1. Calcular la deuda técnica existente dentro del proyecto nuevo utilizando las herramientas que se han utilizado hasta este momento en el curso. Este cálculo debe apoyarse en las métricas de arquitectura vistas durante las semanas 5 y 6. Deberán realizar un reporte en que expliquen el proceso seguido para esto.

Para poder calcular la deuda técnica se utiliza SonarCloud, en este se hace primero la carga del código inicial junto con la configuración del archivo build.yml necesario para ejecutar el análisis inicial de la aplicación. Se adiciona la configuración necesaria al archivo pom.xml con las credenciales del proyecto creado:

* Proyecto GitHub: <https://github.com/DB20/NuevoProyectoMetricas>
* Proyecto Sonar: <https://sonarcloud.io/summary/overall?id=sonarqubekb_proyectonuevo>

Una vez se realiza el job inicial se encuentra con el siguiente resultado para la parte de Sonar:

A screenshot of a computer

Description automatically generated

Acá vemos que se tiene un Quality Gate exitoso y se ven los distintos problemas de calidad que se encuentran dentro del programa. Para calcular la deuda técnica se deben de sumar las siguientes variables:

* Code Smells: 2 days 5 hours
* Bugs: 1 hr 25 min
* Vulnerabilities: 0 days

Considerando estos valores la deuda técnica total es de 2 días 6 horas 25 minutos (2,27 días). Ahora bien, se considera que SonarQube considera el costo de desarrollar una línea es de 0.06 días (8 horas = 1 día). Teniendo esto en cuenta y considerando que la cantidad de líneas de código son 3.800, lo que se traduciría en 228 días; se considera que la deuda técnica que estamos considerando es del 0,9956% del proyecto total.

1. Calcular, con Sonargraph, las métricas de acoplamiento y cohesión del proyecto. En particular, deberán utilizar las medidas de ACD y los grupos cíclicos (otras posibles métricas para tener en cuenta incluyen el PC o el CCD) para evaluar la calidad del proyecto con respecto al acoplamiento y la cohesión.

Para poder analizar el código en lenguaje java, es necesaria la inclusión de las clases dentro del código, para ello se hace necesaria compilación del código para la creación de las clases dentro del folder “target”, para ello primero se ubica en la carpeta raíz del programa y se ejecuta el código “mvn clean install”

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Una vez se tiene el programa ajustado, se hace el importe en SonarGraph, allí se encuentran las siguientes métricas

A screenshot of a computer

Description automatically generated

Dentro de estas, haremos énfasis en las que brindan la información sobre el Acoplamiento y la cohesión, para ello vamos a ver primero la métrica del system ACD, esta tiene un valor de 8,26, lo que indica que se tiene un acoplamiento relativamente bajo; pues, lo que nos dice este indicador es que en promedio para modificar un archivo se deben de modificar otros 8,26 archivos del programa.

Por otra parte, se observa que el programa contiene un único grupo cíclico, en este se incluyen 307 de código (LOC) de los 3.758 totales. Esto demuestra un bajo porcentaje de código entrelazado ( lo que generaría un efecto negativo en el mantenimiento del código si fuera alto). Esto sugiere que se tiene un impacto del 8.17% de la base del código cada vez que se realiza un cambio en el programa.

Otras métricas importantes que nos permiten observar este aspecto de calidad son el propagation cost que tiene un valor de 23.59, un valor medio que permite confirmar que el programa se encuentra acoplado pero no en gran medida.

A screenshot of a computer

Description automatically generated

1. Analizar el impacto de las diferentes clases, métodos y atributos en las métricas de acoplamiento y cohesión.

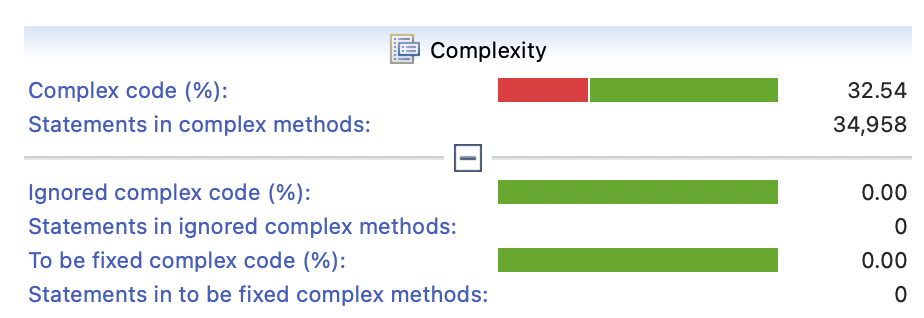
En cuanto a las diferentes clases se considera que se tiene las métricas de Logical Programming Element, estas permiten ver la dependencia que tienen entre las diferentes clases que tenemos en el programa. Aquí vemos que la clase “model.data\_structures.Vertex<K extends Comparable<K>, V extends Comparable<V>>”es la que más dependencias tiene; y, en general, se tiene una dependencia entre clases no tan alta, ya que, solo se consideran 7 con un valor por encima de 10

A screenshot of a computer

Description automatically generated

1. Calcular, con Sonargraph, la métrica de complejidad del proyecto. En particular, deberán tener en cuenta la cantidad de instrucciones complejas dentro de un método para evaluar la calidad del proyecto.

En cuanto a la complejidad del proyecto se analizan sus métodos y módulos; para ello, Sonargraph brinda un apartado que permite conocer la complejidad del código; en este caso para el programa se encuentra que la complejidad representa un 13.46%, es decir se tiene aproximadamente una décima parte de código que es compleja; esto, no supera el perfil de calidad aceptado que es del 15%. Por otra parte, una métrica importante que nos brinda SonarGraph es el hecho de que se cuenta con 262 líneas de código complejo, que considerando la cantidad total de LOC (Lines of Code) de 3.758 es cerca de un 7% de la cantidad total de líneas del código



A screenshot of a computer

Description automatically generated

1. Analizar el impacto de las diferentes clases, métodos y atributos en la métrica de complejidad.

Para considerar las métricas que se pueden analizar de parte de SonarGraph para la complejidad son las que se muestran en la siguiente figura:

A screenshot of a computer

Description automatically generated

En esta figura se observa que el “Number of Statements in Complex Methods” es de 262 que representa en “Number of Statements in Complex Methods (%)” un 13,46%

1. Identificar, con Sonargraph, los bloques de código duplicado del proyecto.

En cuanto a la cantidad de líneas duplicadas en el código tenemos un solo bloque duplicado, y en cuanto a redundancia hablamos del 3.05% del código, un porcentaje realmente bajo

A screenshot of a computer

Description automatically generated

1. Analizar la viabilidad de solucionar cada uno de los clones identificados y describir un plan para la solución del clon, en caso que sea posible. Justificar en caso de que no.

En cuanto a las clases duplicadas vemos que el problema se encuentra en dos clases que tienen métodos idénticos. Estas clases son las siguientes:

A screenshot of a computer program

Description automatically generated

Para solucionar este problema, se puede crear una clase de utilidad que permita a ambas clases invocar estos códigos y no ponerlos directamente en ambas clases.

8. Identificar los refactoring necesarios para aumentar la calidad del proyecto con respecto a las métricas evaluadas: acoplamiento y cohesión, complejidad y duplicación de código; así como las clases, métodos y atributos involucrados para cada uno de los problemas identificados en las métricas.

9. Aplicar los refactorings necesarios para aumentar la calidad del proyecto con respecto a las métricas evaluadas: acoplamiento y cohesión, complejidad y duplicación de código.

10. Explicar en detalle el cambio y describir su efecto sobre los elementos del programa (clases, métodos y atributos).

11. Calcular nuevamente la deuda técnica para observar el efecto de todos los refactorings (necesariamente la deuda técnica debe haber disminuido) y analizar el resultado obtenido.