Universidad de Antioquia

Ingeniería de Sistemas

Calidad del Software

Práctica 1 - Deuda técnica

PRESENTA:
Jonatan Stiven Restrepo Lora
Daniela Andrea Pavas Bedoya

Tutor Principal:
Robison Coronado Garcia

1. Objetivos

1.1. Objetivo General

Comprender los conceptos fundamentales de la calidad del software y su relación con la deuda técnica, así como desarrollar la capacidad para identificar, analizar y abordar eficazmente la deuda técnica mediante el uso de herramientas y recursos de software de terceros.

1.2. Objetivos Específicos

- Investigar y familiarizarse con una herramienta de análisis de código estático, como SonarQube, SonarCloud u otras disponibles, para comprender su funcionamiento y capacidades.
- Configurar adecuadamente los quality gates en la herramienta de análisis de código estático de acuerdo con los estándares y requisitos de calidad del proyecto.
- Comprender a fondo la importancia de la deuda técnica en el desarrollo de software, reconociendo cómo puede afectar la calidad, el costo y la eficiencia del proyecto.

2. Herramientas de software empleadas

- Sonar Cube
- Java
- Gradle
- Junit 5
- Git
- Mockito

3. Marco teorico

La calidad del software es fundamental en el desarrollo de aplicaciones. Utilizamos herramientas como SonarQube y SonarLint para identificar problemas antes de que afecten a los usuarios. Además, implementamos pruebas unitarias y establecemos "quality gates"para mantener la calidad del código y reducir la deuda técnica.

4. Procedimiento

4.1. Caso de desarrollo

Diseñar un gestor de tareas el cual contiene la siguiente información:

- Título de la tarea.
- descripción.
- Estado en la que se encuentre:
  - En progreso
  - Creada
  - terminada.

Además, entre el funcionamiento de la aplicación se podrá hacer lo siguiente:

- Crear nuevas tareas.
- Completar tareas.
- Iniciar tareas.
- Eliminar tareas.
- Listar las tareas pendientes.
- Listar las tareas completadas.
- Cada tarea debe tener un identificador único.

4.2. Ajuste de los Quality Gates

- Fiabilidad **A (No bugs)**
- Clasificación de seguridad **A (Ninguna vulnerabilidad)**
- Grado de mantenimiento **A (El coeficiente de endeudamiento técnico es inferior al 5 %)**
- Cobertura mayor a **95 %**
- Líneas duplicadas (%) **<2 %**
- Pruebas unitarias omitidas **0**
- Éxito de las pruebas unitarias (%) **100 %**

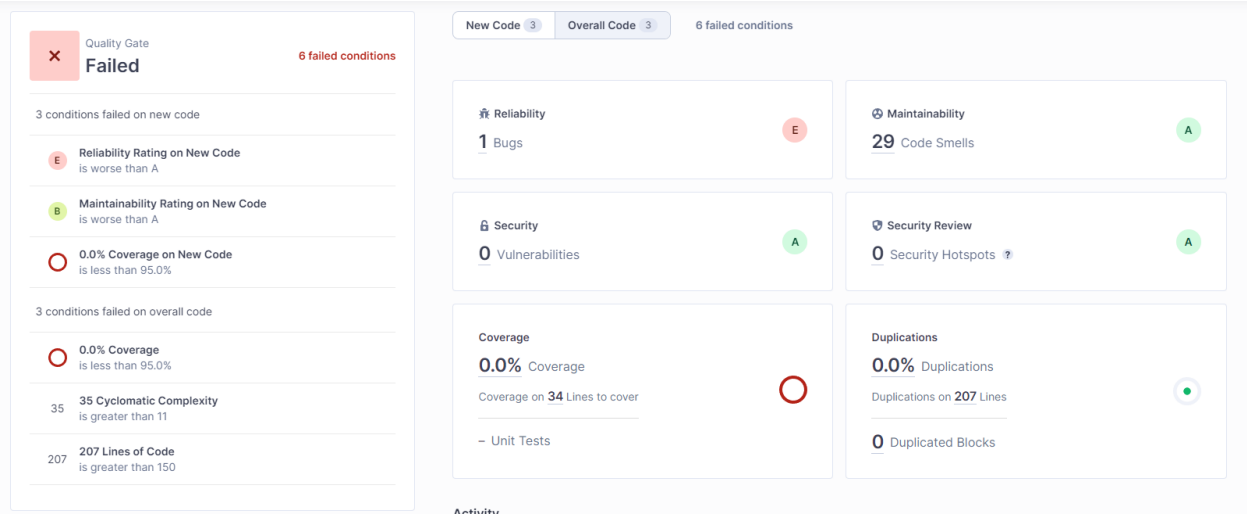

Figura 1: Repore de SonarQube

4.3. Reporte SonarQube

Como se puede observar, el código inicial no se encuentra para nada en óptimas condiciones, ya que este cuenta con múltiples errores y recomendaciones por parte del analizador de código, tales como: El código cuenta con bugs.

1. Hay cero cobertura de código con pruebas unitarias.
2. La complejidad ciclomatica está demasiado alta, para las reglas definidas antes.
3. Hay un error de seguridad con respecto al token de inicio de sección del SonarQube
4. Hay clases que hacen muchas cosas.
5. Se deben definir variables constantes, varios textos repetidos

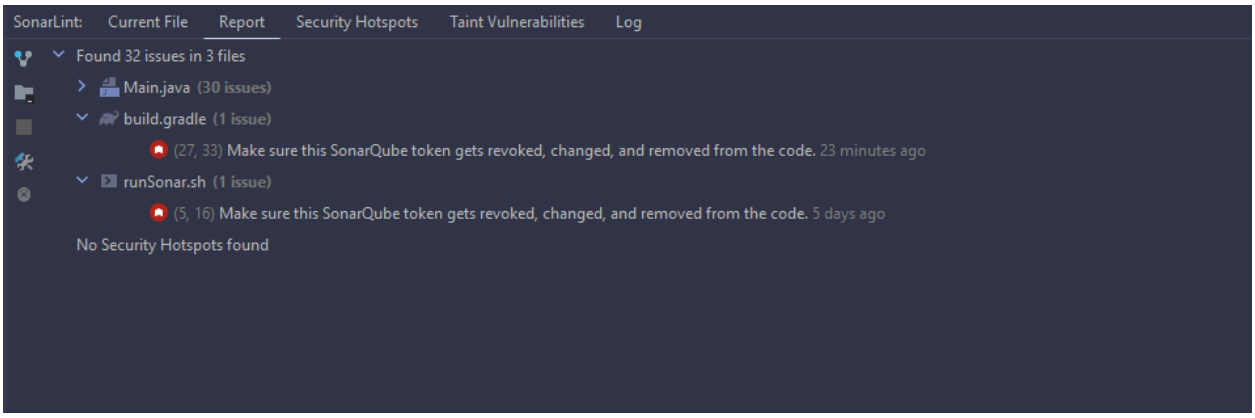

Figura 2: Reporte de Sonar Link

```java
public boolean createTask(Integer id, String title, String description) {
    Task task = new Task(id, title, description);
    tasks.put(id, task);
    return true;
}

1 usage  Jonatancon
public Task getTask(Integer id) { return tasks.get(id); }

1 usage  Jonatancon
public boolean completeTask(Integer id) {
    Task task = tasks.get(id);
    task.setStatus(Status.CLOSE);
    completeTask.add(task);
    tasks.remove(id);

    return true;
}

1 usage  Jonatancon
public boolean startTask(Integer id) {
    tasks.get(id).setStatus(Status.PROGRESS);
    return true;
}

1 usage  Jonatancon
public boolean deleteTask(Integer id) {
    tasks.remove(id);
    return true;
}
```

Figura 3: Fragmento de código de la implementación

También si miramos el código nos vamos a dar cuenta de que este no contiene ningún trato hacia los errores y en todos los casos siempre devuelve verdadero, lo cual es una mala práctica, dado que si se llega a presentar un error será complicado seguir la traza.

5. Solución de hallazgos

5.1. Diagrama de flujo

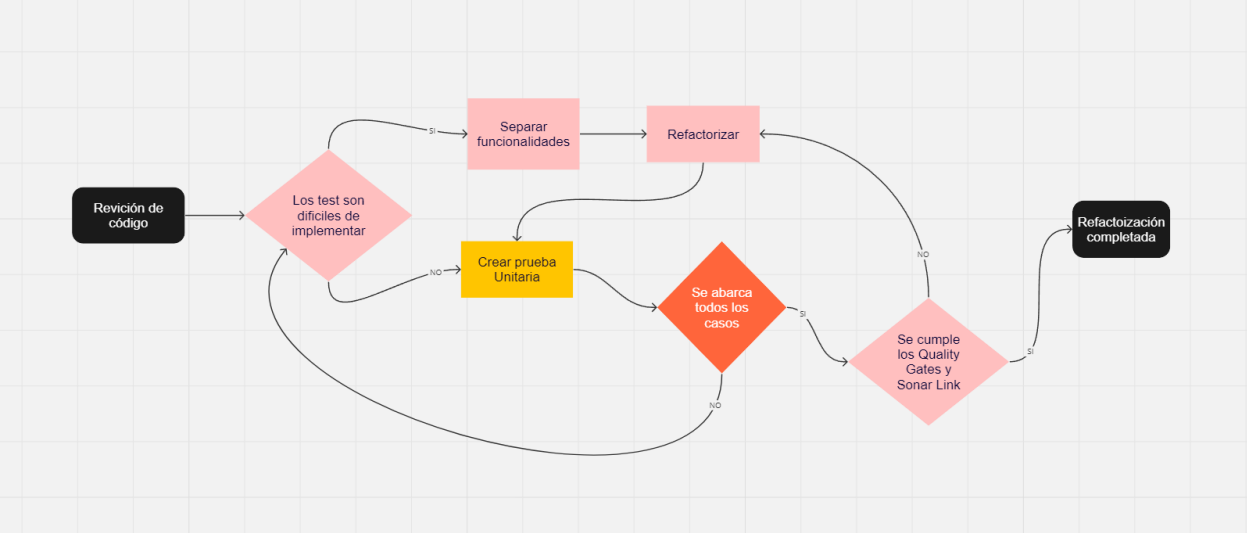

Figura 4: Flujo a seguir para llevar a cabo las correcciones.

6. Correcciones

Se ajustaron las correcciones que proponía SonarQube, Sonar Link. Además, se distribuyo los paquetes y desacoplo la lógica para poder realizar los test más fácilmente.

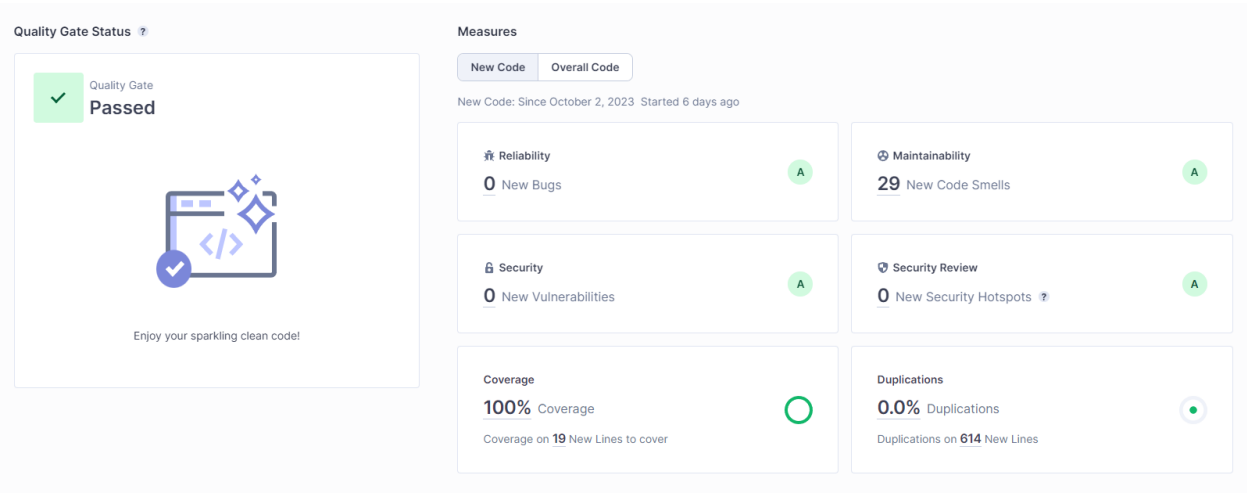

Figura 5: Todas las issues corregidas, SonarQube

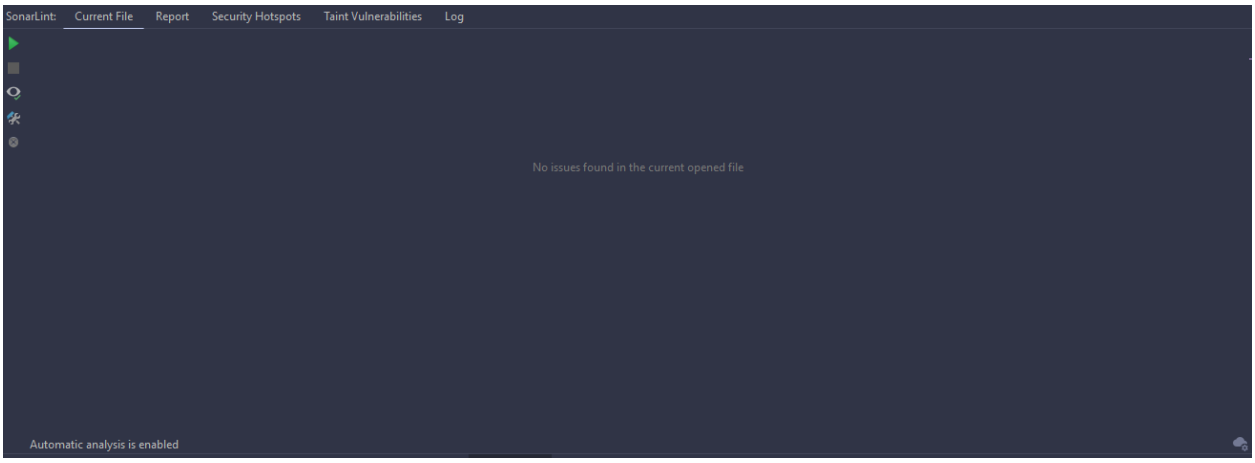

Figura 6: Todas las issues corregidas, Sonar Link

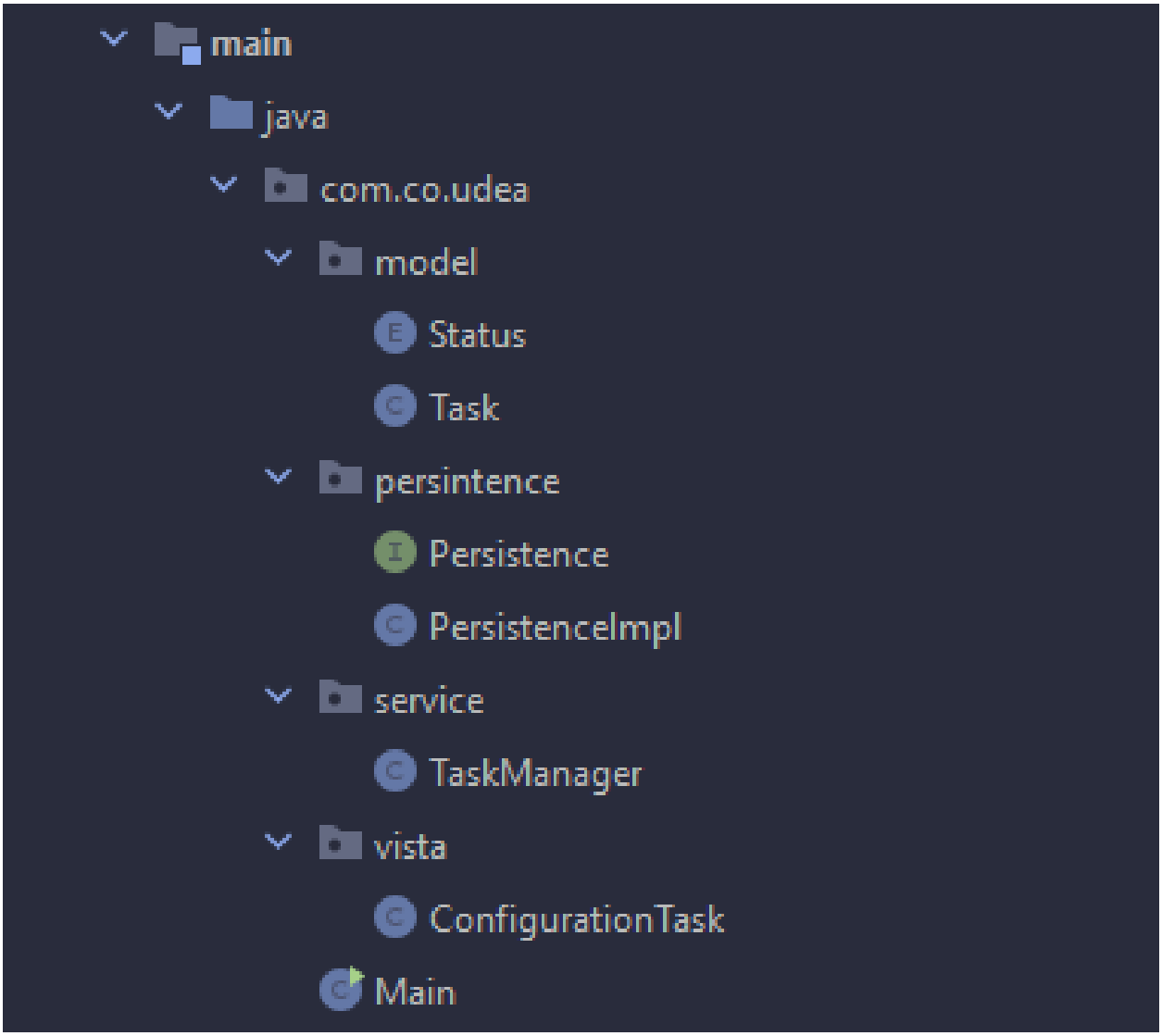

Figura 7: Se organizo la distribución de paquetes

Figura 8: Se mantuvo la complejidad ciclomática sobre los estándares establecidos

7. Concluciones

La deuda técnica se revela como un factor crítico en el desarrollo y la calidad del software. En esta práctica, se han cumplido los objetivos iniciales al refactorizar el proyecto y adaptarlo a los estándares establecidos, facilitando así su mantenimiento futuro. En resumen, el establecimiento de métricas y la consideración de la calidad contribuyen a la creación de un producto más confiable y menos propenso a errores.

8. Bibliografía

- Junit 5
- Jacoco
- SonarQube
- SonarLink

9. Proyecto - Github

Repositorio