

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

## Análisis del Código Fuente y Métricas




Grado en Ingeniería Informática – Ingeniería del Software

Proceso de Software y Gestión 2

Curso 2022 – 2023


Fecha	Versión
12/03/2023	1.0

Grupo de prácticas: G4-43		
Autores por orden alfabético	Rol	Descripción del rol
Alcobendas Santos, Jose Javier - 53986326J	Developer	Miembro del equipo de desarrollo
Campos Garrido, Juan Jesús - 47547107N	Scrum Master	Encargado de facilitar la labor de Scrum
Nunes Ruiz, Javier - 29517615J	Developer	Miembro del equipo de desarrollo
Pizarro López, Eduardo - 77933507P	Developer	Miembro del equipo de desarrollo
Reyes Alés, David - 29504757N	Developer	Miembro del equipo de desarrollo

	Proceso de Software y Gestión 2
	<b>Control de Versiones</b>


### Control de Versiones

Fecha	Versión	Descripción
12/03/2023	0.1	Creación del documento
21/03/2023	1.0	Versión final
29/08/2023	1.1	Corrección de errores

	<div>Proceso de Software y Gestión 2</div>
---	--

## Índice de contenido

<b>1. Introducción</b>	<b>2</b>
<b>2. Objetivo</b>	<b>2</b>
<b>3. Contenido</b>	<b>2</b>
<b>3.1 Captura de las métricas</b>	<b>2</b>
<b>3.2 Descripción de los bugs potenciales</b>	<b>4</b>
<b>3.3 Descripción de los code smells</b>	<b>5</b>
<b>3.4 Conclusiones</b>	<b>10</b>

	<div>Proceso de Software y Gestión 2</div>
---	--

## 1. Introducción

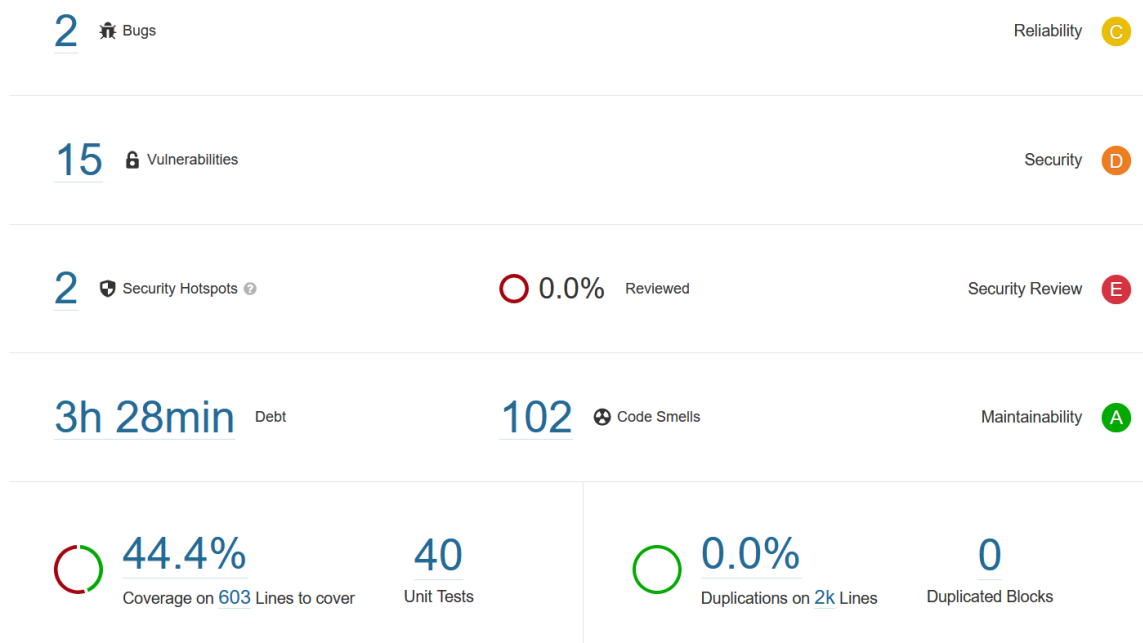
En los procesos software, una de las cuestiones más importantes es la calidad del código, si queremos que nuestro proyecto tenga éxito, tenemos que intentar que nuestro código tenga la máxima calidad posible, para eso se definen métricas como los bugs potenciales o los olores del código.


## 2. Objetivo

En este caso en cuestión, nosotros hemos utilizado SonaQube para automatizar la medición de estas métricas y en este documento, las comentaremos.

## 3. Contenido

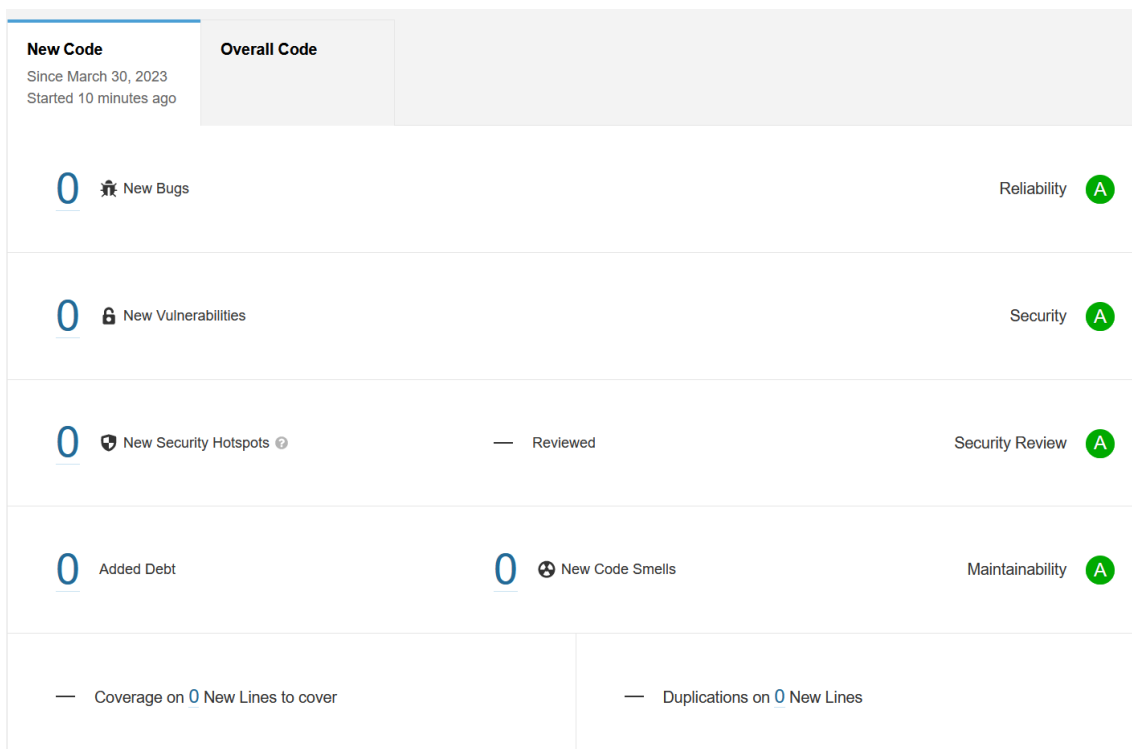
### 3.1 Captura de las métricas




	<div>Proceso de Software y Gestión 2</div>
---	--

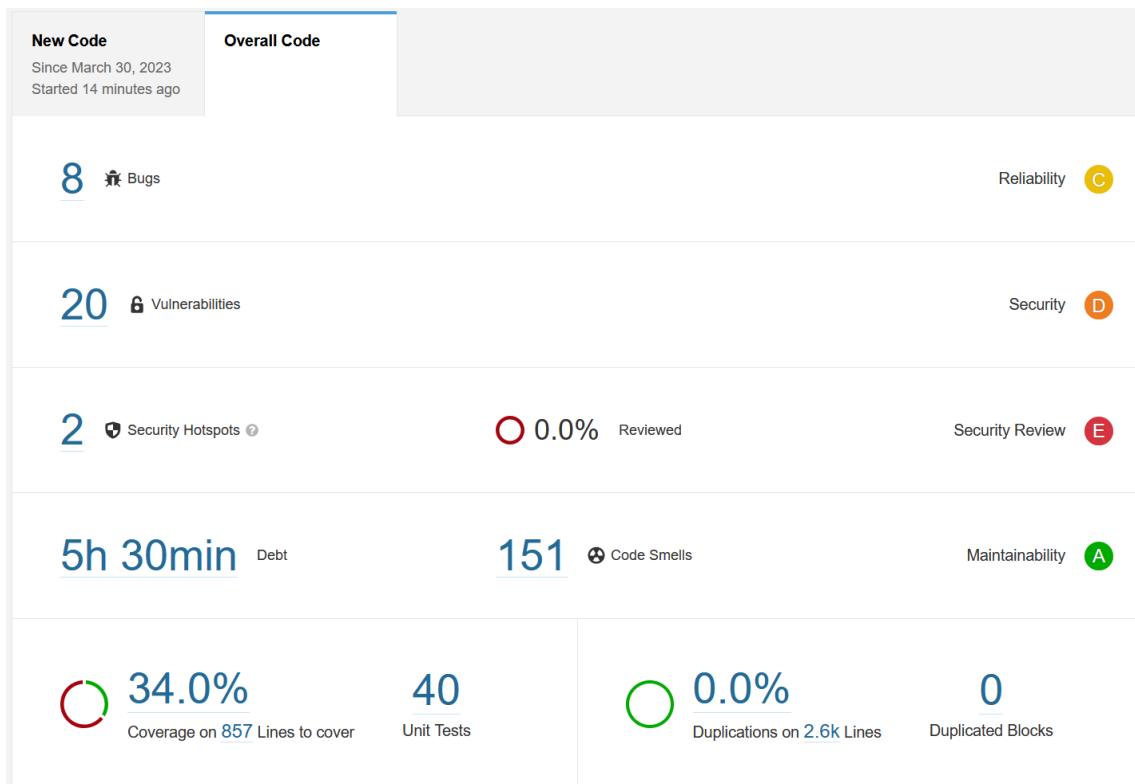
Dicho esto, lo que podemos observar en esta captura de las métricas es que tenemos se cuenta el número de bug potenciales que son errores que en un futuro podrían producir algún fallo o comportamiento inesperado en la aplicación. También podemos observar las vulnerabilidades que son debilidades que pueden comprometer la seguridad de la aplicación, además de dos security hotspots que son posibles vulnerabilidades que deben ser revisadas manualmente para determinar si realmente comprometen la seguridad de la aplicación. Por otro lado, podemos observar los code Smells, que son olores que nos indican que puede haber algo mal en el código o que se están incumpliendo principios y las cosas no se están haciendo de la forma correcta, además, SonarQube, nos indica el tiempo de tardaremos en corregir estos Code Smells. Además, podemos ver el porcentaje de código que tiene test que comprueben que este se comporta como debe.

Antes de comentar las métricas del tercer sprint, queremos comentar que parece ser que hemos cometido algún fallo y por alguna razón al subir el código al SonarQube del tercer sprint, este no ha hecho el resumen del nuevo código y se ve de la siguiente manera el resumen.



Así que pondremos la captura del código general y tendremos que compararla con la del sprint anterior.


	<div>Proceso de Software y Gestión 2</div>
---	--



Podemos ver que han aparecido 6 nuevos bugs y 5 vulnerabilidades nuevas, los security hotspots no han aumentado, la deuda técnica ha aumentado en 2 horas y dos minutos, los code smells han aumentado en 49 más y el porcentaje de código probado ha descendido un 10,4%, que es normal ya que hemos aumentado la cantidad de código pero no la cantidad de test.

### 3.2 Descripción de los bugs potenciales


- 1) Use the "equals" method if value comparison was intended: este bug nos recuerda que, si se quiere comparar dos datos que se desean o esperan iguales, usar el método `.equals()`. A simple vista se puede ver que en el método `savePet`, se necesita una desigualdad (`!=`), habría que investigar este bug. Se detectó el anterior mes, y se estiman 5 minutos de coste para reparar el bug. (Sprint 2)
- 2) Este bug es el mismo detectado en el anterior (1), pero en el método `getPetwithIdDiffetent`. Se detectó el mes pasado, y se estiman 5 minutos de coste para reparar el bug. (Sprint 2)
- 3) Call "Optional#isPresent()" before accessing the value: este bug nos avisa que si queremos acceder a un valor opcional, primero comprobemos que existe con el método `isPresent()`. Se refiere al método `saveNewCause`. Se detectó ayer, y se estiman 10 minutos de coste para reparar el bug. (Sprint 3)

	<div>Proceso de Software y Gestión 2</div>
---	--

- 4) Este bug es el mismo detectado en el anterior (3), pero en el método editCause. Se detectó ayer, y se estiman 10 minutos de coste para reparar el bug. (Sprint 3)
- 5) Este bug es el mismo detectado en el anterior (3), pero en el método causeDetails. Se detectó ayer, y se estiman 10 minutos de coste para reparar el bug. (Sprint 3)
- 6) A "NullPointerException" could be thrown; "adoptionRequest" is nullable here: se puede observar que nos indica que puede que haya una posible excepción de que lance una excepción debido a que adoptionRequest podría ser null en deleteAdoptionResponse. Se detectó hace 3 días, y se estiman 10 minutos de coste para reparar el bug. (Sprint 3)
- 7) Este bug es el mismo detectado en el anterior (6), pero en el método adoptionRequestToChange, en selectAdoptionResponse. Se detectó hace 3 días, y se estiman 10 minutos de coste para reparar el bug.(Sprint 3)
- 8) Este bug es el mismo detectado en el anterior (6),en el mismo método adoptionRequest pero en createAdoptionResponse. Se detectó hace 4 días, y se estiman 10 minutos de coste para reparar el bug.(Sprint 3)

### 3.3 Descripción de los code smells

1. Rename this package name to match the regular expression `'^[a-z_]+(\\.[a-z_][a-z0-9_]*)*$'`.  
El problema es el uso de camelcase en el nombre del paquete, las consecuencias de esto son minúsculas ya que al ser dos palabras por convención se usa el camel case.  
La solución sería seguir otra clase de convención como unir las palabras en minúsculas por una barra baja. Este code smell se da más veces a lo largo del código pero como en nuestros estándares decidimos utilizar camelcase es normal que ocurra.  
Este code smell aparece en ambos Sprints
2. Declare this local variable with "var" instead.(ObjOr Abusers)  
El problema es poner el tipo de la variable en vez de poner "var", las consecuencias de esto no son significativas pues muchas veces el poner var no es posible; hay que mencionar que este code smell salta con las clases que no son de java.  
La solución sería cambiar todos los code smells de este tipo por "var".  
Este code smell aparece en ambos Sprints.

	<div>Proceso de Software y Gestión 2</div>
---	--

3. Define a constant instead of duplicating this literal "admin" 5 times. (DISPENSABLES)  
Hay ciertos casos donde sería conveniente definir una variable para no tener que repetir la declaración explícita de un argumento que se vuelve a encontrar seguidamente en varias ocasiones por el uso consecutivo de algunos métodos específicos.  
Este code smell aparece en ambos Sprints.
  
4. Remove this unused method parameter "principal" (DISPENSABLES)  
El problema es el uso del parámetro "principal" cuando no es usado en la función.  
Podría causar problemas en el futuro por la mantenibilidad del código.  
La solución es tan simple como borrarlo.  
Este code smell aparece en el segundo Sprint.
  
5. Use the primitive boolean expression here.(ObjOr Abusers)  
El problema es cuando el booleano es null que no se comprueba.  
Podría causar problemas en caso de que el booleano tome valor null y al no tenerlo en cuenta no gestionarlo.  
La solución sería evaluar el booleano así:  

```
Boolean b = getBoolean();
if (Boolean.TRUE.equals(b)) {
    foo();
} else {
    bar(); // will be invoked for both b == false and b == null
}
```

Este code smell aparece en el segundo Sprint.
  
6. Remove the unnecessary boolean literal.(ObjOr Abusers)  
El problema es que los literales booleanos redundantes deberían ser eliminados de las expresiones para mejorar la legibilidad, sin embargo un daño muy pequeño.  
La solución sería cambiar el booleano por esto:  

```
if (booleanMethod()) { /* ... */ }
if (!booleanMethod()) { /* ... */ }
if (booleanMethod()) { /* ... */ }
doSomething(true);
doSomething(booleanMethod());
```



```
booleanVariable = booleanMethod();  
booleanVariable = booleanMethod() || exp;  
booleanVariable = !booleanMethod() && exp;  
booleanVariable = !booleanMethod() || exp;  
booleanVariable = booleanMethod() && exp;
```

Este code Smell aparece en ambos sprints.

7. Make description a static final constant or non-public and provide accessors if needed.(ObjOr Abusers)

El problema es no poner el String como private ya que los campos variables de clase pública no respetan el principio de encapsulación y tienen tres desventajas principales:


- No se pueden agregar comportamientos adicionales, como la validación.
- La representación interna está expuesta y no se puede cambiar después.
- Los valores de los miembros están sujetos a cambios en cualquier parte del código y es posible que no cumplan con las suposiciones del programador.

Mediante el uso de atributos privados y métodos de acceso (establecer y obtener), se evitan las modificaciones no autorizadas.

Este code Smell aparece en ambos sprints.

8. Define a constant instead of duplicating this literal "adoptionRequest" 3 times.(DISPENSABLES)

El problema es que los literales de cadena duplicados hacen que el proceso de refactorización sea propenso a errores, ya que debe asegurarse de actualizar todas las ocurrencias.

	<div>Proceso de Software y Gestión 2</div>
---	--

Se solucionaría asignando a una variable para referenciar.

Este code Smell aparece en ambos sprints.

9. Local variables should not be declared and then immediately returned or thrown.(ObjOr Abusers)

El problema es que declarar una variable solo para devolverla o tirarla inmediatamente es una mala práctica.

La solución es devolverla directamente, la gravedad de esto es menor.

Este code Smell aparece en ambos sprints.

10. Cognitive Complexity of methods should not be too high.(Bloaters)

La complejidad cognitiva es una medida de cuán difícil es comprender el flujo de control de un método. Los métodos con alta complejidad cognitiva serán difíciles de mantener.

La solución sería refactorizar el código de alguna manera para que se entienda mucho mejor, esto se considera un code smell grave.


Este code Smell aparece en el segundo sprint.

11. Replace the type specification in this constructor call with the diamond operator ("<>").(ObjOr Abusers)

Java 7 introdujo el operador de diamante (<>) para reducir la verbosidad del código genérico. Por ejemplo, en lugar de tener que declarar el tipo de una Lista tanto en su declaración como en su constructor, ahora puede simplificar la declaración del construct

or con <>, y el compilador inferirá el tipo.

La gravedad de esto es menor.

	<div>Proceso de Software y Gestión 2</div>
---	--

La solución sería cambiarlo a:

```
List<String> strings = new ArrayList<>();
```

```
Map<String,List<Integer>> map = new HashMap<>();
```

Este code Smell aparece en ambos sprint.

#### 12. Remove this unused import

```
'org.springframework.http.ResponseEntity'.(DISPENSABLES)
```

El problema es el importe de paquetes que no son usados en esa clase, la gravedad de esto es menor.

La solución es borrarlo directamente.

A lo largo de ambos sprint, este es uno de los code smells que mas veces se ha repetido

#### 13. Remove this use of "NoOpPasswordEncoder"; it is deprecated.(DISPENSABLES)

El problema es que esta clase está desfasada, la gravedad de esto es menor.

La solución es usar la variante que sea más actual.

Este code Smell es del sprint1, pero realmente es un problema de la aplicación base ya que nosotros no hemos tocado el encoder.

### 3.4 Conclusiones

En conclusión, es bastante normal que cometamos este tipo de errores



debido a nuestra corta experiencia, pero estamos bastante contentos viendo que nuestros fallos no son demasiado graves y además los podemos solucionar sin demasiados problemas. Sin embargo, hay ciertos code smells que no sabemos si deberíamos corregir ya que hay alguno, como por ejemplo el nombrado de los paquetes usando camelCase, que SonarQube detecta como code smells pero nosotros en nuestros estándares definimos usar este formato en base a nuestra experiencia y lo que nos han ido enseñando nuestros profesores.