Tarea 5: Code Refactoring

Hugo Eduardo Rivas Fajardo - 22500
José Santiago Pereira Alvarado - 22318
Nancy Gabriela Mazariegos Molina - 22513
Giovanni Alejandro Santos Hernández - 22523
Mauricio Julio Rodrigo Lemus Guzmán- 22461
Alexis Mesías Flores - 22562

¿Que es Exploratory Testing?

El Exploratory Testing es una técnica de pruebas de software donde los testers exploran la aplicación sin seguir un conjunto estricto de casos de prueba predefinidos. Se basa en la experiencia y habilidades del tester, quienes investigan el sistema mientras prueban, buscando identificar comportamientos inesperados, errores o inconsistencias. En lugar de depender completamente de la planificación previa, se aprende sobre la marcha y se adapta a lo que se va descubriendo.

a. *Aplicación en Agile:*
   En metodologías Agile, donde los ciclos de desarrollo son rápidos y continuos, el *Exploratory Testing* encaja perfectamente, ya que permite a los testers adaptarse al ritmo de los cambios frecuentes y hacer pruebas de manera más flexible y rápida. Se puede usar en sprints para validar características en desarrollo y detectar problemas antes de que se formalicen todos los casos de prueba automatizados.
b. *Diferencia con testeo manual:*
   Aunque ambos métodos requieren interacción humana, el *Exploratory Testing* no sigue un plan predefinido de pruebas, mientras que el testeo manual generalmente implica ejecutar casos de prueba definidos de antemano. El *Exploratory Testing* es más flexible y reactivo, mientras que el testeo manual puede ser más estructurado y repetitivo.
c. *Ventajas de este vs otros métodos de testing que hemos visto en el curso:*
   - Flexibilidad: Permite descubrir errores inesperados que no están contemplados en los casos de prueba predefinidos.
   - Adaptación rápida: Dado que Agile implica entregas frecuentes, este método es ideal para adaptarse a los cambios rápidos sin depender de documentación detallada.
   - Eficiencia: No requiere tiempo para escribir y mantener grandes conjuntos de casos de prueba detallados, lo que lo hace más adecuado para proyectos ágiles.
d. *¿Esta de acuerdo que esta es una caja negra del testing?*
   Sí, el Exploratory Testing se puede considerar una técnica de caja negra porque el tester no necesita tener un conocimiento profundo del código fuente o de la implementación interna. En su lugar, interactúa con la aplicación a nivel funcional, explorando cómo responde a diversas entradas y acciones desde una perspectiva de usuario.
e. *Defina como incorpora este método hacia su proyecto en grupo.*
   Debido a que usamos selenium para las pruebas end to end, podemos ejecutar aleatoriamente flujos de acción, creación, etc. También se han ido probando todas las transacciones con usuarios aleatorios sin explicarle nada previo, por lo que debe de explorar y aprender a usar el sistema durante la marcha.

¿Que es Code Refactoring?

El Code Refactoring es el proceso de mejorar el diseño interno de un código existente sin cambiar su comportamiento externo. El objetivo es hacer el código más limpio, más eficiente y fácil de mantener. Se enfoca en la mejora continua del código para optimizarlo y eliminar duplicidades o complejidad innecesaria.

a. *Aplicación en Agile:*
   En Agile, el Refactoring es una actividad continua que se realiza a lo largo de los sprints. La metodología Agile promueve la entrega continua de software, y el Refactoring ayuda a mantener el código en buenas condiciones para que pueda adaptarse y crecer sin convertirse en una carga técnica. Durante cada sprint, se suelen reservar periodos para hacer pequeñas mejoras de Refactoring en el código, lo que facilita la escalabilidad y evita problemas técnicos a largo plazo.

b. ¿Que retos importantes tiene para aplicarlo en su proyecto?
   - Tiempo y recursos limitados: En proyectos con plazos ajustados, puede ser difícil justificar dedicar tiempo al Refactoring si no está relacionado directamente con la entrega de una nueva funcionalidad.
   - Pruebas insuficientes: Hacer Refactoring sin tener un buen conjunto de pruebas automatizadas puede ser riesgoso, ya que podría introducir errores no detectados.
   - Resistencia del equipo: Algunas veces los desarrolladores y stakeholders pueden resistirse al Refactoring porque no ven un beneficio inmediato o no entienden su importancia a largo plazo.

c. ¿Cuales son los beneficios de aplicar este de forma general a cualquier proyecto?
   - Mejora la mantenibilidad: Un código más limpio y estructurado es más fácil de entender, lo que facilita su modificación en el futuro.
   - Reducción de la deuda técnica: Al reducir la complejidad y eliminar el código duplicado, se evita la acumulación de deuda técnica que puede obstaculizar el progreso futuro.
   - Mejor rendimiento: Aunque no siempre es el objetivo principal, el Refactoring también puede mejorar el rendimiento del software al optimizar código ineficiente.
   - Facilita las pruebas y el debugging: Un código más limpio es más fácil de probar y depurar, lo que puede aumentar la calidad del software.

d. ¿Puede definir si existen distintos tipos de técnicas para aplicar este?
   Sí, existen diversas técnicas de Refactoring, entre ellas:
   - Extracción de métodos (Method extraction): Se toma un fragmento de código y se convierte en un método o función independiente para mejorar la legibilidad.
   - Renombrado de variables o métodos (Rename refactoring): Cambiar los nombres de variables o métodos para que reflejen mejor su propósito, mejorando la claridad del código.
   - Reubicación de clases o métodos (Move refactoring): Mover clases o métodos a módulos más apropiados o encapsular lógicamente funciones relacionadas.

- Eliminación de duplicación (Remove duplication): Combinar o eliminar código duplicado en diferentes partes del sistema.

e. *Defina como incorpora este método hacia su proyecto en grupo.*
    - Depurando el código inutilizado
    - Code Coverage para revisar lint´s warnings.
    - Dado a que tenemos los moldes, siempre vamos a mandar y recibir lo mismo, por lo que podemos cambiar la manera en la que se codifica, sin cambiar el resultado.