilo mencionadas, por lo que claramente el código no cumple este estilo.



Publicado el: 2023-10-27 | Versión: 1.0-SNAPSHOT

Checkstyle Results

The following document contains the results of Checkstyle 9.3 with sun_checks.xml ruleset.

Summary

Files	Info	Warnings	Errors
11	0	0	922

Files

File	I	W	E
main/java/es/ull/esit/utilities/BellmanFord.java	0	0	12
main/java/es/ull/esit/utilities/ExpositoUtilities.java	0	0	73
main/java/es/ull/esit/utilities/PowerSet.java	0	0	9
main/java/es/ull/esit/utilities/Pair.java	0	0	19
main/java/top/TOPTW.java	0	0	165
main/java/top/TOPTWEvaluator.java	0	0	8
main/java/top/TOPTWGRASP.java	0	0	336
main/java/top/TOPTWReader.java	0	0	21
main/java/top/TOPTWRoute.java	0	0	35
main/java/top/TOPTWSolution.java	0	0	184
main/java/top/mainTOPTW.java	0	0	60

Rules

Category	Rule	Violations	Severity
blocks	LeftCurly	26	Error
	NeedBraces	3	Error
	RightCurly	2	Error
coding	HiddenField	23	Error

Figura 1. Captura de pantalla del informe generado por CheckStyle

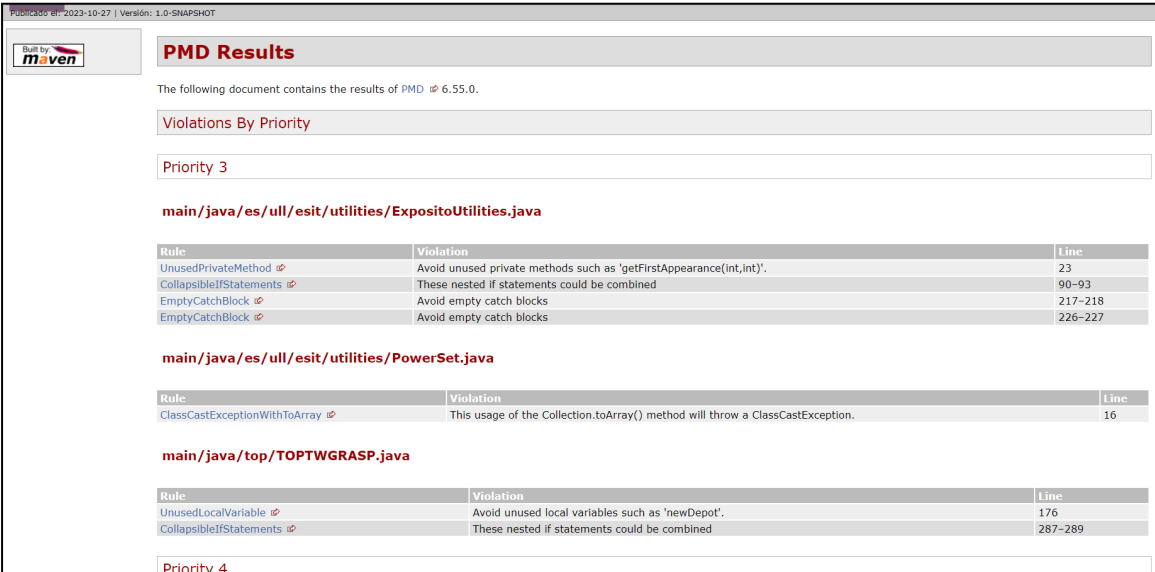
En este informe también se nos muestra la lista de ficheros del proyecto con la cantidad de errores que hay en cada uno de ellos, las reglas de estilo establecidas con el número de veces que se ha violado cada una, y una lista con todos y cada uno de los errores detectados junto a su mensaje y la línea de código donde se encuentran.

PMD

Después de utilizar Checkstyle para analizar el código, he ejecutado el comando “mvn pmd:pmd” para lanzar el plugin PMD. Esto ha generado otro informe, pmd.html, que contiene información sobre los defectos detectados en el código.

En este informe se nos muestran los defectos detectados por orden de prioridad, de menor a mayor. Estos defectos, además, se encuentran separados por ficheros junto a una descripción de cada uno y el número de línea en la que se encuentra.

En la Figura 2 podemos observar el aspecto que tiene este informe. Se han detectado 11 defectos en total, que son en su mayoría malas prácticas.



Rule	Violation	Line
UnusedPrivateMethod	Avoid unused private methods such as 'getFirstAppearance(int,int)'.	23
CollapsibleIfStatements	These nested if statements could be combined	90-93
EmptyCatchBlock	Avoid empty catch blocks	217-218
EmptyCatchBlock	Avoid empty catch blocks	226-227

Rule	Violation	Line
ClassCastExceptionWithToArray	This usage of the Collection.toArray() method will throw a ClassCastException.	16

Rule	Violation	Line
UnusedLocalVariable	Avoid unused local variables such as 'newDepot'.	176
CollapsibleIfStatements	These nested if statements could be combined	287-289

Figura 2. Captura de pantalla del informe generado por PMD

OWASP Dependency-Check

Por último, he lanzado el plugin Dependency-Check con la orden “mvn site”. Tras la ejecución de este comando, se ha creado el informe de esta herramienta, que contiene

información acerca de las dependencias del proyecto y de las posibles vulnerabilidades presentes en las mismas.

En la Figura 3 se muestra una captura de pantalla de este informe. Podemos observar que el proyecto no tiene dependencias, por lo que no se ha detectado ninguna vulnerabilidad.

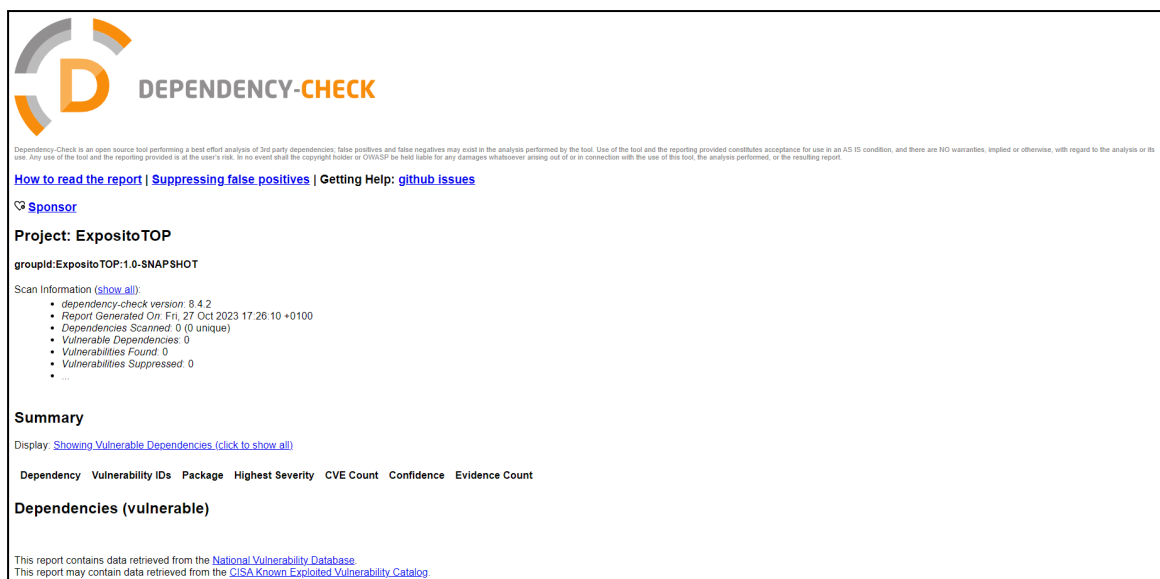


Figura 3. Captura de pantalla del informe generado por OWASP Dependency-Check

SonarQube

Tras analizar el proyecto con la ayuda de todos los plugins instalados, he utilizado también SonarQube. Para ello, he iniciado la herramienta desde una terminal y he accedido a localhost:900 con mi navegador. Desde ahí, he creado un nuevo proyecto “ExpositoTOP”, que he configurado para poder realizar el análisis con Maven, como se ve en la Figura 4.

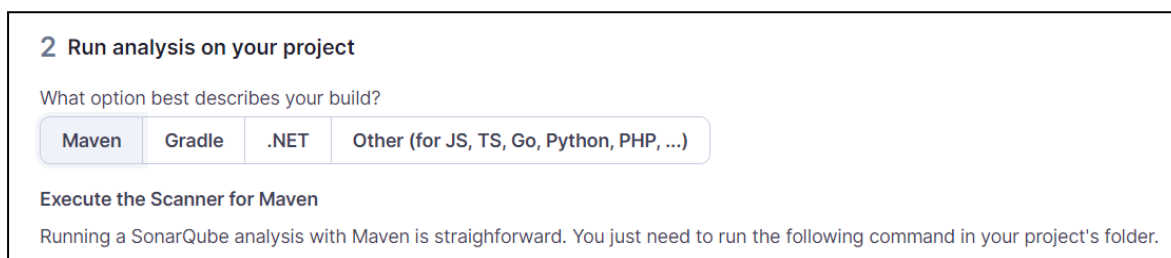


Figura 4. Captura de pantalla de SonarQube donde se establece que el proyecto se construye con Maven

Al ejecutar el comando proporcionado por SonarQube, he obtenido los resultados que se muestran en la Figura 5.

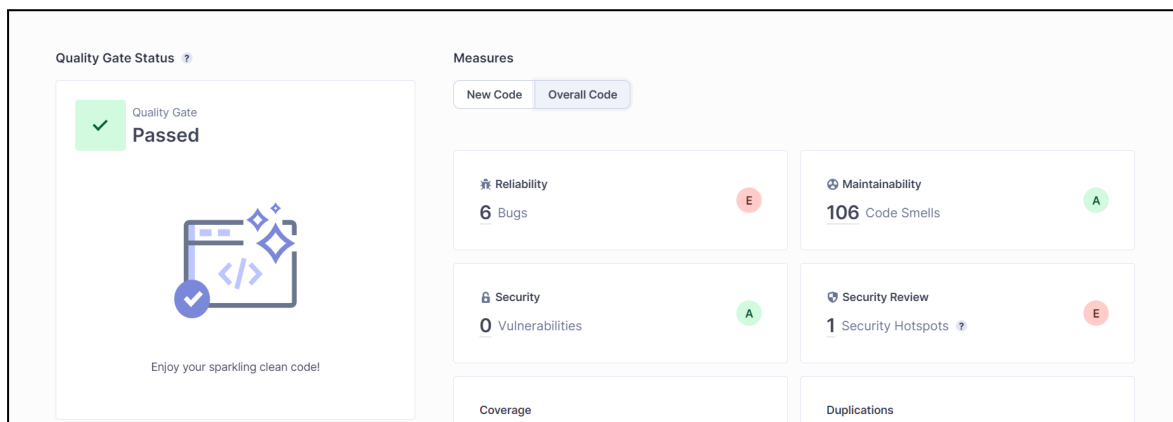


Figura 5. Captura de pantalla de los resultados del análisis de ExpositoTOP de SonarQube

Podemos observar que el código contiene problemas de confiabilidad y de mantenibilidad, y que una porción de código necesita ser evaluada manualmente para comprobar si presenta una vulnerabilidad o no. A pesar de esto, el proyecto pasa la *Quality Gate*, lo que significa que cumple con los niveles mínimos de calidad.

Identificación y solución de problemas de seguridad

De acuerdo con el análisis de SonarQube comentado anteriormente, el proyecto ExpositoTOP no contiene ningún problema de seguridad. Sin embargo, sí que contiene algunos problemas de confiabilidad y hay un fragmento de código que necesita ser evaluado manualmente. Este apartado lo dedicaremos a dar solución a algunos de estos problemas.

Security Hotspots

SonarQube detecta una pieza de código que podría ser sensible y que requiere de revisión manual para comprobar si realmente se trata de una vulnerabilidad. Esta pieza de código es la que se muestra en la Figura 6.

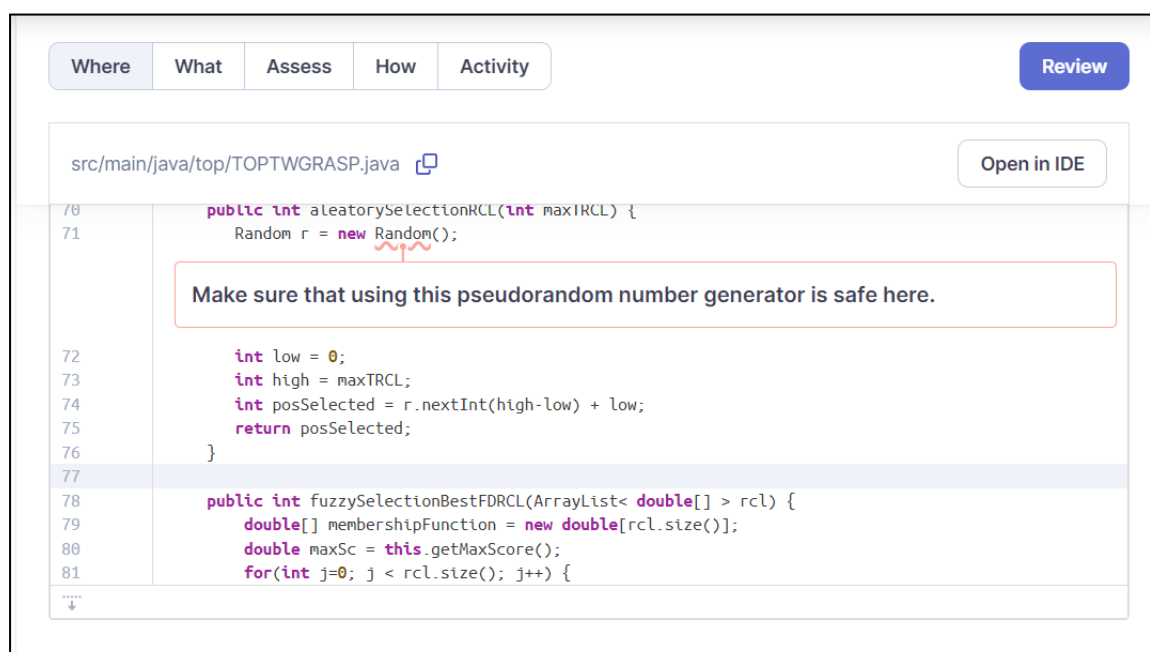


Figura 6. Security Hotspot detectado por SonarQube

Este Security Hotspot se debe a que SonarQube considera que el uso de generadores de números pseudoaleatorios representa un riesgo de seguridad, ya que puede dar lugar a vulnerabilidades. Cuando el software genera valores predecibles en situaciones que requieren imprevisibilidad, los atacantes pueden adivinar los valores futuros y utilizarlos para suplantar usuarios o acceder a información sensible.

Por lo tanto, se desaconseja el uso de la clase `java.util.Random` y el método `java.lang.Math.random()` en aplicaciones críticas en términos de seguridad o para proteger datos sensibles.

En su lugar, se recomienda el uso de la clase `java.security.SecureRandom`, que emplea un generador de números aleatorios criptográficamente sólido. Vamos a sustituir la línea de código problemática resaltada en la Figura 6 por:

Java

```
SecureRandom r = new SecureRandom();
```

Bugs

El proyecto tiene un total de seis bugs. Vamos a arreglar los que se muestran en la Figura 7.

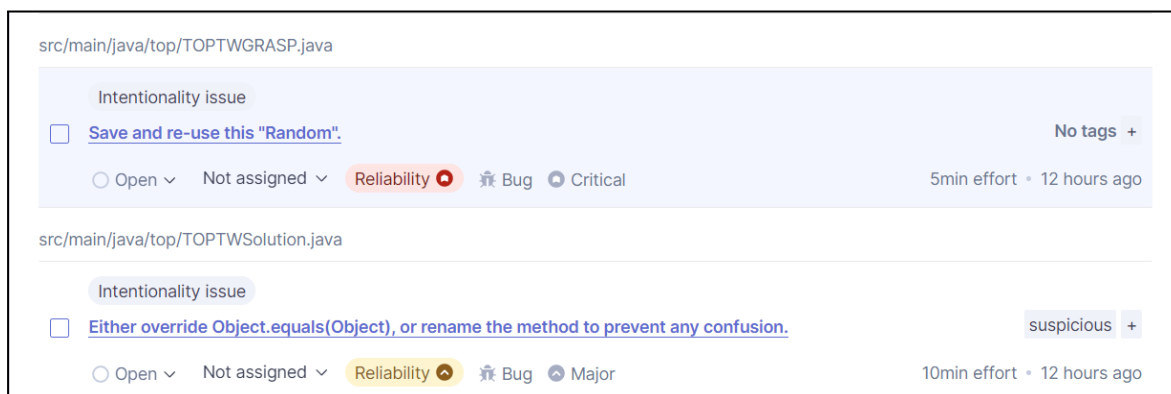


Figura 7. Bugs a solucionar detectados por SonarQube

El primer bug se refiere a la misma línea de código resaltada en la Figura 6. SonarQube detecta como error generar un nuevo número aleatorio cada vez que se ejecuta el método `“public int aleatorySelectionRCL(int)”` porque lo considera ineficiente y, en función del JDK utilizado, podría generar números que no son aleatorios. Por ello, propone declarar esta variable como variable de clase, tal como aparece en la Figura 8.

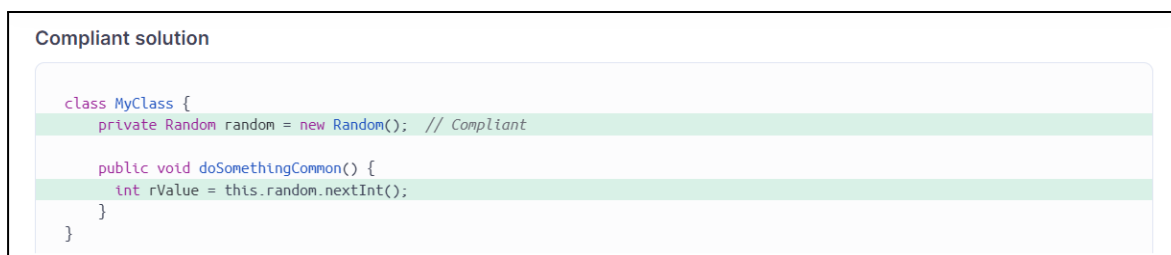


Figura 8. Ejemplo de código proporcionado por SonarQube en el que se soluciona el bug de Random

Por otro lado, el segundo bug se refiere a la línea 56 del fichero “src/main/java/top/TOPTWSolution.java” del proyecto. En la Figura 9 se muestra el mensaje de error de SonarQube.

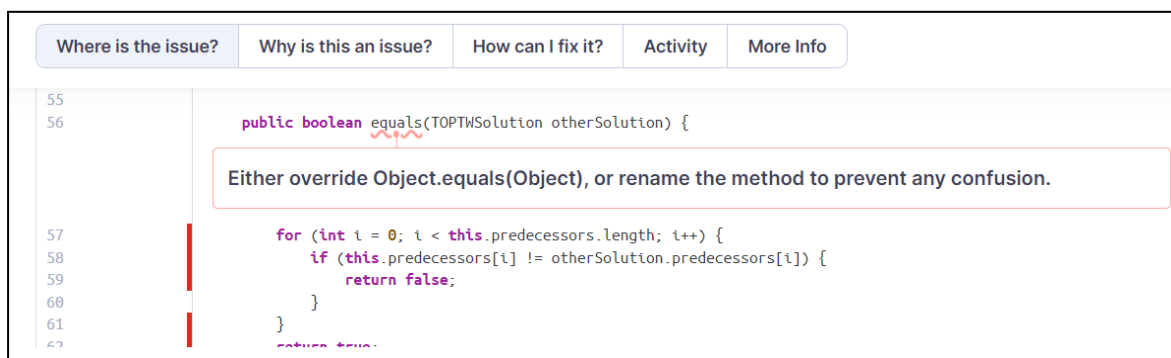


Figura 9. Bug detectado por SonarQube con respecto al método equals

En Java, el método “Object.equals()” se utiliza para comparar objetos y suele sobrescribirse en las clases para definir un criterio de igualdad personalizado. La implementación predeterminada de “equals()” en “Object” compara las referencias de memoria, lo que verifica si los objetos son la misma instancia en memoria. SonarQube considera que debe usarse el nombre del método “equals” exclusivamente para anular “Object.equals(Object)” y evitar confusiones, y es por eso que ha detectado este bug.

Para solucionarlo, he cambiado el nombre del método por “isEqual”.

Una vez realizados los cambios pertinentes en el código, volvemos a ejecutar el comando de Maven para realizar un nuevo análisis con SonarQube. Ahora, obtenemos el resultado que se muestra en la Figura 10.

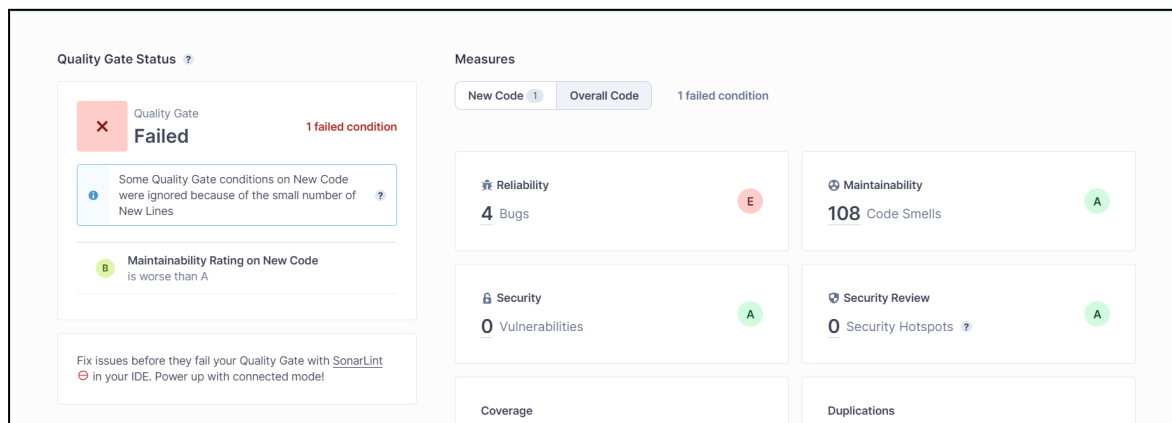


Figura 10. Captura de pantalla de los resultados del segundo análisis de ExpositoTop con SonarQube

Ahora SonarQube solo detecta cuatro *bugs* y ya no hay ningún *security hotspot*.

Referencias

- [1] [ExpositoTOP.rar \(https://campusingenieriaytecnologia2324.ull.es\)](https://campusingenieriaytecnologia2324.ull.es)
- [2] [SonarQube 10.2 \(sonarsource.com\)](https://sonarsource.com)
- [3] [Maven - Introduction \(https://maven.apache.org\)](https://maven.apache.org)
- [4] [Java: ¿Qué es Maven? ¿Qué es el archivo pom.xml? \(www.campusmvp.es\)](http://www.campusmvp.es)
- [5] [Doxygen: Doxygen \(https://www.doxygen.nl\)](https://www.doxygen.nl)
- [6] [checkstyle - Checkstyle 10.12.4 \(https://checkstyle.org\)](https://checkstyle.org)
- [7] [PMD \(pmd.github.io\)](https://pmd.github.io)
- [8] [Apache Maven PMD Plugin - Introduction \(maven.apache.org\)](https://maven.apache.org)
- [9] [dependency-check - About \(jeremylong.github.io\)](https://jeremylong.github.io)
- [10] [dependency-check-maven - Plugin Documentation \(jeremylong.github.io\)](https://jeremylong.github.io)