Curso de testing funcional y accesibilidad web

Bloque #2

<u>Clase 1 – Segunda parte</u>

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

1. ¿Cómo crear casos de prueba?

- Hay que conocer la aplicación: Para esto, necesitamos obtener la mayor cantidad de información posible (requerimientos, casos de uso, guías de usuario, tutoriales, usar nosotros mismos el sistema). Y una vez que contamos con esto, haremos una lista de las funcionalidades.

- Estandarizar la forma de trabajo: Usando diferentes archivos para casos de prueba de diferentes funcionalidades, usando Excel o alguna herramienta. Estos archivos de casos de pruebas se deberían nombrar de acuerdo a la funcionalidad en la que se están basando.

- Identificar conjuntos de casos de pruebas (test suites). Esto se logra buscando conjuntos de casos de pruebas relacionados por funcionalidad, roles, o integración de ellos. Ciertos casos de pruebas específicos es mejor separarlos para obtener una mejor organización de los mismos, por ejemplo, casos de prueba específicos a navegadores, o sistemas operativos, de usabilidad, de interfaz, etc.

- Estructurar los casos de prueba. Cada caso de prueba contiene una cantidad de datos que permite determinar si una aplicación está funcionando correctamente. Entre ellos:

- Nombre de la prueba: el nombre será dado en base al estándar definido en los puntos anteriores. El nombre debe ser único. Por ejemplo: Eliminar_Cliente.

- Descripción: es el objetivo que se quiere alcanzar. Ejemplo: Verificar que el sistema solicita confirmación para la eliminación de un registro. Debe estar alineado con el resultado que estamos esperando.

- Precondiciones: aquellas situaciones que deben cumplirse para que el Caso de Prueba se pueda ejecutar.

  Ejemplo: Para eliminar un registro, el usuario debe tener los permisos específicos sobre la aplicación. Pasos: Secuencia de pasos a seguir para realizar la prueba y alcanzar el objetivo. Ejemplo: Realizar una búsqueda, seleccionar un registro, presionar el botón Eliminar

- Resultado esperado: Estado que se espera luego de ejecutar los pasos del caso, los cambios que el usuario ve y los de Base de datos también. Ejemplo: Mensaje de confirmación de eliminación.

- Resultado actual: Respuesta que se obtiene luego de ejecutar los pasos.

- Estado: Los estados los obtendremos luego de correr los casos de prueba. No hay una definición exacta de cuáles son los estados correctos, cada proyecto o herramienta contiene sus propios nombres, pero en general son:

  * Pasó = Los resultados esperado y actual coinciden

  * Falló = Los resultados esperado y actual no coinciden

  * No probado = No se ha ejecutado

  * No aplica = Puede ser cuando el requerimiento ha cambiado

- Comentarios: Cualquier información necesaria, una determinada condición.

- Dependencias: Orden de ejecución de casos de prueba, como vimos previamente los casos de pruebas se unen en conjuntos o test suite, allí estará definido qué casos de pruebas se ejecutan primero y cuales después.

- Referencias: Muestra a que requerimiento / caso de uso correspondeCreación de casos de prueba.

2. Ciclo de vida de un defecto

Antes que nada, repasemos el concepto de defecto. Un defecto o bug, es una condición por la cual el producto de software no cumple con alguno de los requerimientos (o especificación) o expectativas del usuario final. Básicamente, un defecto es un error en el código o la lógica que causa un mal funcionamiento en el sistema. Un defecto es lo que encontramos luego de ejecutar un caso de prueba.

El ciclo de vida de un defecto es el ciclo en el cual el defecto atraviesa durante su vida. Comienza cuando el defecto es encontrado, es decir cuando hemos corrido un caso de prueba y hemos encontrado un defecto y va a terminar cuando es cerrado, luego de asegurarnos de que no puede ser reproducido de nuevo.

Dependiendo de si utilizamos alguna herramienta de seguimiento de defectos, podemos encontrarnos con una gran variedad de nombres para cada etapa del defecto. Es importante definir apropiadamente el ciclo de vida de un defecto, y añadirlo como un proceso más a la estrategia de pruebas. Por ejemplo, todo defecto, como mínimo, pasa por los siguientes estados:

- Abierto o Nuevo: El tester encuentra un bug que no había sido detectado anteriormente y lo reporta.
- Asignado: El defecto ha sido asignado a un desarrollador.
- Arreglado: El desarrollador ha resuelto el problema.
- Verificado o Cerrado: Una vez que el tester ha verificado que el defecto ha desaparecido, entonces el defecto se cierra.

- Rechazado: Una vez que se ha encontrado el defecto y éste es revisado, si el desarrollador o el analista o
- quien corresponda considera que no es un defecto, puede llegar a pasar al estado rechazado.
- Reabierto: El tester lo ubica en este estado cuando luego de ser arreglado, el defecto puede ser reproducido.

Para que la comunicación entre el equipo de desarrollo y el de pruebas sea correcto, es necesario reportar correctamente los defectos que se encuentran durante la ejecución de los casos de prueba. Para esto, tener en cuenta estas buenas prácticas:

• Asegurarse que realmente es un defecto, por ejemplo, que no sea problema de nuestra configuración.

• Revisar que no haya sido creado anteriormente: en este caso podemos agregar un comentario con más información.

• Asegurarse que la versión del sistema que estamos utilizando sea la última, así no corremos riesgo de que haya sido ya corregido el problema.

3. Cómo reportar un defecto

Reportar correctamente un defecto facilita a cualquier persona del proyecto a entender cuál es la falla.

Debemos tener en cuenta estas *"buenas prácticas"*:

- Descripción detallada: Se describe el escenario, cuál era el resultado esperado y cuál fue el resultado real, mencionar el caso de prueba que originó el defecto.

- Pasos para reproducir: si el desarrollador no puede seguir los pasos, no podrá corregir el defecto, es por este motivo que los pasos son la parte más importante del reporte.

- Defecto específico: Si se han encontrado 2 problemas, aunque sean similares, se crean 2 defectos diferentes. Si el desarrollador descubre que la causa de los dos defectos es la misma, nos avisará, de lo contrario, es mejor tenerlos separados porque así el desarrollador sabe cuál es la causa de uno y cuál es la causa del otro.

- Entorno: Se necesita brindar toda la información posible del escenario en el que estamos (sistema operativo, navegador y versión, versión del sistema) .

- Un buen título: El título del defecto debe ser especifico, brindando una especie de sumario de lo que contiene.

- Datos propios: En caso de que el desarrollador no comprenda alguna parte del problema, es necesario que pregunte a la persona que creó el defecto.

- Severidad: Dependiendo de la severidad que le asignemos al defecto, puede que el mismo sea corregido pronto o pasen varios días / meses.

Curso de testing funcional y accesibilidad web

Bloque #2

4. Severidad y prioridad

Un error que cometemos frecuentemente es no diferenciar la prioridad de la severidad, cuando en realidad son dos conceptos muy diferentes.

- Severidad

  La severidad de los defectos depende de las funcionalidades bajo prueba. Se define de acuerdo con en qué medida afecta el defecto al sistema en general.

  Si no afecta las funcionalidades, entonces la severidad será trivial (Defectos Cosméticos), Menor (Defecto nivel interfaces) o Normal (Cualquier otro defecto que no interrumpa el uso de la aplicación).

  En cambio, si afecta las funcionalidades podrá ser Mayor (si las funcionalidades tienen un resultado erróneo o imprevisto) o Urgente (si la aplicación no es utilizable).

- Prioridad

  En cambio, la prioridad indica qué tan importante es para el negocio o el cliente que se corrija este defecto. Aquí nos vamos a preguntar cuán importante es para el negocio es que se arregle este defecto.

  Habrá casos en que los defectos tendrán alta severidad pero prioridad baja, o al revés, defectos con baja severidad pero con alta prioridad.

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