REFACTORING

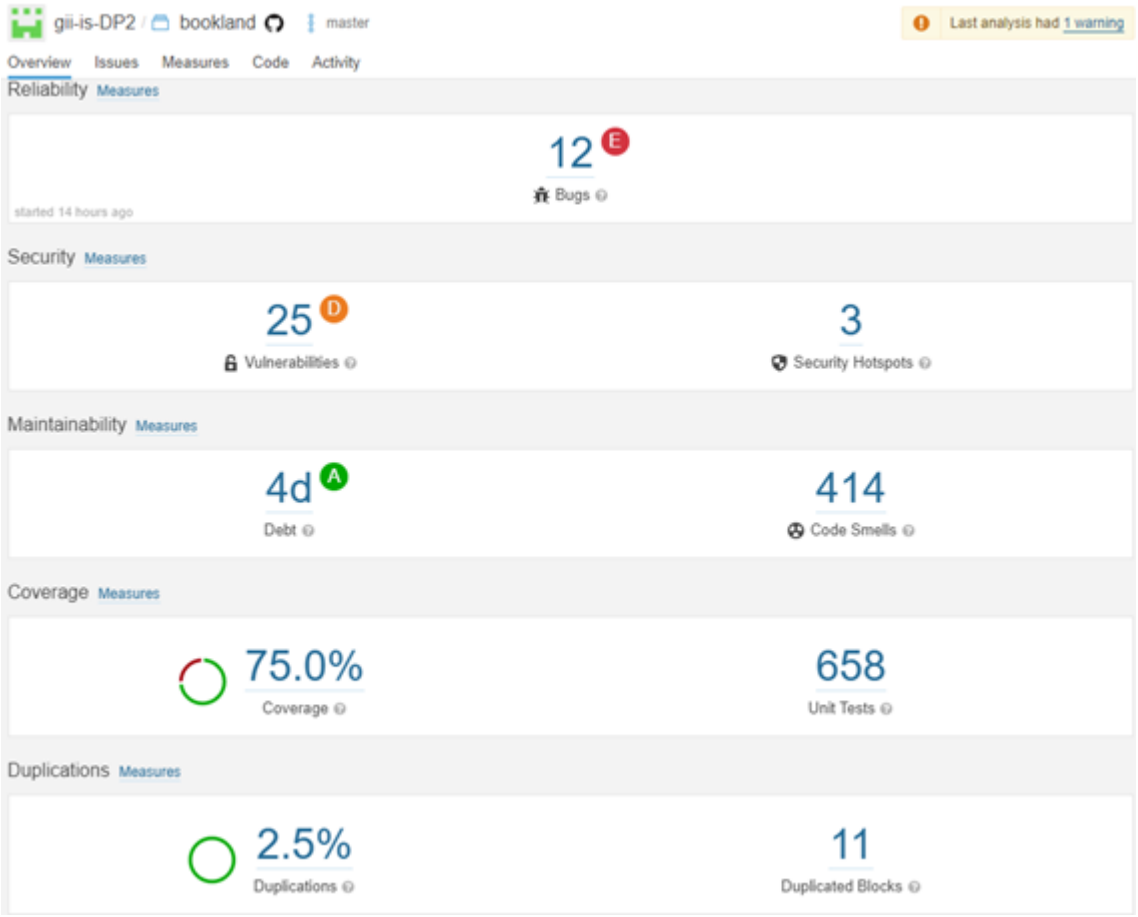
DP2-1920-G3-05

JUAN FERNÁNDEZ FERNÁNDEZ
ESTEFANÍA GANFORNINA TRIGUERO
JAVIER GARCÍA CERRADA
FRANCISCO PEREJÓN BARRIOS
FERNANDO ROMERO RIOJA

Contenidos

Análisis inicial SonarQube 2
Refactorizaciones Juanfer 3
Refactorizaciones Estefanía 4
Refactorizaciones Francisco 5
Refactorizaciones Javier 6
Refactorizaciones Fernando 7
Análisis final SonarQube 8

Análisis inicial SonarQube



Aquí se puede observar el análisis realizado por Sonar Qube antes de realizar ninguna refactorización.

Podemos observar un número elevado de malos olores acumulados a lo largo del desarrollo del proyecto, pero en general los resultados son aceptables ya que el análisis está aprobado.

Refactorizaciones Juan

Los malos olores detectados son:

- String literals should not be duplicated
- Boolean literals should not be redundant
- Boxed Booleans should be avoided

Para corregir el primer tipo de mal olor, el cual se trata de un change preventer, lo que se hace es crear una constante para cada String que aparece varias veces y sustituir cada aparición de ese String por la constante, aumentando así la mantenibilidad del código. El patrón de refactorización utilizado sería entonces el de extract constant.

En el segundo tipo de mal olor, vemos que se trata de un dispensable, ya que se trata de código redundante y eliminarlo aumentaría la legibilidad y la mantenibilidad del código. El proceso que se ha seguido para corregir este error ha sido básicamente sustituir aquellas expresiones booleanas del tipo booleano==true o booleano==false por booleano o !booleano respectivamente. El patrón de refactorización utilizado es simplifying conditional expressions.

El tercer tipo de mal olor, que se trata de un object orientation abuser, ya que no se está teniendo en cuenta que el Boolean pueda ser null y salte una NullPointerException al evaluar la expresión en un if por ejemplo. Para lidiar con esto, lo que se ha hecho es utilizar Boolean.TRUE.equals(booleano) o Boolean.FALSE.equals(booleano) según el caso, en vez de poner solo el objeto Boolean. En este caso al tratarse de un fallo menor, lo realizado no se corresponde con ningún patrón de refactorización específico.

Refactorizaciones Estefanía

Los malos olores solucionados son:

- Remove the declaration of thrown exception 'org.springframework.dao.DataAccessException'.
- Remove unused import.
- Sections of code should not be commented out.
- Dead code.

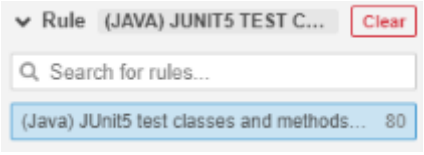
Todos estos malos olores se tratan de dispensables. Cómo se trata de código innecesario, la solución ha sido eliminar las declaraciones de excepción lanzadas 'org.springframework.dao.DataAccessException', las secciones de código comentadas y todas las clases y métodos obsoletos.

Refactorizaciones Francisco José

El mal olor arreglado es una mala práctica que se ha repetido mucho durante todo el proyecto y es el siguiente:

- JUnit5 test classes and methods should have default package visibility

Este mal olor se basa en poner un modificador a la clase y métodos de Junit5, cuando lo recomendado es dejar la visibilidad por defecto.



como se puede ver este error se repite 80 veces durante todo el proyecto. La solución se ha basado en borrar todos estos modificadores.

Refactorizaciones Javier

Los malos olores a solucionar son los siguientes:

- Assertions should be complete.
- Replace this “if-then-else” statement by a single method invocation.
- Replace this “if-then-else” statement by a single return statement.
- Method should not be too complex.

El primer mal olor está relacionado con las pruebas. Es fácil escribir aserciones incompletas, especialmente usando un framework con grandes posibilidades como AssertJ. Por tanto, en las refactorizaciones arreglaremos estos problemas y seguiremos las buenas prácticas vistas en clase para usar AssertJ.

El segundo olor se debe a usar un if para evaluar una condición y en función de la misma asignar a un atributo el valor true o false. Para corregirlo podemos asignar directamente al atributo la expresión booleana.Este mal olor lo podríamos considerar un bloaster ya que hace los métodos más largos y además innecesariamente, por lo que también podríamos catalogarlo de dispensable. En cuanto al patrón de refactorización que emplearemos para solucionarlo, será el de simplificar las expresiones condicionales.

El tercer olor está muy relacionado con el segundo, y se trata de evitar hacer un if-else evaluando una condición y devolver true o false, cuando directamente podemos devolver la expresión booleana. Catalogaría este olor de dispensable ya que es redundante y su eliminación simplificará el código y reducirá su extensión. En cuanto a la estrategia de refactorización, como en el caso anterior, será la de simplificar los condicionales.

El tercer olor se debía a que un método tenía una complejidad ciclómatica muy elevada debido a que es bastante largo y con muchas condiciones (método deleteById de BookService.java) por lo que lo consideramos un bloaster. Para solucionarlo hemos seguido el patrón de refactorización de extraer método, llamando a varios métodos auxiliares lo que facilita la mantenibilidad y la entendibilidad.

Refactorizaciones Fernando

Los malos olores que se van a resolver son:

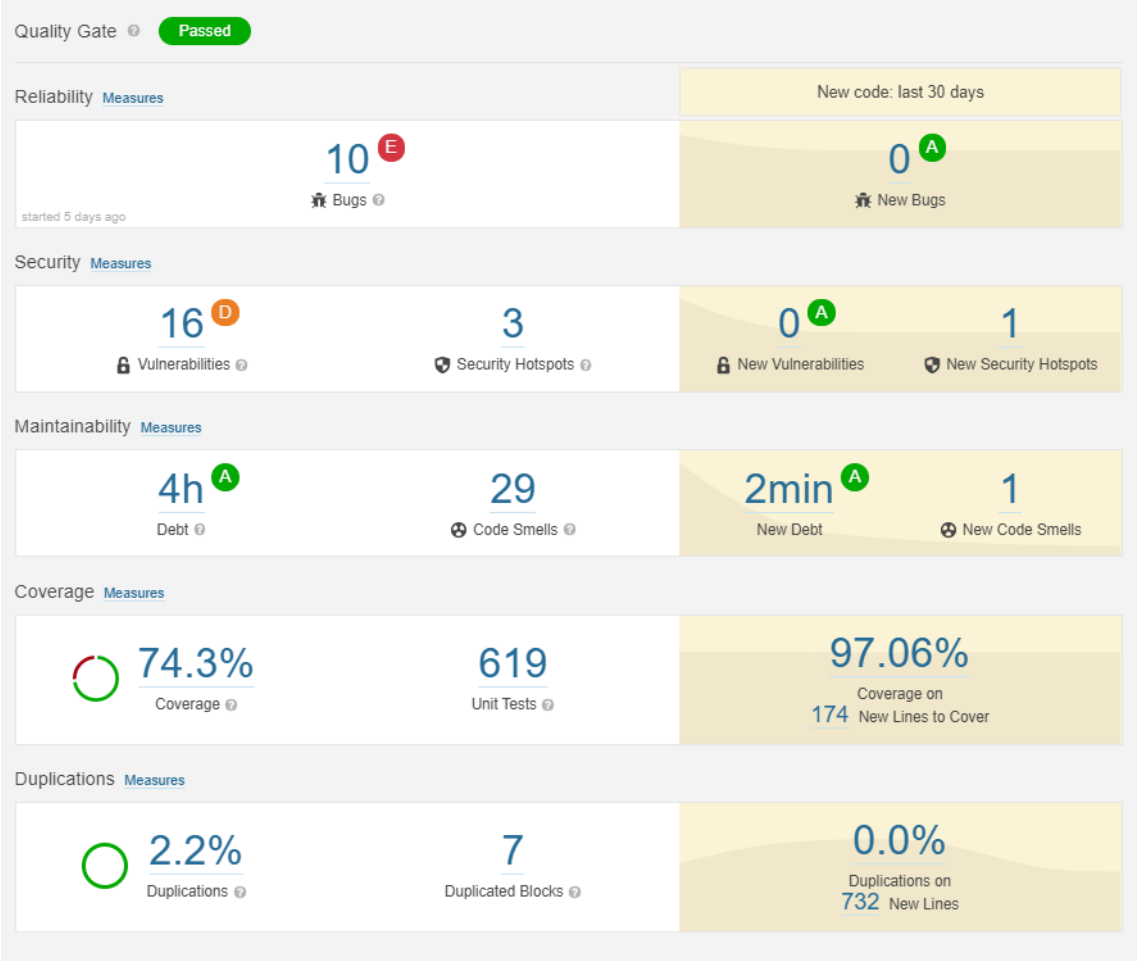
- Add at least one assertion to test case
- Replace the type specification in this constructor call with the diamond operator ("<>")
- Duplicated blocks of code must been removed

El primer mal olor está directamente relacionado con las pruebas realizadas, este caso es casi un error en lugar de un mal olor, ya que al no haber ninguna aserción el test no hacía nada. La solución para este mal olor ha sido identificar la prueba en la que faltaba la aserción y añadirla.

El segundo mal olor se trata de un dispensable, es decir código innecesario. Para solucionar este mal olor simplemente se han borrado las especificaciones que se encontraba entre <> en los métodos indicados.

El tercer mal olor se trata de una duplicación de código. La solución proporcionada para resolver este mal olor ha sido poner el método validateISBN de la clase BookValidator.java, como public static, y después llamarlo en la clase ItApiService.java, y para el otro caso de código duplicado simplemente se extrajo la parte duplicada se creó un metodo con esa parte y luego se llamó a dicho método en ambos métodos los cuales tenían el mismo código. Esto se encuentra en la clase NewController.java

Análisis final SonarQube



Se han resuelto gran parte de los malos olores como se puede apreciar.

Se han empleado 10 horas y 23 minutos, mientras que según Sonar Qube deberíamos haber empleado en torno a 28 horas. Por tanto, hemos tenido un buen rendimiento.