Análisis de la calidad del producto

1. INTRODUCCIÓN

En este informe se va a analizar la calidad del producto desarrollado durante la primera semana del tercer sprint del proyecto integrado. Se realiza a fecha de 17 de noviembre de 2024, habiéndose desarrollado las dos interfaces nuevas que se requerían, así como la implementación del código fuente de *Ordenar Gasolineras por precio* casi en su totalidad, la de *Comparar Precios* de manera parcial y la del ticket de cambio *Mantener Municipio Sin Gasolinera* está completa. Por lo que se tiene material suficiente para analizar lo que se ha realizado e introducir cambios antes del fin del sprint.

2. ANÁLISIS DE LA CALIDAD

En la captura 2.1 podemos observar el resumen del análisis realizado por la herramienta SonarQube, podemos ver que no hay errores de seguridad(security), hay 2 errores de fiabilidad(reliability) y 51 de mantenibilidad (maintainability).

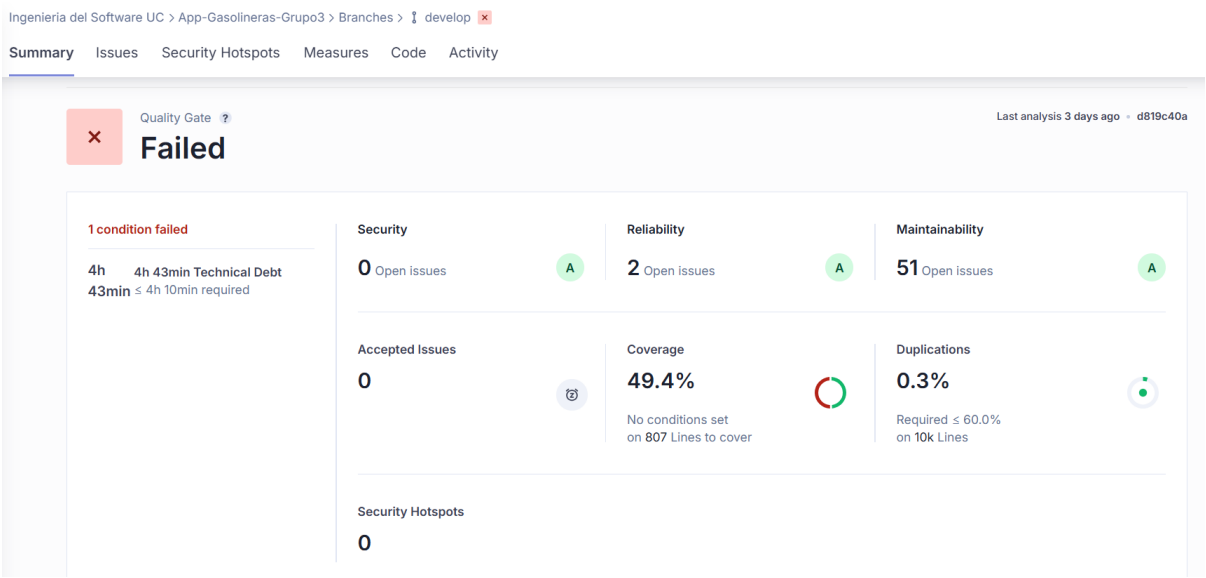
Hay un 0.3% de código duplicado, esto se debe a que en la clase *GasolinerasRepository* hay 26 líneas de código duplicadas, en concreto por el método *requestGasolinerasHistoricoFechas*.

La cobertura es inferior al 50%, en concreto es del 49.4%, pero hay que tener en cuenta que aún no han sido implementadas las pruebas.

El *Quality Gate* no pasa debido a que la deuda técnica supera el máximo establecido que es de 4h 10min.

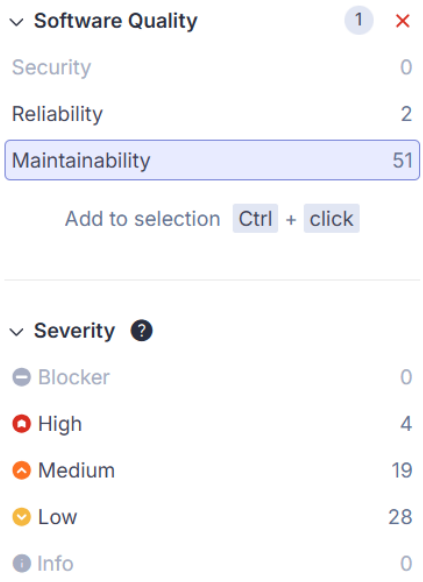
En cuanto a los errores detectados podemos observar que en su mayoría son fallos de mantenibilidad del código.

Para corregir completamente el proyecto deberíamos emplear 4 horas y 43 min.

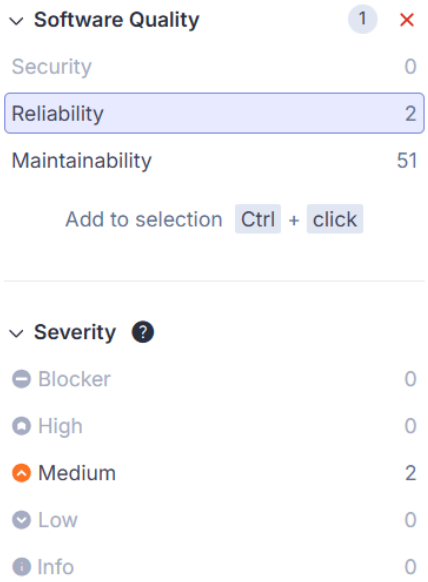


Captura 2.1 Resumen del análisis

En las capturas 2.2 y 2.3 se nos muestra el detalle de la severidad de los fallos de cada tipo. Aunque hay bastantes fallos de mantenibilidad, solo 4 son de severidad alta por lo que no hay una situación extremadamente costosa en lo que a tiempo se refiere. La mayoría son de severidad media o, sobre todo, baja. En el caso de la fiabilidad los dos fallos son de severidad media. Con esto podemos deducir que, aunque hay fallos de calidad, no son excesivamente graves por lo que se pueden solventar y reconducir la situación.



Captura 2.2 Detalle fallos mantenibilidad



Captura 2.3 Detalle fallos fiabilidad

En el caso de mantenibilidad los fallos graves son:

1. *Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation*: que es debido a que en la clase *ConsultarView* el método *init* todavía no está implementado.
2. *Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation:* que es debido a que en la clase *DetailsView* el método *init* todavía no está implementado.
3. *This class is part of 2 cycles containing 3 classes:* éste se debe a que en las clases *ConsultarView*, *RegistrarView* y *MainView* se produce una dependencia cíclica, es decir, que se están referenciando mutuamente.
4. *Define a constant instead of duplicating this literal "Gasolina" 3 times:* esto se produce porque en la clase *MainView* se utiliza el String "Gasolina" en múltiples ocasiones siendo más eficiente crear una constante.

De los de severidad media los que más se repiten son:

1. *Remove this commented out code*: tenemos varias líneas comentadas ya que estamos probando cosas, además de alguna cosa que en el código proporcionado inicialmente ya estaba comentado.
2. *Make this annonymous inner class a lambda*.
3. *Swap these 2 arguments so they are in the correct order: expected value, actual value:* a la hora de comprobar valores en los tests deberíamos de cambiar el orden de los argumentos.

De los de de seguridad baja:

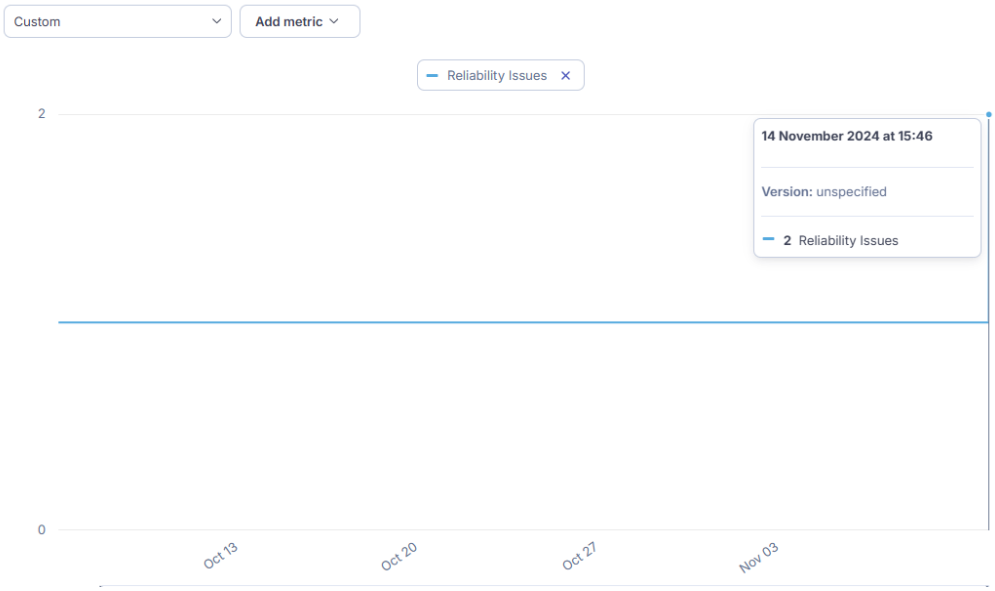
1. *Remove this unused import*: en varias clases hay imports que no se están utilizando por lo que son innecesarios y deberían ser eliminados.

Los fallos de fiabilidad son el mismo, pero uno en la clase *MainView* y otro en *DetailsView*: *Remove this field injection and use constructor injection instead":* es una mejora que no ha sido desarrollada por nuestro equipo, pero nos dice que podríamos eliminar la directiva de *inject* y hacer un constructor inyectivo en su lugar.

En las siguientes capturas 2.4 y 2.5 vemos la evolución de los diferentes errores a lo largo del desarrollo del proyecto. Se puede observar que ya existían varios

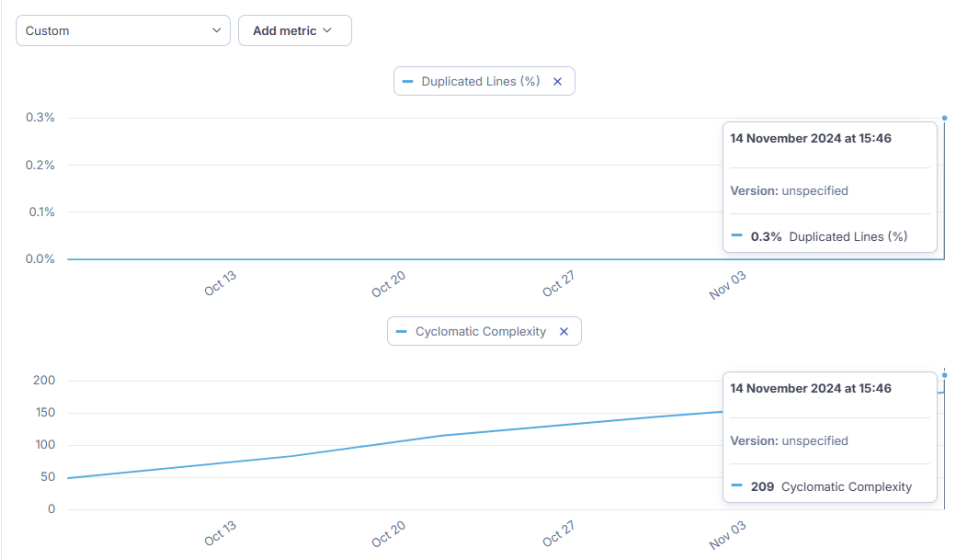
fallos de mantenibilidad al inicio del sprint. Éstos han ido aumentando ligeramente, sin embargo, llegado a un punto los fallos empezaron a disminuir, esto es normal ya que tras dos sprints se ha mejorado varios de los errores que ya se encontraban en el código, manteniéndose en un balance con los nuevos fallos implementados. Aunque al final surgieron algunos problemas más que incrementaron el número total. El resto de issues de momento se mantienen a excepción de un fallo de fiabilidad que surgió el 14 de noviembre.

Captura 2.4 Evolución fallos mantenibilidad y seguridad



Captura 2.5 Evolución fallos fiabilidad

Finalmente, en la captura 2.6, observamos que el código obtiene un conjunto de líneas duplicadas el 14 de noviembre, rompiendo esa ausencia que había tenido todo el proyecto hasta ahora. Por otra parte, la complejidad ciclomática ha aumentado debido a la incorporación de nuevas clases y métodos, lo cual era esperado. Aunque un aumento moderado en la complejidad es normal durante el desarrollo, es importante monitorear este valor para evitar que el código se vuelva más difícil de mantener y probar en futuras iteraciones.

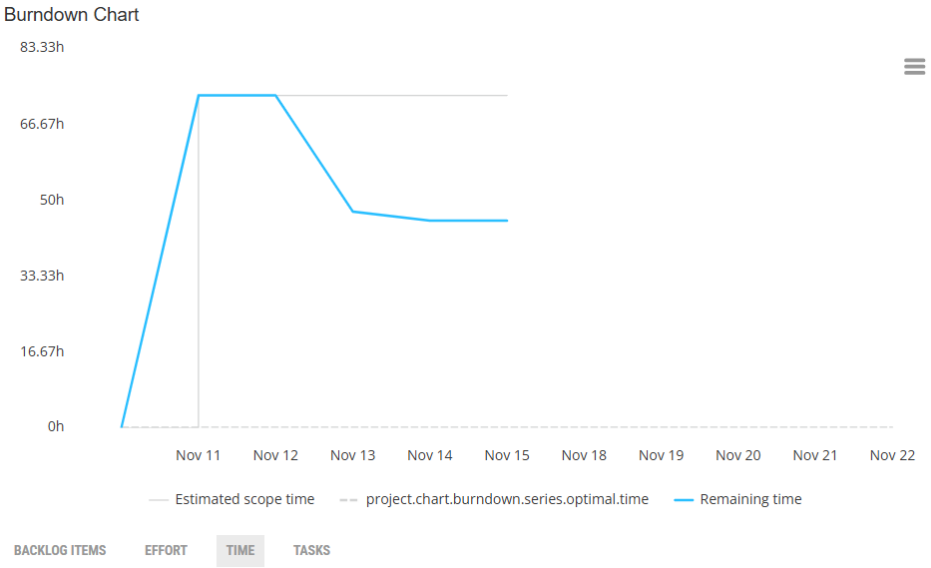


Captura 2.6 Evolución líneas duplicadas y complejidad ciclomática

3. EVOLUCIÓN CARGA DE TRABAJO

En el gráfico del Sprint Burndown Chart (Captura 3.1) podemos observar cómo ha evolucionado la cantidad de trabajo pendiente a lo largo del sprint. La línea azul representa el progreso real, mientras que la línea gris discontinua muestra la trayectoria ideal. A medida que avanza el sprint, vemos que el equipo está logrando reducir las tareas de manera constante y se mantiene bastante cerca de la línea ideal, aunque ha reducido su ritmo inicial por un momento. Esto indica que, se espera finalizar todas las tareas a tiempo. Cabe comentar que, aunque el progreso es adecuado, es importante prestar atención a los últimos días del sprint, ya que cualquier incremento en tareas no previstas o bloqueos podrían afectar la finalización. En resumen, el equipo está mantenido un buen ritmo de trabajo durante el sprint, y, de continuar con esta tendencia, se podrá cumplir con los objetivos planteados.

En función de la diferencia con la trayectoria ideal se ve que se puede aumentar la carga de trabajo ligeramente y todavía mantenerse en los tiempos establecidos, por lo que sería posible aplicar un plan de acción que redujese la deuda técnica aumentando así la calidad del código.
Este plan de acción no deberá superar la hora y media de duración con el fin de no sobresaturar a los programadores con demasiadas tareas pudiendo quedar en riesgo el trabajo del sprint.



Captura 3.1 Sprint burndown chart

4. PLAN DE ACCIÓN PARA LA MEJORA DEL CÓDIGO

Prioridad	Nombre	Severidad	Localización	Esfuerzo
1	*Define a constant instead of duplicating this literal "Gasolina" 3 times*	High	MainView	8 minutos
2	*Make this annonymous inner class a lambda*	Medium	RegistrarDescuento View RegistrarView	30 minutos
3	*Rename field "descuento"*	Medium	Descuento	10 minutos

4	*Replace this 3 tests with a single Parametrized one*	Medium	RegistrarDescuento PresenterITest	10 minutos
5	*Swap these 2 arguments so they are in the correct order: expected value, actual value*	Medium	MainPresenterITest MainPresenterTest	6 minutos
6	*Remove this unused import*	Low	Varias clases	8 minutos
				1h y 12 min

No se ha decidido resolver los otros 3 problemas de severidad alta ya que, los dos referentes a que los métodos *init* están vacíos desaparecerán cuando se realice, de manera definitiva, la implementación de esa parte del código. En el caso del otro problema de severidad alta no se va a resolver ya que no hay forma de hacerlo, debido a que para que la aplicación funcione correctamente es necesario las llamadas en ambas clases.

En cuanto a la eliminación de imports que no se usan, puede que parte de ellos sí que sean necesarios ya que por ejemplo las clases de prueba están aún sin implementar por lo que necesitarán varios de estos imports, entonces no será necesario eliminarlos y el esfuerzo se reducirá.

5. IMPLEMENTACIÓN PLAN DE MEJORA

Para ejecutar el plan de acción diseñado se incluirán tres nuevas tareas, la primera consistirá en la ejecución del punto 2 ya que lleva de por sí sola 30 minutos, la segunda, consistirá en los puntos 1 y 6 debido a que no son muy costosos por si solos (16 minutos en total) y así se puede agrupar mejor la tercera tarea, la cual consistirá en la realización del resto de tareas de severidad media (puntos 3, 4 y 5) llevando un total de 26 minutos en realizarla. Estas tareas serán realizadas por tres integrantes del grupo para equilibrar de mejor manera la carga de trabajo, ocupándose cada integrante de una tarea.

El nombre de las tareas será:

1. Plan de acción punto 2
2. Plan de acción puntos 1 y 6
3. Plan de acción puntos 3-5