Informe Practica 5B

Kenneth Moisés Romero Monserrate y David Ruiz Rodriguez

- **Análisis Superficial**
    - GestionVentasTienda
        - Incidencias
            - Security: 0 incidencias.
            - Reliability: 2 incidencias de Bajo Impacto.
            - Maintainability: 14 de Alto Impacto, 57 de Medio Impacto y 33 de Bajo Impacto
        - Tiempo Total: 7h y 32 min.
        - Principales Métricas de Complegidad.
            - Complegidad Ciclomática: 71
            - Complegidad Cognitiva: 74
    - GestionVentasTiendaRefactor
        - Incidencias
            - Security: 0 incidencias
            - Reliability: 2 incidencias de Bajo Impacto.
            - Maintainability: 12 incidencias de Alto Impacto, 53 de Medio Impacto y 28 de Bajo Impacto.
        - Tiempo Total: 6h y 1 min.
        - Principales Métricas de Complegidad.
            - Complegidad Ciclomática: 61
            - Complegidad Cognitiva: 55
- **Análisis Profundo**

Entrando a filtrar las incidencias y jugando con los filtros, vemos que la totalidad de incidencias de Alto Impacto son de Mantenibilidad, de tipo Code Smell específicamente y todas en el cuerpo del programa, no en los tests. Casi la mitad de todas las incidencias están en el fichero Tienda.java.

El fichero qué más esfuerzo va a llevar arreglar es Tienda.java según los diagramas de burbujas de Security, Reliability y MainTainability. Los tres.

- **Plan de la mejora de la calidad del código**

Los incidentes de este plan están ordenados, primero por **Severidad**, de mayor severidad a menos. Luego por **tipo**. Son más importantes las incidencias de Vulnerabilidad (para proteger a los clientes), luego los Bugs (para que el programa haga lo que debe) y luego los Code Smell.

En caso de que haya un empate, se ordenará en función de su **Calidad de Software** (Software Quality), primero Security, segundo Reliability y tercero Maintainability.

Y si aun así hay empate, se deberá revisar el código y agrupar los incidentes en función de si se **solucionan de la misma manera**.

Por último, se **agruparán por ficheros**, y se atenderán primero los incidentes que estén dentro del mismo fichero.

Al final, como medida estándar en caso de que no se sepa como ordenar, se ordenarán **por fecha**. El incidente más antiguo primero.

    - Plan de mejora:
        - Define a constant instead of duplicating this literal " Nombre: " 3 times. (Code Smell, Alto Impacto, Tienda.java, 8 min, Maintainability, 14 days ago)
        - Define a constant instead of duplicating this literal " DNI: " 3 times. (Code Smell, Alto Impacto, Tienda.java, 8 min, Maintainability, 14 days ago)
        - Define a constant instead of duplicating this literal " Id: " 3 times. (Code Smell, Alto Impacto, Tienda.java, 8 min, Maintainability, 14 days ago)
        - Define a constant instead of duplicating this literal " TotalVentasMes: " 3 times. (Code Smell, Alto Impacto, Tienda.java, 8 min, Maintainability, 13 days ago)
        - Define a constant instead of duplicating this literal "ERROR" 4 times. (Code Smell, Alto Impacto, GestionComisiones.java, 10 min, Maintainability, 14 days ago).
        - Refactor this method to reduce its Cognitive Complexity from 23 to the 15 allowed. (Code Smell, Alto Impacto, GestionComisiones.java, 13 min, Maintainability, 12 days ago).
        - Add a default case to this switch. (Code Smell, Alto Impacto, GestionComisiones.java, 5 min, Maintainability, 12 days ago).

- **Resolución de incidencias**

Durante el proceso de revisión de código, se identificaron varias incidencias en las clases Tienda y GestionComisiones. A continuación, se detallan las mejoras realizadas para abordar estas incidencias:

Clase Tienda:

1. Duplicación de Literales en Método vuelcaDatos:
    a. Se detectó que los literales como "Nombre:", "Id:", "DNI:", "TotalVentasMes:", y "TotalComision:" se repetían varias veces en el

método vuelcaDatos de la clase Tienda. Esto dificultaba la mantenibilidad y legibilidad del código.

b. Solución: Se crearon constantes estáticas al inicio de la clase para representar estos literales. Esto permite una fácil modificación y mantenimiento del código en el futuro, ya que cualquier cambio en estos valores se reflejará automáticamente en todo el código relevante.

Clase GestionComisiones:

1. Complejidad Cognitiva Elevada en el Método main:
    a. Se observó que el método main de la clase GestionComisiones presentaba una complejidad cognitiva elevada debido a su longitud y la presencia de varios bloques de código anidados.
    b. Solución: Se refactorizó el método main dividiéndolo en métodos más pequeños y enfocados. Esto mejora la legibilidad y mantenibilidad del código al proporcionar una estructura más clara y modular.
2. Switch Statement sin Default Case:
    c. Se identificó que el switch statement en el método main de la clase GestionComisiones no incluía un caso predeterminado para manejar opciones no válidas del menú.
    d. Solución: Se agregó un caso predeterminado al switch statement para manejar las opciones no válidas del menú de manera adecuada.

Estas mejoras abordan las incidencias detectadas en las clases Tienda y GestionComisiones, lo que contribuye a un código más robusto, mantenible y fácil de entender.

- **Análisis final**

El análisis superficial revela una mejora significativa en la calidad del código entre las dos versiones de las clases GestionVentasTienda y GestionVentasTiendaRefactor. A continuación se detallan los hallazgos clave:

GestionVentasTienda:

- Security: No se detectaron incidencias de seguridad.
- Reliability: Se identificaron 2 incidencias de bajo impacto.
- Maintainability: Se encontraron 14 incidencias de alto impacto, 57 de impacto medio y 33 de bajo impacto.
- Tiempo Total: Se invirtieron 7 horas y 32 minutos.
- Principales Métricas de Complejidad: Complejidad Ciclomática: 71, Complejidad Cognitiva: 74.

GestionVentasTiendaRefactor:

- Security: No se encontraron incidencias de seguridad.
- Reliability: Se detectaron 2 incidencias de bajo impacto.
- Maintainability: Se identificaron 12 incidencias de alto impacto, 53 de impacto medio y 28 de bajo impacto.
- Tiempo Total: Se dedicaron 6 horas y 1 minuto.
- Principales Métricas de Complejidad: Complejidad Ciclomática: 61, Complejidad Cognitiva: 55.

Análisis Profundo:

- La mayoría de las incidencias de alto impacto están relacionadas con la mantenibilidad del código, específicamente identificadas como Code Smell.
- Las incidencias se agruparon y priorizaron según su severidad, tipo y ubicación en el código.
- Se observó que el fichero Tienda.java requiere el mayor esfuerzo para abordar las incidencias, ya que concentra una cantidad significativa de problemas en términos de seguridad, fiabilidad y mantenibilidad.

Plan de Mejora:

- Las incidencias se planificaron en función de su severidad, tipo y ubicación en el código, priorizando la resolución de problemas de mayor impacto.
- Se definió un orden de resolución basado en la gravedad de las incidencias, el tipo de problema y la ubicación en el código.
- Se estableció un proceso para abordar las incidencias en cada fichero, atendiendo primero a aquellas que se encuentren en el mismo fichero.
- En caso de empate en la priorización, se dio preferencia a las incidencias más antiguas.

Resolución de Incidencias:

- Se abordaron las incidencias identificadas, implementando soluciones específicas para cada problema.
- Se realizaron mejoras en la legibilidad, mantenibilidad y eficiencia del código, lo que contribuyó a una mejora general en la calidad del software.

En resumen, las mejoras realizadas en las clases Tienda y GestionComisiones han resultado en un código más robusto, mantenible y fácil de entender, lo que contribuye a una mejora significativa en la calidad del software.