**Introducción al Testing**

**Prework**

* Analizar el porqué y para qué probamos el software.
* Reconocer las diferentes etapas del testing en el ciclo de desarrollo de software.
* Identificar los diferentes tipos de pruebas

\*\*\*Testing\*\*\*

El testing reduce significativamente errores de producción y costos de mantenimiento

La retroalimentación nos permite conocer si nuestras acciones están teniendo el efecto deseado

Las pruebas ayudan a conseguir la retroalimentación lo antes posible de manera instantánea

Los objetivos de las pruebas son:

\* Validar que el software funciona como fue diseñado para un conjunto de datos dado.

\* Evaluar las suposiciones hechas en las especificaciones de requisitos y diseños a través de demostración concreta

\* Encontrar y documentar errores en el software

Fases mentales del testing:

Fase 0: No existe diferencia entre la prueba y la depuración. Aparte del soporte a la depuración, las pruebas no tienen ningún propósito

Fase 1: El objetivo de la prueba es demostrar que el el software funciona

Fase 2: El propósito de las pruebas es demostrar que el software no funciona

Fase 3: El propósito de la prueba no es demostrar nada, sino reducir el riesgo percibido cuando el software no trabaja en valores aceptables

Fase 4: Las pruebas no son un acto. Es una disciplina mental que arroja software de bajo riesgo sin mucho esfuerzo en pruebas.

Me siento identificado con la fase 1 porque cuando las pruebas pasan, pienso que el software funciona

\*\*\*Triángulo de hierro\*\*\*

El objetivo principal del triángulo de hierro es sacar los proyectos adelante lo más rápido posible respetando los estándares de calidad de nuestro desarrollo

El cliente sólo puede fijar dos de los tres vértices: tiempo, alcance, coste. El resultado del equilibrio de estos tres elementos es la calidad.

Pasos del triángulo de hierro:

1. Definir el proyecto: deben definirse los objetivos que se planean alcanzar y aquellos factores que podrían influir en el proceso y desarrollo de las actividades para el óptimo cumplimiento del proyecto. En este punto se da forma al proyecto para tener claro lo que se hará y aquello que pueda afectarlo.

2. Inicio del proyecto: se deberá planificar los medios que se usarán durante todo el desarrollo del proyecto, es decir, cómo se realizará.

3. La planificación: aquí se marca y se define como se desarrollará el proyecto detalladamente. En este punto se define que habrá en cada uno de los vértices del triángulo de hierro y así mismo se evalúan las posibilidades para así tener un plan óptimo y exacto.

4. La ejecución: se deben aplicar las tareas en los tiempos que se hayan definido en los puntos anteriores.

5. La monitorización: se revisa, vigila y monitorea cada una de las fases que tiene el proyecto para que estas salgan adelante.

6. Cierre del proyecto: Al llegar a las metas, se asumen como cumplidos los objetivos; aquí deben analizarse qué recursos se han empleado y evaluar todo el proceso para futuros proyectos o mejoras específicas.

Objetivos del triángulo de hierro:

\* Requisitos: que el software sea el correcto.

\* Análisis, diseño e implementación: que el software funcione correctamente.

\* Pruebas: validar que el software sea el correcto y verificar que funcione correctamente con reducción de coste y/o tiempo.

\*\*\*Metodologías de prueba\*\*\*

\* Cobertura de prueba

\* Cobertura de código

\*\*\*Coverage (Cobertura)\*\*\*

La cobertura es la medida porcentual del grado en que el código fuente de una aplicación ha sido ejercitado durante las pruebas.

Cobertura de prueba: La cobertura es la medida porcentual del grado en que el código fuente de una aplicación ha sido ejercitado durante las pruebas. Sirve para:

\* Identifica defectos en etapas tempranas: Puede identificar brechas en los requisitos, casos de prueba y defectos en las primeras etapas del ciclo de vida de desarrollo de su producto. Esto ahorrará tiempo y recursos a tu equipo u organización.

\* Mejor cobertura: La cobertura de prueba crea más casos de prueba para garantizar una cobertura superior. Esto ayuda a identificar y erradicar los defectos y disminuye el trabajo por hacer en etapas posteriores. Además, puede aumentar la satisfacción del cliente.

\* Elimina casos redundantes: La cobertura nos ayuda a identificar y eliminar casos de prueba que no tienen mucho sentido en el proyecto actual. Estos casos pueden ser eliminados para hacer que el código general sea más ligero.

\* Mayor ROI: Con la reducción de defectos en las etapas de producción y de prueba de aceptación por parte del usuario, la cobertura de prueba genera un impacto positivo en el ROI. Todos los recursos que se habrían gastado en arreglar los defectos, se convierten en ganancias.

\* Descubre áreas descubiertas: La cobertura ayuda a descubrir áreas de un programa que no habían sido cubiertas por un conjunto de casos de prueba. Ayuda a que el programa sea más sólido y disminuye significativamente los errores.

\* Control superior: La cobertura de prueba nos da control sobre los recursos durante el ciclo de vida del desarrollo del producto. Ahorra tiempo al eliminar los defectos antes y más rápido. El tiempo ahorrado te permite mantener un registro de los costos. Y lo más importante, puedes tener mayor control sobre el alcance del proyecto.

\* Ciclos de prueba más suaves: Puede evitar la fuga de defectos mediante el análisis de cobertura de prueba. La cobertura de prueba también ayuda en las pruebas de regresión, la priorización de casos de prueba, el aumento de la suite de pruebas y la minimización de la suite de pruebas. Esto conduce a ciclos de prueba más suaves pero eficientes.

\*\*\*Cobertura de código\*\*\*

La cobertura del código es la medida del grado en que se ha ejecutado el código del software. Cubre tanto las pruebas manuales como las automáticas y le proporciona el porcentaje de código que se ha ejecutado. Con base en esta información, puede implementar pruebas adicionales para el producto y hacerlo mucho más completo.

La principal diferencia entre la cobertura de la prueba y la cobertura del código es que la cobertura de la prueba es una medida cualitativa y la cobertura del código tiene más que ver con la medición cuantitativa en términos de prueba.

La cobertura de código ayuda a determinar qué parte del código se ejecuta mientras se ejecuta la aplicación, mientras que la cobertura de prueba describe los casos de prueba que se pueden escribir en el documento de requisitos.

\*\*\*Técnicas de cobertura de prueba\*\*\*

>> Cobertura de estados de cuenta

Asegura que todas las declaraciones del código fuente se hayan probado al menos una vez.

Proporciona los detalles de los bloques de código ejecutados y fallidos del total de bloques de código.

Ventajas:

- Se puede aplicar directamente al código objeto y no requiere procesar el código fuente.

- Verifica lo que se espera que haga y no haga el código escrito

Desventajas:

- Cubre solo las condiciones verdaderas de cada declaración.

- No informa cuando el bucle llega a las terminaciones.

- La cobertura de la declaración es completamente insensible a los operadores lógicos (|| y &&).

- La cobertura del estado de cuenta es la cobertura básica y, por lo tanto, no garantiza una cobertura del 100%.

>> Cobertura de decisión / sucursal

Comprueba que se cubren todas las rutas o ramas posibles en el código.

Se puede calcular encontrando el número mínimo de caminos que aseguren que se hayan cubierto todos los bordes.

Ventajas:

- Cubre tanto las condiciones verdaderas como las falsas improbables de la cobertura de la declaración.

- Válida si se prueban todas las ramas.

Desventajas:

- Ignora las ramas dentro de las expresiones booleanas que se producen debido a operadores de cortocircuito.

>> Cobertura de ruta

Implica el uso del código fuente de un programa para encontrar todas las rutas ejecutables posibles.

Ventajas:

- Ayuda a reducir las pruebas redundantes.

- Cubre todas las declaraciones y ramas del código.

Desventajas:

- Probar cada ruta es desafiante y requiere mucho tiempo porque una cantidad de rutas es exponencial al número de ramas.

- A veces, muchos caminos son imposibles de ejercitar debido a las relaciones de datos.

>> Cobertura de condición

Verifica si se han ejercido los dos resultados ("verdadero" o falso ") de cada condición.

El resultado del punto de decisión solo es relevante para verificar las condiciones. Requiere dos casos de prueba por condición para dos resultados.

Ventajas:

- La cobertura de condiciones mide las condiciones de forma independiente entre sí.

- Tiene mejor sensibilidad al flujo de control.

Desventajas:

- Similar a la cobertura de sucursales / decisiones.

>> Cobertura de valor límite:

Útil para aquellas aplicaciones en las que se produjo un error debido a los números de entrada. Y estos errores ocurrieron en los valores límite.

Ventajas:

- Es fácil probar un pequeño conjunto de datos en lugar de probar una gran cantidad de conjuntos de datos.

- Es fácil de usar debido a que es fácil automatizar la naturaleza y la uniformidad de las pruebas identificadas.

Desventajas:

- No puede probar las dependencias entre dos entradas.

- No puede cubrir código que contenga funciones booleanas

\*\*\*Métricas de cobertura de prueba\*\*\*

Miden el esfuerzo de la prueba y ayudan a conocer cuánto de la aplicación se probó.

Las métricas de cobertura se dividen en tres partes:

\* Métricas a nivel de código

\* Métricas de prueba de funciones

\* Métricas a nivel de aplicación.

\*\*\*5 métricas clave de cobertura de pruebas\*\*\*

\* Métricas de nivel de código => Tests ejecutados = (número de tests corridos/número total de tests a correr) x 100%

\* Métricas de prueba de funciones => Cobertura de requisitos = (número de requisitos cubiertos/número total de requerimientos) x 100%

\* Casos de prueba por requisito =>

Esta métrica es muy importante ya que muestra el progreso del desarrollo de la aplicación / software.

Es una tabla que con encabezado: Request | Test Case | Test Result (Passed, Failed, Pending)

\* Métricas de nivel de aplicación => Densidad de defectos = número de defectos conocidos/Size of software entity

\*Requisitos sin cobertura de prueba =>

Después de calcular la cobertura de requisitos, encontrarás algunos requisitos que no están cubiertos.

Es importante conocer cada requisito que no se ha cubierto y en qué etapa se encuentra.

Es una tabla que con encabezado: Request | Test Case | Test Result (To do, done).

Esta métrica ayuda a los ingenieros y desarrolladores de pruebas a identificar y eliminar los requisitos descubiertos de los requisitos totales antes de enviarlos a la fase de producción.

\*\*\*Matriz de cobertura de la prueba\*\*\*

TCM es una tabla matricial que nos ayuda a asegurarnos que se tienen en cuenta todas las posibles condiciones y características que se van a probar. También ayuda a identificar las posibles brechas. Puede verse como un checklist que asegura que la funcionalidad de la pantalla dada se verifica en todas las combinaciones posibles.

No hay requisitos previos. Puedes crear tu propia tabla:

Característica | Módulo | Tipo de caso | Caso de prueba

\*\*\*Para mejorar la cobertura de la prueba\*\*\*

\* Eliminar código muerto:

Al hacerlo, se baja el denominador y aumenta la cobertura total: cobertura total = código cubierto/codigo total)

\* Eliminar código redundante / clonado:

**Work**

* Demostrar el porqué y para qué probamos el software
* Identificar las diferentes etapas del testing en el ciclo de desarrollo de software
* Aplicar los diferentes tipos de pruebas
* Elaborar pruebas aplicando técnicas básicas

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*CONCLUSIÓN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Exceso de pruebas: La cobertura al 100% no significa que el 100% esté probado

\* Falta de pruebas: Los valores de baja cobertura, inferiores al 50% son una señal de problemas.

\* Equilibrio: Cuando existe un equilibrio en las pruebas, rara vez se encuentran errores en producción

Reto 1: Altas y consultas de entrevistadores

Se discutió en equipo:

* Mediante terminal permite agregar nuevos entrevistadores.
* Mediante terminal se pueden consultar a los entrevistadores existentes en el sistema.
* Algunos de los datos que se esperan de un entrevistador son: correo, nombre completo, tecnologías, entre otras.
* La persistencia de datos no está en el alcance de este sprint, por lo que los datos serán efímeros viviendo solo en memoria.

Reto 2: Probando nuestro software

Se discutió en equipo:

* Verificar que se puedan agregar nuevos entrevistadores mediante terminal.
* Verificar se puedan consultar a los entrevistadores existentes en el sistema.
* Verificar que se hayan guardado los datos esperados de un entrevistador, como son: correo, nombre completo, tecnologías, entre otras.
* Verificar que los datos vivan en memoria.

Reto 3: Añadir persistencia de datos

Se discutió en equipo:

* Añadir persistencia de datos
* Se debe poder leer y escribir los datos de los entrevistadores en archivos

**Postwork**

* Desarrollar funcionalidad adicional a nuestro programa.
* Añadir pruebas unitarias adicionales para probar su funcionalidad.
* Aplicar técnicas de diseño de pruebas.
* Utilizar la herramienta JUNIT.