Curso de testing funcional y accesibilidad web

Bloque #2

Clase 2

Bienvenidos y bienvenidas al segundo bloque del curso de testing funcional y accesibilidad web.

1. Introducción a las técnicas

Las técnicas estáticas, se basan en el examen manual o en el análisis automatizado del código, o pueden basarse también en cualquier otra documentación del proyecto en los que no tengamos que ejecutar el código propiamente dicho.

Las técnicas estáticas, suelen realizarse antes de las técnicas dinámicas, porque los defectos que encontremos van a ser al principio del ciclo de vida del producto, por lo cual van a ser menos costosos de corregir que los detectados durante la ejecución de las pruebas dinámicas. Ambas técnicas son complementarias ya que detectan diferentes tipos de errores.

El test estático encuentra **defectos** mientras que el test dinámico encuentra **fallas** o desviaciones en el resultado esperado. Las técnicas estáticas pueden realizarse antes de las dinámicas para encontrar defectos antes que se conviertan en fallas.

2. Introducción al testing manual

El testing manual es un tipo de prueba de software donde los testers ejecutan los casos de prueba sin usar ninguna herramienta de automatización.

La prueba manual es la más básica de todos los tipos de prueba y ayuda a encontrar errores en el sistema de software. Cualquier aplicación nueva debe probarse manualmente antes de que esta pueda automatizarse. La prueba manual requiere más esfuerzo, pero es necesaria para verificar la viabilidad de la automatización.

La prueba manual significa probar una aplicación manualmente por una persona para garantizar que un sitio web o una aplicación funcionen correctamente según las diversas condiciones que se escriben en los casos de prueba.

El objetivo de un tester de software es verificar el **diseño, la funcionalidad y el rendimiento** de la interfaz de usuario del sitio web haciendo clic en varios componentes, pero también es intentar **corromperlo** para ver si existe algún punto ciego que se pueda considerar como una vulnerabilidad.

Curso de testing funcional y accesibilidad web

Bloque #2

3. Objetivos de las pruebas manuales

- Garantizar que la aplicación esté libre de errores y que funcione de conformidad con los requisitos funcionales especificados.

- Garantizar que los conjuntos de pruebas o casos, estén diseñados durante la fase de prueba y que tengan una cobertura de prueba del 100%.

- Asegurar que los defectos informados sean reparados por los desarrolladores, y que los testers hayan realizado una nueva prueba en los defectos corregidos.

- Finalmente verificar la calidad del sistema y la entrega de un producto libre de errores al cliente.

4. Técnicas dinámicas

Las pruebas dinámicas se dividen en dos grandes categorías/grupos. La agrupación se realiza en función del carácter básico del método utilizado para obtener los casos de prueba.

La identificación de los diferentes tipos de defectos que se pueden encontrar en un sistema de software, requiere diferentes técnicas y tipos de prueba.

Para tener una buena cobertura en el sistema de software deberíamos aplicar las 3 técnicas (caja negra, caja blanca y basadas en experiencia), ya que cada una descubre defectos diferentes.

a) Técnicas de caja blanca

Las técnicas de caja blanca se realizan cuando el tester tiene **acceso al código fuente** de la aplicación, a sus algoritmos y estructuras de datos utilizadas. La implementación de este tipo de pruebas **requiere habilidades de programación**, un conocimiento del framework de desarrollo y un cierto conocimiento funcional que permita conocer qué misión tienen determinadas clases y métodos. *La estructura del programa es el foco.*

Entre los objetivos más importantes de ejecutar tests de caja blanca, están:

• Alcanzar código que no se alcanzó con las pruebas de caja negra.

• Encontrar inconsistencias en el código o código inútil.

Esta técnica se utiliza en:

• Nivel de componente: la estructura de un componente software, es decir sentencias, decisiones, ramas, distintos caminos.

• Nivel de integración: la estructura puede ser un árbol de llamada (un diagrama en el cual unos módulos llaman a otros módulos).

• Nivel de sistema: la estructura puede ser una estructura de un menú, un proceso de negocio o la estructura de una página web.

¿Cuál es su ventaja? La principal ventaja de aplicar técnicas de caja blanca es que los métodos pueden aplicarse en etapas temprana del sistema, y como hemos mencionado anteriormente, encontrar fallas en etapas tempranas hace que el costo de corregir esta falla es mucho menos que corregirlo en etapas finales, además tenemos una cobertura casi total de la estructura del sistema.

¿Qué desventaja presentan? La principal desventaja que tienen estas técnicas es que el tester o la persona que está probando debe tener conocimientos en lenguajes de programación.

b) Técnicas de caja negra

Esta estrategia de testing se centra en la verificación de las funcionalidades de la aplicación: datos que entran, resultados que se obtienen, interacción, funcionamiento de la interfaz de usuario y en general todo aquello que suponga estudiar el correcto comportamiento que se espera del sistema. No obstante, como el estudio de todas las posibles entradas y salidas de un programa sería impracticable se selecciona un conjunto de ellas sobre las que se realizan las pruebas.

Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, hay que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en nuestro sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos. Es necesario encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible.

Se observa el objeto de prueba como una caja. Aquí el sistema es tratado como un sistema cerrado del que se desconoce cómo fue desarrollado. Entonces, la diferencia fundamental respecto al testing de caja blanca es que en este caso no se trabaja con el código fuente sino con el programa.

Es la técnica más cercana a la experiencia del usuario. El principal objetivo es asegurar que el sistema funciona de acuerdo con los requerimientos y cumple las expectativas del usuario. También conocido como **Test Funcional o Test de comportamiento**.

El resultado de las pruebas de caja negra depende de la calidad de la especificación del sistema, por lo tanto, si las especificaciones son erróneas, también lo serán los casos de prueba.

¿Cuál es su ventaja?

• Que las pruebas se realizan desde un punto de vista del usuario buscando discrepancia con las especificaciones.

• Que el tester no necesita conocimiento de lenguajes de programación.

• Las pruebas pueden diseñarse a medida que se terminan las especificaciones, no hace falta esperar a que el desarrollar termine con el código.

¿Qué desventaja presentan?

• No hay forma de probar todos los caminos posibles en el sistema.

• Si las especificaciones no son claras o falta algo, los casos de pruebas no serán buenos o de buena cobertura.

b.1) Clase de equivalencia

Es el método que la mayoría de los testers hacen de forma intuitiva: dividimos los posibles valores en clases, mediante lo cual observamos valores de entrada de un programa y valores de salida.

La partición equivalente permite definir casos de pruebas que descubran clases de errores, reduciendo así el número total de casos de pruebas que hay que desarrollar. Entonces tenemos que con una mínima cantidad de casos de pruebas se puede esperar un valor de cobertura específico y además es aplicable a todos los niveles de prueba (componente, integración, sistema y de aceptación).

El rango de valores definido se agrupa en clases de equivalencia donde:

- todos los valores para los cuales se espera que el programa tenga un comportamiento común se agrupan en una clase de equivalencia (CE)

- Las clases de equivalencia pueden consistir en un rango de valores (por ejemplo, $0<x<10$) o en un valor aislado (por ejemplo, x = Verdadero).

Entonces, las clases de equivalencia se dividirán también en CE validas y CE inválidas. Entre las inválidas nos encontramos con valores que están fuera de rango y valores de formato incorrecto.

Por ejemplo:

Si mi valor x está definido como $0 \leq x \leq 10$ tendremos 3 clases de equivalencias:

$0 \leq x \leq 10$ - valores de entradas válidos

$x < 0$ - valores de entradas inválidos

$x > 10$ - valores de entradas inválidos

También podemos definir otras clases de equivalencias no validas, por ejemplo: que x no sea numérico o que no tenga un formato numérico admitido.

b.2) Análisis de valores límites

Permite ampliar o complementar la técnica anterior introduciendo una regla para seleccionar representantes. Probamos con más énfasis los valores límites de cada clase ya que frecuentemente no están bien definidos o implementados. Esta técnica también puede ser utilizada en todos los niveles de prueba.

En esta técnica, también seleccionamos representantes. Por lo tanto, vamos a tener 2 partes:

- Partición en clases de equivalencia: Evalúa un valor (representante) de la clase de equivalencia.

- Análisis de valores límite: Evalúa los valores límite y su entorno

b.3) Pruebas basadas en Casos de Uso

Los casos de prueba se obtienen directamente a partir de los casos de uso del objeto de prueba. El objeto de prueba es visto como un sistema que reacciona con actores. Un caso de uso describe la interacción de todos los actores involucrados. Todo caso de uso tiene precondiciones (que deben se cumplidas con el caso de prueba de forma satisfactoria) y poscondiciones (que describen el sistema tras la ejecución del caso de prueba.

Este tipo de pruebas, son más apropiadas para pruebas de aceptación y pruebas de sistema, dado que cada caso de uso describe un escenario de usuario a probar.

En un sistema Biblioteca, tendremos al usuario y al bibliotecario:

La "caja" donde están los casos de uso define los límites del sistema de biblioteca es decir, los casos de uso representados son parte del sistema a modelar pero no los actores. También tenemos que todos estos elementos descriptivos son utilizados para definir los casos de prueba correspondientes. Entonces, a partir de un caso de uso voy a tener como mínimo un caso de prueba.

La cantidad de casos de prueba derivados de cada caso de uso va a depender de la complejidad del caso de uso. Cada caso de uso puede es utilizado como la fuente para un caso de prueba.

Curso de testing funcional y accesibilidad web

Bloque #2

c) Basados en Experiencias

El conocimiento de las personas involucradas es importante para generar casos de pruebas. Es un método complementario para los demás. Las preguntas aquí serían: ¿Dónde se han acumulado errores en el pasado? ¿Dónde falla normalmente el software? Por lo tanto el tester utilizará la predicción de errores y las pruebas exploratorias para la creación de casos de prueba. Es un método complementario para los demás.

Vamos a utilizar:

• Intuición: Conocimiento de Testers, usuarios, personas relacionadas con el software sobre su uso y el ambiente.

• Experiencia: Conocimiento de errores pasados en su distribución

• Pruebas exploratorias: pruebas iterativas basadas en el conocimiento adquirido respecto del sistema. Es muy útil cuando el tiempo disponible para pruebas es escaso.

d) **¿Cómo seleccionar la técnica correcta?**

¿Cómo sabemos que técnica de especificación elegir para crear los casos de prueba? Esto depende de muchos factores, entre ellos:

- Tipo de sistema: ¿Tenemos disponible el código fuente? ¿Tenemos personas que comprenden el código fuente para hacer casos de pruebas? ¿Tenemos buenas especificaciones para casos de prueba de caja negra o solo exploratorias?

- Requerimientos: ¿Tenemos requerimientos especificados? ¿Bien especificados? ¿Disponemos del cliente disponible para respondernos preguntas?

- Niveles de riesgo: ¿El objeto de prueba es muy usado? ¿Hay algún estándar contractual sobre la ejecución de pruebas que tiene que ser cumplido? ¿Cuánto tiempo disponemos? ¿Tenemos que entregarlo pronto?

- Objetivo del Test: ¿Qué pruebas no funcionales son necesarias? ¿Las funcionales han sido requeridas formalmente? ¿Qué debe ser probado sí o sí?

- Documentación: ¿Disponemos de documentación importante, como documentación sobre el negocio, documentación de usuario?

- Conocimiento del Tester/s: ¿Qué conocimiento tiene el tester? ¿Tiene conocimiento de código fuente? ¿Tiene conocimiento de técnicas de pruebas? ¿Es principiante?

- Tiempo y presupuesto: ¿Cuánto tiempo tenemos?

¡Felicitaciones! ¡Llegaste al final de la clase! ¡Nos encontramos en el foro!