Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Grupo G2-23

Análisis de código fuente y métricas asociadas

Grado en Ingeniería Informática – Ingeniería del Software

Proceso de Software y Gestión 2

Curso 2020 – 2021

Fecha	Versión
<03/05/2021>	V01r01

Control de Versiones

Fecha	Versión	Descripción
<03/05/2021>	v01r01	Versión inicial

Índice

I.1. Análisis de código entrega S2

I.1.1. Descripción general

En el dashboard de SonarCloud podemos ver un resumen general del estado del proyecto en el momento de la entrega del S2:

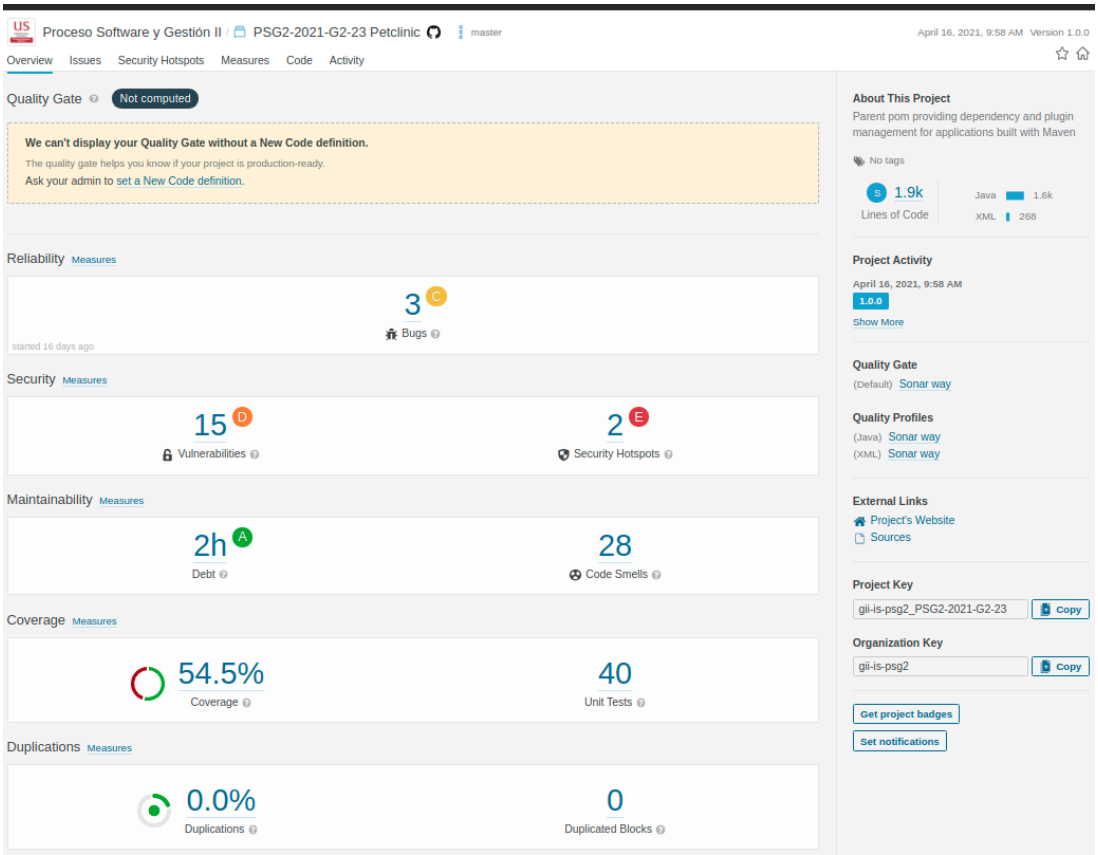

Fig 1: Dashboard de SonarCloud tras la entrega S2

Muestra 3 posibles bugs, varias vulnerabilidades de seguridad y una estimación del coste en tiempo de resolver la deuda técnica basándose en los olores (code smells) detectados.

A tener en cuenta que los tests sólo cubren en torno a la mitad del proyecto (54,5%) por lo que la calidad del mismo puede empezar a resentirse.

I.1.2. Análisis de potenciales bugs

En la sección *Reliability* se muestra el resultado de un análisis de posibles Bugs y una estimación de tiempo necesario para ser corregido cada uno de ellos:

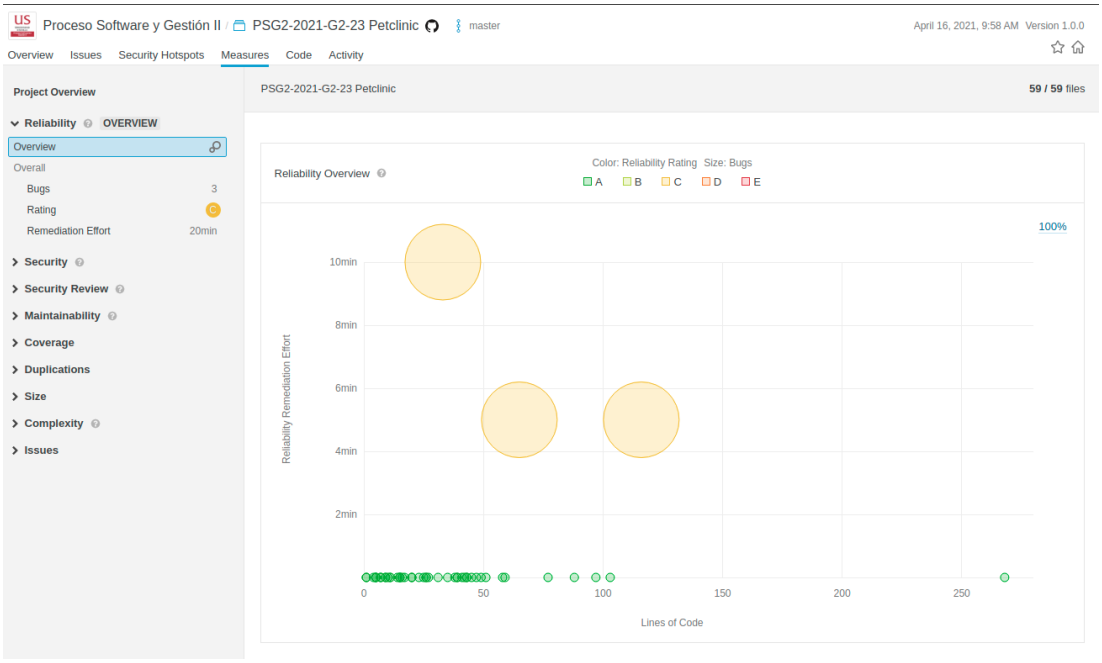

Fig 2: Posibles bugs y estimación de resolución de cada uno de ellos

Se muestra la detección de tres posibles bugs (errores que sólo se producen en determinadas circunstancias y no están siendo controlados por el código desarrollado), estimándose 10 minutos en la resolución de uno de ellos y 5 minutos en la resolución de cada uno de los otros dos.

I.1.3. Olores (code smells)

En la sección *Maintainability* se evaluan los riesgos a largo plazo. Se hace una estimación de la deuda técnica acumulada por desvíos en las buenas prácticas en codificación. La situación de nuestro proyecto en el momento de la entrega del S2 es:

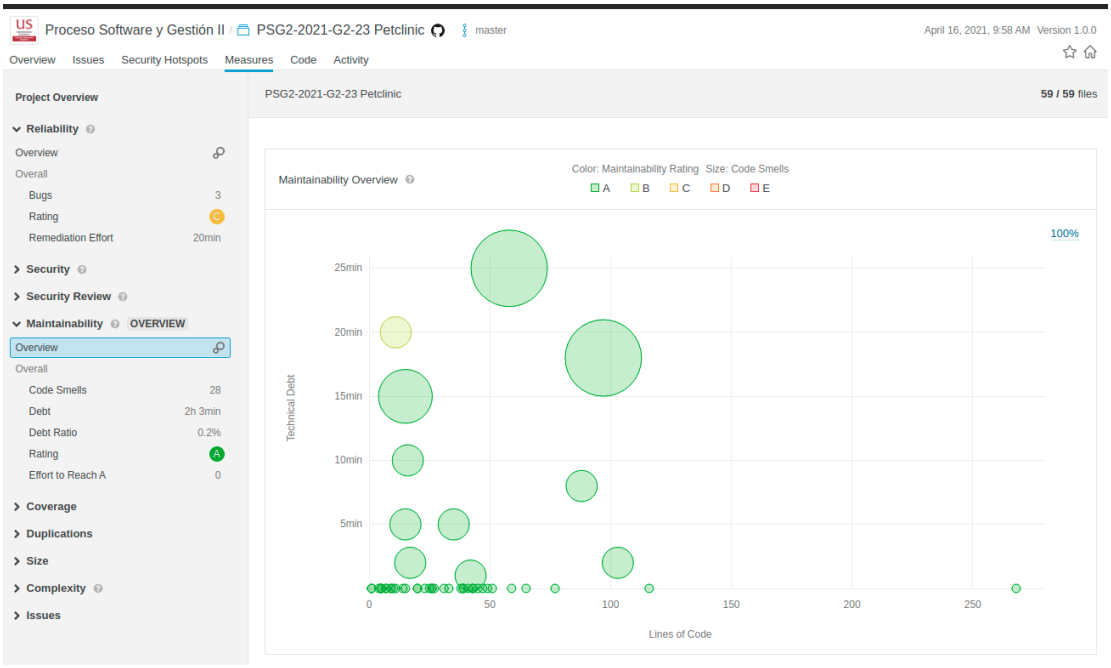

Fig 3: Code Smells tras la entrega S2

Han sido reconocidos 28 code smells, entre otros:

A) Change preventers: Cambiar algo en una parte del código implica cambios en otras partes.

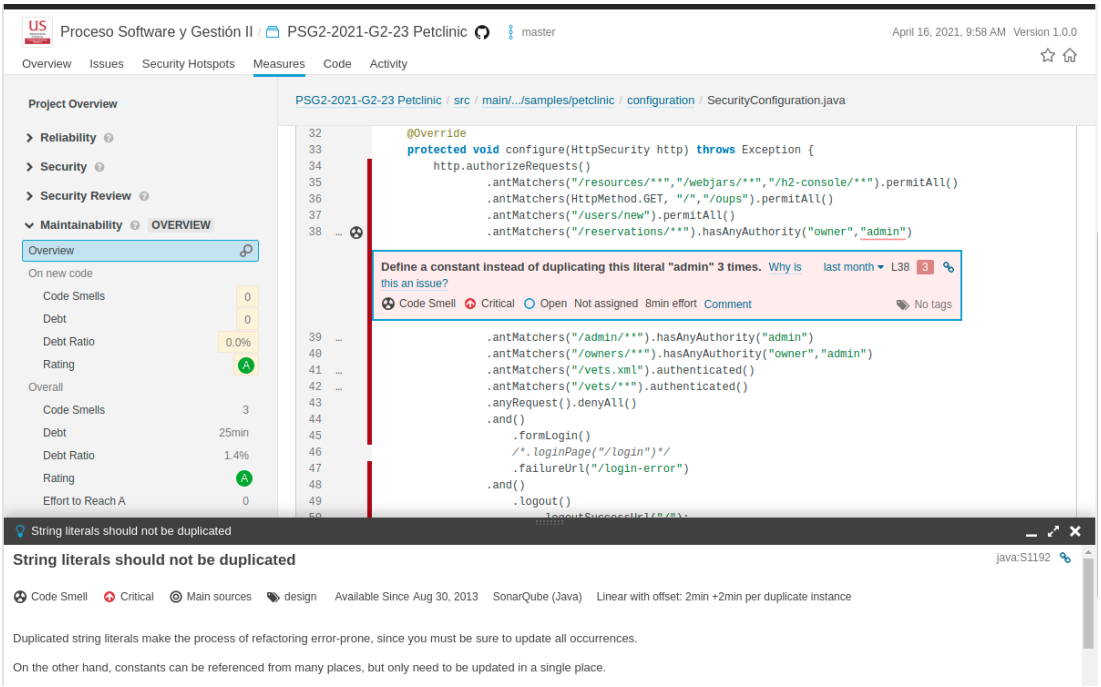

Fig 4: Change Preventer, varios usos del mismo literal

Se está usando un literal en lugar de usar una variable a la que asignar el valor de dicho literal.

La severidad asignada es crítica ya que cambiar el literal en un punto implica cambiarlo en otros. Cuantos más cambios más posibilidad de error y más lentitud en hacer efectivo dicho cambio.

La manera de resolver este problema es asignar el literal a una variable que se usará en todos los lugares del código en los que se está usando el literal.

B) Bloaters: Partes del código que llegan a crecer tanto que no pueden ser manejados de forma efectiva. En nuestra aplicación nos encontramos código comentado en lugar de ser eliminado:

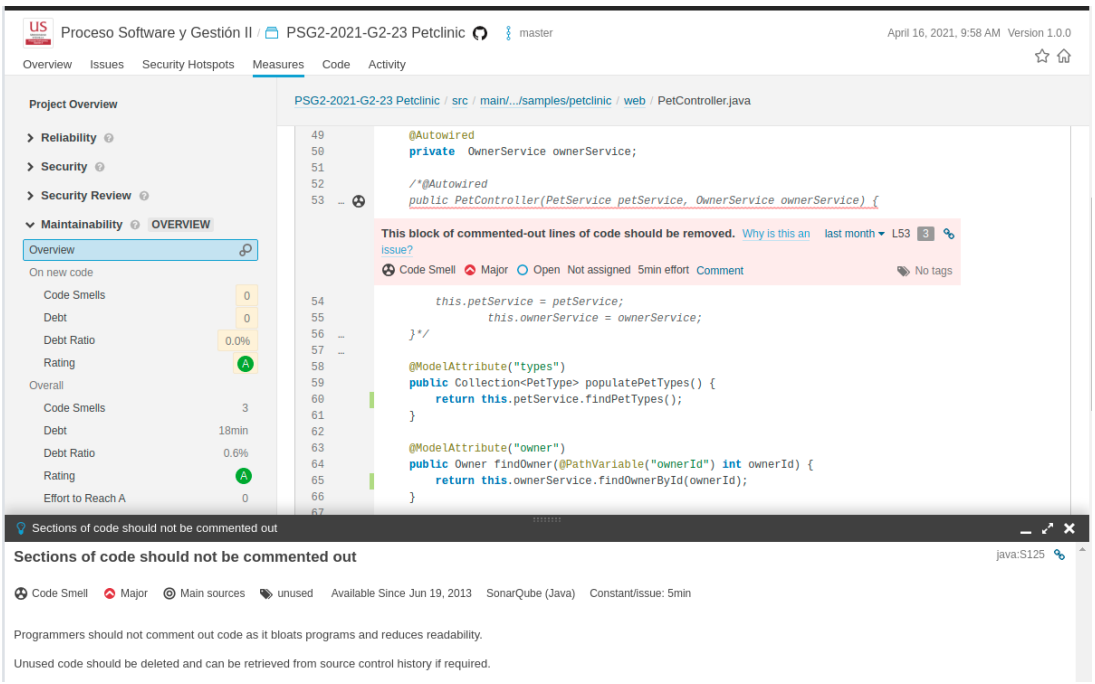

Fig 5: Código comentado que dificulta la legibilidad del código en uso.

Se ha comentado código en lugar de eliminarlo.

La severidad asignada es Major ya que dificulta la lectura del código que sí está en uso.

La forma de resolver este problema es eliminar las líneas, ya que no se pierden por poder ser “rescatadas” en versiones previas del código fuente.

C) Dispensables: Algo innecesario cuya ausencia podría hacer un código más eficiente y fácil de entender. Podemos encontrar en nuestro código una variable privada de una clase con el mismo nombre que la propia clase:

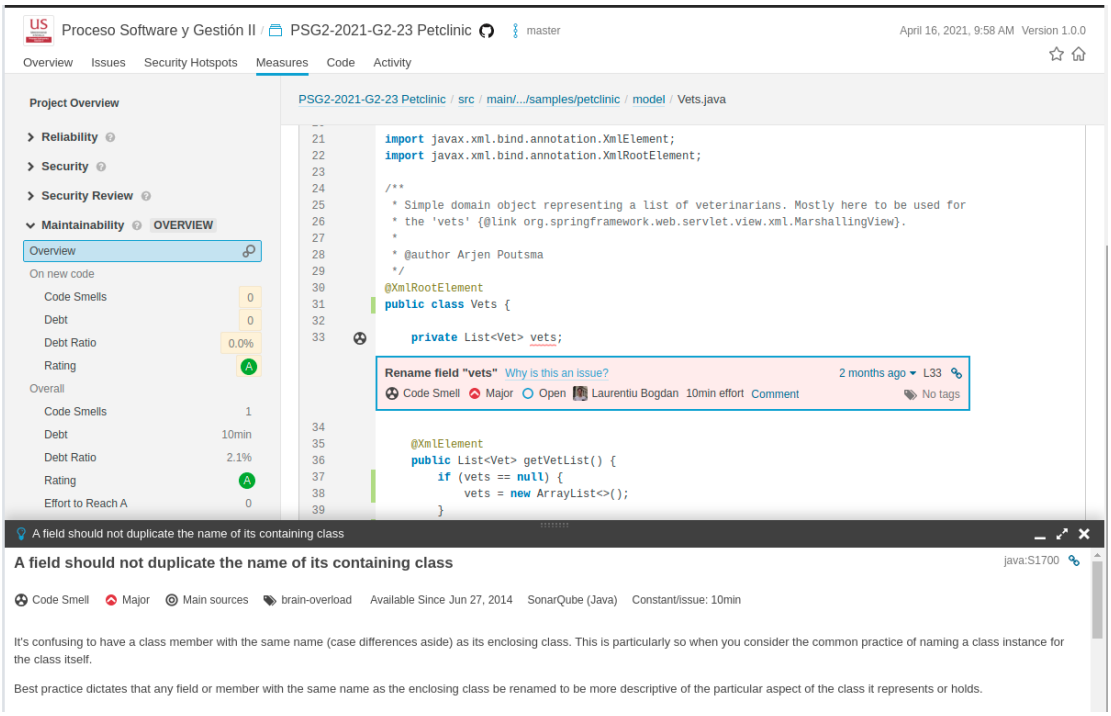

Fig 6: campo de una clase con el mismo nombre que la clase

Se ha puesto el mismo nombre a una clase y a una variable privada suya.

La severidad asignada es Major ya que la lectura del código se presta a confusión.

La solución de este problema consiste en cambiar el nombre de la variable.

Otro olor de tipo Dispensables es el mantener importaciones que no se usan:

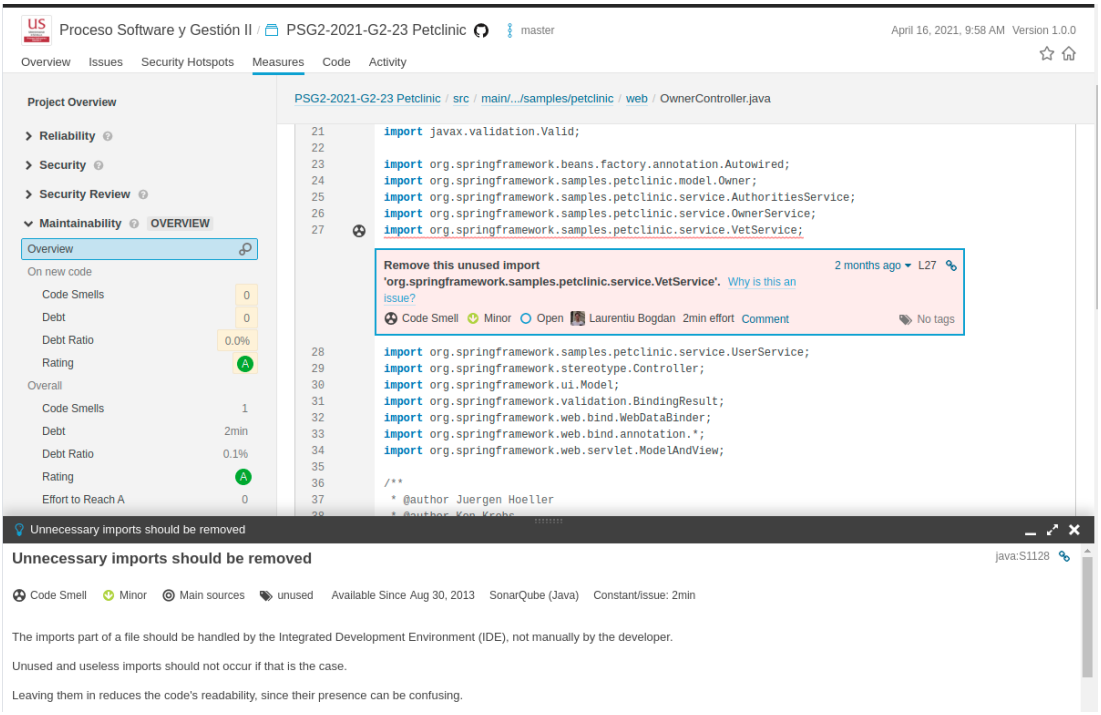

Fig 7: Import no usado

Se ha mantenido un Import a un paquete que no se usa en el código.

La severidad asignada es Minor ya que afecta poco a la lectura y comprensión del código.

La solución a este problema sería eliminar el Import.

I.2. Análisis de código entrega S3

I.2.1. Descripción general

Cuadro resumen del estado del proyecto en el momento de la entrega del S3:

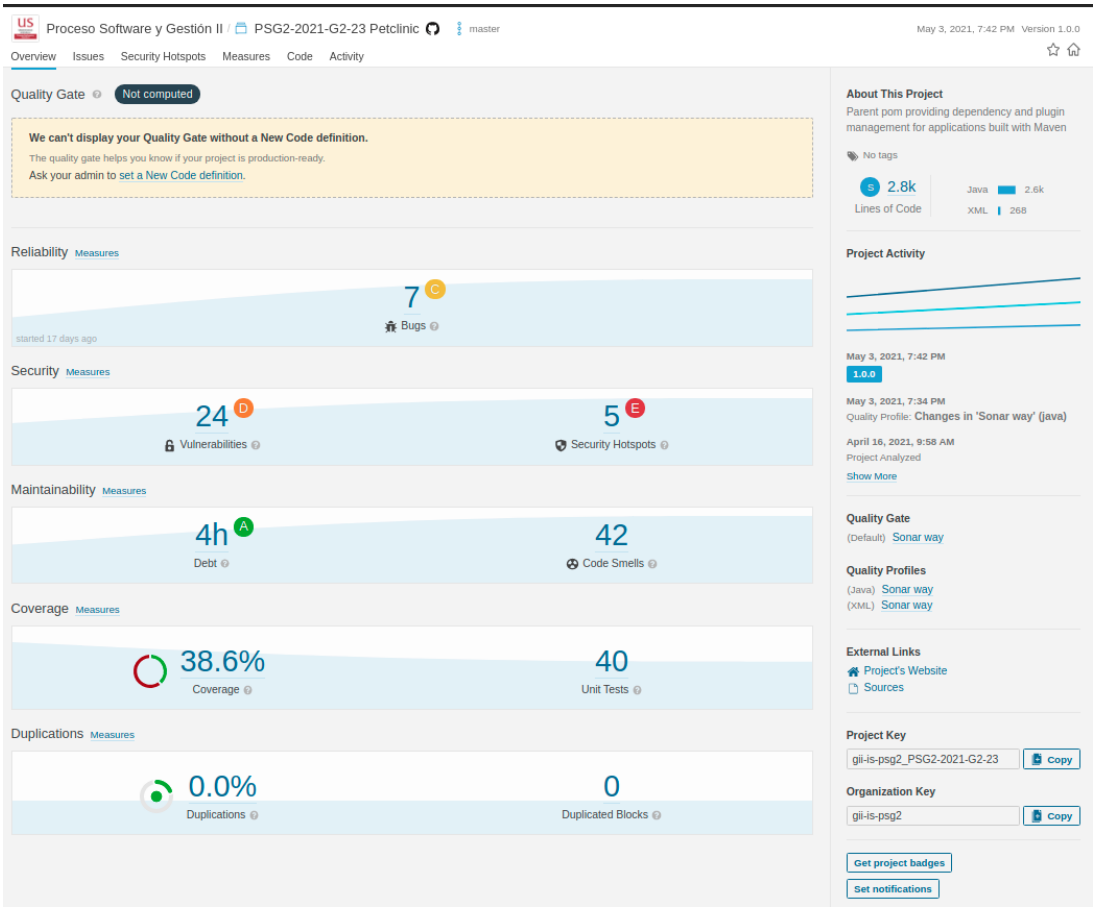

Fig 8: Dashboard S3

Muestra 7 posibles bugs, un incremento en vulnerabilidades de seguridad de aproximadamente un 40% respecto de S2 (de 15 a 24) y prácticamente se ha duplicado la estimación temporal necesaria para pagar la deuda técnica (de 2 horas con 28 code smells a 4 horas con 42 code smells).

Ya que no se han añadido nuevos tests pero sí ha aumentado la cantidad de código del proyecto el porcentaje cubierto por dichos tests ha disminuido (del 54,5% al 38,6% actual).

I.2.2. Análisis de potenciales bugs

El aumento de líneas de código ha llevado aparejado un aumento en el número de bugs y en el tiempo estimado en resolverlos:

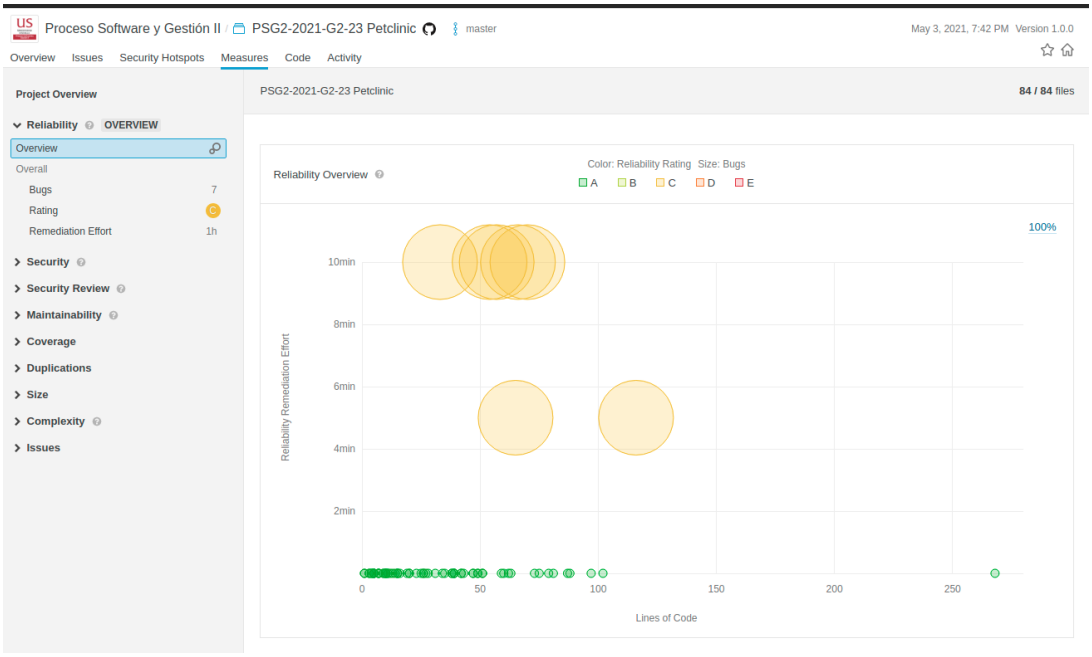

Fig 9: Posibles bugs y estimación de resolución de cada uno de ellos

Los bugs señalados son de la misma naturaleza que los preexistentes en S2, es decir, acceso a variables que podrían estar vacías pero no se está comprobando con IsEmpty() y comparaciones entre dos variable de sus posiciones de memoria en lugar de sus respectivos valores (se usa == o != en lugar del método equal()).

I.2.3. Olores (code smells)

Al no haberse corregido los code smells de S2 y al haberse desarrollado más código durante S3 el número de code smells ha subido casi hasta duplicarse:

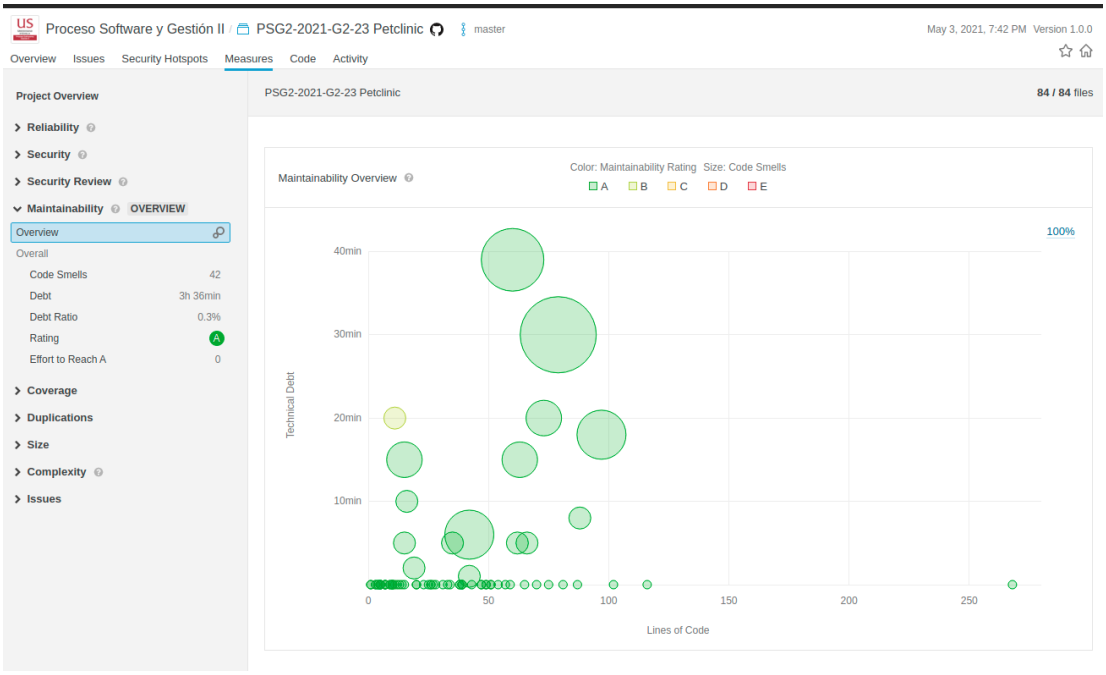

Fig 10: Code Smells tras la entrega S3

Algunos de los nuevos code smells surgidos en S3 son:

A) Dispensables: código innecesario.

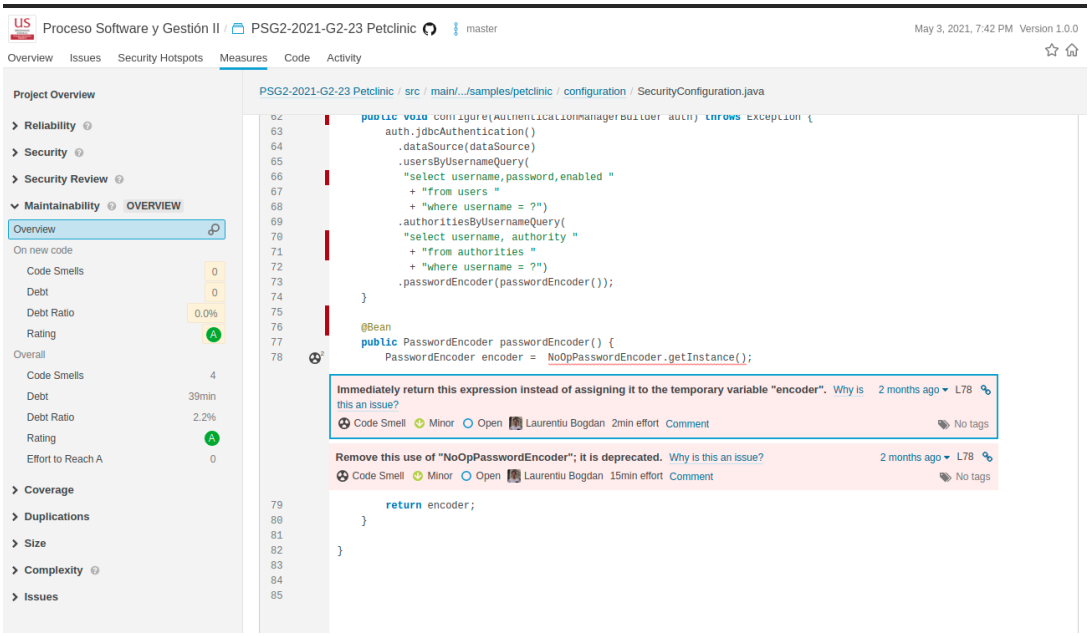

Fig 11: Dos ejemplos de Dispensable smell en el mismo fragmento de código

Nos encontramos con un método obsoleto que se ha usado y que en la próxima actualización de software podría dejar de funcionar y, en el mismo trozo de código tenemos una variable que de define justo la línea antes de ser devuelta en un return.

B) Object-Orientation Abusers: Uso incorrecto de la orientación a objetos:

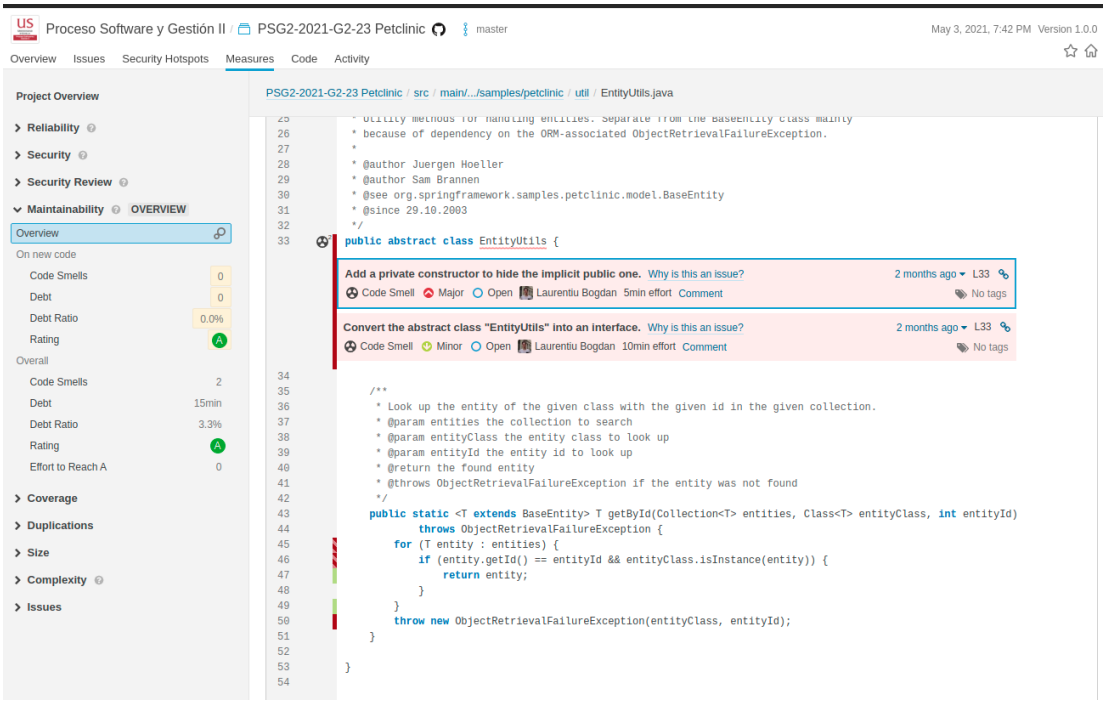

Fig 12: Dos ejemplos de Objetc-Orientation Abusers smell en el mismo fragmento de código

Se trata de una clase abstracta que no debería tener constructores públicos y posiblemente además sea más recomendable que se redefiniera como interfaz en lugar de mantenerla como clase abstracta.

I.3. Conclusiones

En la entrega S2 hubo en general una aceptable calidad del código, el cual arrastraba problemas menores y fáciles de resolver (tanto en tiempo como en dificultad) al tratarse de un momento temprano de desarrollo.

Para la S3 no se han corregido los errores anteriores y, al incrementar la cantidad de código, ha aumentado el número de smells y disminuido el porcentaje de código cubierto por los tests ya que estos no se han implementado.

En definitiva, sigue habiendo una cantidad razonablemente pequeña de errores cuyas correcciones no implicarían mucho tiempo pero es previsible que, de no corregirlos el aumento de la deuda técnica aumente rápidamente.