**PRÁCTICA 5B: SANDRA ALEGRÍA Y SEBASTIÁN RAMÍREZ**

El análisis de esta práctica se ha realizado con SonarQube 9.8.

1. Análisis de la mejora obtenida en la práctica de refactorización:

Primero vamos a realizar un análisis superficial en el que observaremos el cuadro de mando resumen de ambos proyectos y las medidas que Sonar nos ha calculado.

En primer lugar mostramos las capturas del Sonar para cada proyecto:

Proyecto sin refactorizar:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Proyecto refactorizado:

Captura de pantalla de computadora

Descripción generada automáticamente

En estas capturas observamos que para el proyecto sin refactorizar (original) y el proyecto refactorizado Sonar nos calcula unas medidas bastante similares. Esto se debe a que los cambios que realizamos en la refactorización iban más enfocados en la estructuración del código, y pasamos algo más por alto las refactorizaciones centradas en el diseño, como el nombre de las clases escrito de la manera correcta. Y Sonar realiza sus medidas centrándose más en el “estilo” y no tanto en la estructuración del código.

1. Análisis de calidad del código refactorizado:

En este apartado vamos a realizar un análisis más profundo del código refactorizado, vamos a analizar los datos que nos proporciona Sonar para identificar los problemas principales que afectan a la calidad del código.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamentePara ello hemos empezado entrando en los Issues del proyecto y nos hemos fijado en el apartado Severity, aquí podemos observar los problemas en orden de importancia según indica el Sonar en sus métricas.

En este apartado podemos observar que tenemos un error bloqueante, cero errores críticos y 31 errores mayores. Estos errores nombrados los consideramos los más importantes.

Además hay algunos errores menores, que afectan en menor medida. Sin embargo, no podemos pasarlos por alto ya que en las siguientes observaciones realizadas podemos encontrarnos con que errores menores afectan más de lo esperado.

Para obtener más información acerca de cuales son los problemas más importantes vamos a ir al apartado Measures dentro del Sonar en el cual encontramos la gráfica siguiente:

Gráfico, Gráfico de burbujas

Descripción generada automáticamente

A simple vista observamos que los colores de las clases son aceptables que hace referencia a la fiabilidad y seguridad del código. Y el tamaño hace referencia a la cantidad de líneas de código.

Imagen que contiene Texto

Descripción generada automáticamenteSin embargo, podemos obtener información mas concreta poniendo el ratón encima de cada círculo (que hace referencia a cada clase). Y obtenemos la información que mostramos en la captura de la derecha.

El valor que vamos a tener más en cuenta va a ser el “Technical Debt” que hacer referencia al tiempo de trabajo necesario para corregir los errores presentes dentro de la clase en cuestión.

Por ejemplo, en la captura de la derecha observamos que la clase Cliente tiene un tiempo de trabajo necesario de 43 min.

Después de observar todas las clases de la manera explicada hemos obtenido la siguiente información. Las clases con mayor tiempo de trabajo (y por tanto, con mayor cantidad de errores o errores más significativos) son: Cliente (45 min) y Dirección (30 min). El resto de las clases tienen todas un “Technical Debt” de entre siete y cero.

Cabe destacar que aunque tengan un tiempo de trabajo muy pequeño pueden tener errores muy importantes pero de rápida corrección. Un ejemplo representativo de esto es el error “Blocker” que observamos anteriormente, que se encuentra en la clase Valor que tiene un tiempo de trabajo de 5 min. Por lo que esto es únicamente orientativo y no descartamos clases únicamente por este motivo.

Dentro de la clase Cliente (45 minutos de tiempo de trabajo) hay varios errores significativos, que suman 10 min cada uno, y consideramos de los errores más importantes (a pesar de que sean errores menores, ya que afectan a la seguridad de la clase Cliente), lo mismo pasa con la clase Dirección.

Tabla

Descripción generada automáticamente con confianza media

Tabla

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamenteEn adición a lo anterior, vamos a entrar en el apartado Issues y dentro de esto vamos a Rule. Aquí observaremos que regla se incumple más.

Aquí llegamos a la conclusión de que la regla que más se incumple (un total de 21 veces) es “Junit assertTrue/asserFalse should be simplified to the corresponding dedicated assertion”. Esta regla quiere decir que tendríamos que cambiar los assertTrue/assertFalse por assertEquals. Esto podemos relacionarlo con los errores “Major” que en su gran mayoría (de 31 errores, 21 hacen referencia a esta regla).