UNIDAD 2

FUNDAMENTOS DEL TESTING

2.1 INTRODUCCIÓN

Veremos cómo crear planes de prueba, como diseñar los casos de prueba y como reportarlos. Veremos algunos ejemplos donde se va a tener la oportunidad de practicar con sitios web que tenemos en internet de manera que sea más práctico, entretenido e interactivo. Inicialmente recordar que cuando vamos a planificar las pruebas de software, es importante tener en cuenta los principios del software testing.

Este es un tema que se evalúa en la certificación de software testing como lo es FOUNDATION LEVEL DE LA FUNDACION ISTQB que significaría **International Software Testing Cualificación Board** que es como la entidad que certifica software testing a nivel mundial que recomiendo hacerla si se quieren profesionalizar como testers.

empezar a ver el detalle cada uno de sus principios con algunos ejemplos.

2.2 PRICIPIOS DEL SOFTWARE TESTING

2.2.1 **Ejecutar pruebas nos muestra la presencia de defectos, pero no puede probar que no los hay.** ¿Y qué significa esto?

Aunque se realice una cantidad considerable de pruebas, no es posible asegurar que un sistema se encuentre libre de defectos. Bueno, para la muestra tenemos el Samsung Galaxy Note, que fue liberado por la empresa Samsung en el año 2016 y que tuvo que ser retirado del mercado, porque se incendiaba tanto en reposo como en uso. Entonces, Samsung, siendo una empresa tan importante que tuvo suficiente tiempo para probar este dispositivo tanto en su hardware como en su software, con personas que estoy seguro muy profesionales pudieron haber encontrado los defectos, sin embargo, el dispositivo salió con errores cuando se liberó al mercado.

Entonces, esto es un ejemplo del principio número 1 que las pruebas nos muestran que hubo objetos, pero no pueden probar que no los hay.

2.2.2 **El testing exhaustivo es imposible.** ¿Qué quiere decir esto?

Probar todas las combinaciones de entrada y precondiciones no es posible, excepto en casos triviales o como muy fáciles, como un formulario con dos campos o más o tres campos. Es muy fácil probarlos, pero imaginémonos que es un formulario con muchos campos, demasiadas las combinaciones que habría que hacer de casos de prueba es casi que imposible, debido a esa situación, lo correcto es realizar un análisis de riesgo y así priorizar y ejecutar los casos de prueba que tienen mayor importancia.

Por ejemplo, el caso de la página de **https://weather.com/**en una página que muestra el estado del tiempo a nivel mundial, de todas las ciudades del mundo, y si miramos en el planeta tierra, en el mundo hay 195 países y cada uno de los países tiene muchas ciudades. Si hacemos el cálculo rápido, da más o menos. ***quinientas mil ciudades*** las que hay en el mundo y nos ponen a probar este software, sería casi imposible probar con cada una de las ciudades cómo se comporta el clima y convivir con sus diferentes combinaciones. Entonces esa prueba sería muy larga, muy costosa.

Por esta razón, en lugar de probar excesivamente esta página del clima es mejor hacer un análisis de riesgos, utilizar algunas técnicas de pruebas y establecer prioridades para enfocar los esfuerzos de las pruebas.

2.2.3 **La verificación de la calidad de un sistema debe empezarse lo antes posible.** ¿Qué quiere decir esto?

Las pruebas de software deben empezarse lo más temprano posible durante el ciclo de vida del desarrollo del software, ***en el control de eventos en etapas tempranas del proyecto.***

Equivale a raíz costo de tiempo y dinero. Mientras más tarde se encuentra defectos, más costoso es arreglarlos, así como la imagen que tenemos. Este bicho representa los bugs, a medida que el **bug** se va encontrando en una etapa posterior el desarrollo del software se va volviendo más costoso.

En el eje X tenemos especificaciones, diseño, codificación, pruebas y la liberación, y a medida que el software va cambiando de etapa se va volviendo más grande y en el eje G tenemos el costo de en arreglare de bag en el tiempo.

Es mejor encontrarlo durante las especificaciones, durante la negociación de los requisitos del software o al menos durante la codificación, cuando el software, cuando el bag se encuentra durante la etapa de desarrollo, es mucho más barato de corregir.

.

Hay una investigación muy buena que hizo IBM y dice que el costo de eliminación de un defecto aumenta con el tiempo y, por ejemplo, un defecto encontrado en producción vale 30 veces más que encontrado en etapa de diseño.

2.2.4 La mayoría de efectos relevantes suelen concentrarse en un grupo muy determinado de módulos de nuestro producto y existen distintas razones por la que esto sucede**.**

Una de ellas son los cambios se hacen regularmente en un solo módulo o en una sola opción del sistema que estás probando. O un módulo o una página o sitio en una aplicación móvil en una de las opciones de la aplicación **puede que sea una opción muy crítica** por ahí en la página web o en la aplicación móvil, o donde se concentra toda la lógica de la aplicación.

Entonces, al realizar una opción muy importante, ahí es donde se hacen la mayoría de cambios y en con la introducción de nuevos cambios hay una probabilidad alta de que se introduzcan errores en la programación. Si miramos un ejemplo, imaginemos que tenemos un sistema de reservas en línea de una aerolínea y los defectos se tienden a encontrar en uno o dos, dos de las opciones dentro de sistema de reservas en línea.

Este fenómeno está relacionado con la regla 80/20, llamado el **principio de Pareto**, que nos indica que el 80 por ciento de los defectos se encuentra en el 20 por ciento del sistema que se está probando por los y por cual, si queremos descubrir una mayor cantidad de bugs, debemos enfocarnos en estos módulos que son más importantes o en un buen análisis de riesgos y en esas funcionalidades que son más importantes. Eso no quiere decir que dejemos de ignoremos las demás, pero siguiendo el principio de Pareto, la mayoría de los **bugs** que van a encontrar en esos 20 por ciento el sistema que se está probando.

2.2.5 La paradoja del pesticida

Cuando ejecutan los mismos casos de prueba una y otra vez y sin ningún cambio. Eventualmente estos dejarán de encontrar defectos nuevos, o sea, nuevos bugs. Es necesario que cambiemos las pruebas y los datos de prueba para poder encontrar nuevos defectos.

En la imagen tenemos el software tester en esta persona que está rociando con pesticida sobre un insecto que sería el insecto sería el bug y explica claramente que pasa con el bug y como no le hace ningún efecto el pesticida. Entonces cuando usamos este mismo pesticida a los mismos insectos, estos se vuelven resistentes y no tienen el mismo efecto para detectar uno de los bugs.

2.2.6 El testing es totalmente dependiente del contexto

Es necesario adecuar las pruebas según el contexto del objeto que se está probando. Por ejemplo, no es igual probar una aplicación web para uso médico que maneja datos muy confidenciales a una aplicación móvil de diversión o un sistema bancario, son dos aplicaciones o dos sistemas totalmente distintos, y la gente tiene que adecuarse a ese contexto. El riesgo es un factor crítico a definir para las pruebas. Entonces hay que hacer un análisis de riesgos, pensar que hay más riesgo cuando se trata de vidas humanas, pérdida económica, etc. y por consiguiente, se deben ejecutar más pruebas o unas pruebas con mayor rigor.

2.2.7 **La falacia (mentira) de ausencia errores.** ¿Qué quiere decir esto?

Es posible que la mayoría de los defectos críticos de una aplicación hayan sido corregidos. Pero esto no asegura que el software sea exitoso. Por ejemplo, podríamos probar todos los requisitos del sistema de una aplicación de comercio electrónico, una página que vende productos que podemos probar todos los casos y de acuerdo a las especificaciones de del negocio.

Digamos que encontramos los errores y los desarrolladores los probaron, los arreglaron, nosotros lo probamos nuevamente y todo ya funciona perfectamente. Aun así, el software que vamos a liberar puede que sea difícil de usar o que en realidad ese sistema sea ineficiente o no se ajuste a la necesidad del mercado o que el sistema tenga una calidad inferior a los sistemas de la competencia.

Entonces eso es un ejemplo claro de la falacia de errores. No quiere decir que porque nosotros proveemos mucho el sistema va a tener alta calidad.