Anotaciones sobre Testing

**Tipos de metodología:**

**Metodologías secuenciales:**

Son metodologías o ciclo de vida de desarrollo de software las cuales son secuenciales, es decir, cada etapa requiere de que todas las anteriores sean cumplidas y se ejecutan una después de otra, tienen como característica principal que en ellas solo hay UNA SOLA ENTREGA al cliente, la cual se le hace de una vez por todas cuando el software ya esta totalmente terminado.

**Ventajas:**

* Es fácil de entender

**Desventajas:**

* Al hacerse una SOLA ENTREGA cuando el Software ya esta completamente terminado, si al cliente no le gusta o se debe hacer algún cambio, se deberá repetir el proceso desde 0, lo cual sería muy costoso y trabajoso.
* Al hacerse la entrega solamente cuando el Software ya este completamente terminado, el cliente tarda más tiempo en ver resultados.
* No se obtiene Feedback Continuo del cliente, o incluso no se obtiene Feedback alguno hasta que ya el software está terminado.

**Uso optimo:**

* Cuando tenemos garantías y 100% asegurado que no habrá NINGÚN TIPO DE CAMBIO en los requerimientos

**Waterfall o Cascada:** Es una metodología secuencial en la cual los pasos son secuenciales y no hay ningún tipo de solapamiento entre las etapas, los pasos de Testing no se ejecutan en paralelo con los pasos de Development. Consta de las siguientes etapas:

Requerimientos -> Diseño -> Desarrollo -> Testing -> Despliegue -> Mantenimiento

**V-Model:** Es una metodología secuencial en la cual aunque las etapas también son secuenciales, los pasos de Testing se ejecutan en paralelo con otras etapas:

Requerimientos <------------------------------------------------ > Pruebas de aceptación

Especificaciones <------------------------------------------ > Pruebas de sistema

Diseño arquitectónico <------------------------------ > Pruebas de integración

Diseño detallado <------------------- > Pruebas unitarias

Desarrollo o codificación

Para cada etapa se desarrolla escenarios de pruebas, las cuales van por orden de complejidad, de la más sencilla a la más compleja:

**Pruebas unitarias:** Son pruebas de unidades pequeñas de código como funciones o clases, en el contexto de las pruebas se puede decir que es cuando probamos una función muy especifica como por ejemplo verificar la validez de una contraseña en un registro de usuarios.

**Pruebas de integración:** Son pruebas en las cuales verificamos que distintos componentes del software (Base de Datos, UI, o incluso otros módulos o componentes) trabajan en conjunto de manera correcta. Hay dos tipos de pruebas de integración:

* **De Componentes:** Es generalmente hecha por un desarrollador, en la cual prueba que los diferentes componentes o porciones de código que programo funcionen correctamente en conjunto.
* **De Sistemas:** Es generalmente hecha por el tester, en la cual prueba el frontend, el backend y la base de datos, los cuales, en este contexto, cada uno de ellos son considerados un sistema.

**Pruebas funcionales:** Son pruebas en las cuales verificamos que las funcionalidades funcionen de manera correcta desde la perspectiva de un usuario final.

**Pruebas de sistema:** Son pruebas en las cuales verificamos el funcionamiento del sistema como un todo, tratando de encontrar la mayor cantidad de defectos posibles. Este es el tipo de pruebas más importante de las mencionadas ya que aquí es donde esta la mayor parte del trabajo de un QA Tester. Aquí es donde verificaremos el funcionamiento del sistema “End To End” es decir, al completo, desde una perspectiva similar a la de un usuario final, ya que el sistema ya debería poder generar información que será utilizada por el usuario.

**Pruebas de aceptación:** Son pruebas en las que nos aseguramos que el sistema esta listo para el despliegue, verificando que el rendimiento del software es el proceso. Mientras que las pruebas de sistema están enfocadas en encontrar bugs o errores, las pruebas de aceptación están enfocadas en el rendimiento del sistema como un todo. Hay ciertos tipos de pruebas de aceptación que involucran a los clientes para que utilicen el software mientras aún esta en fase de pruebas (versiones Alpha y Beta).

**Metodologías ágiles:**

**Scrum:** Es una metodología ágil organizada por sprints, en la cual tenemos un Scrum Team formado por el Product Owner (que sustituye al Business Analyst en las metodologías secuenciales), el Scrum Master y los Developers (todo el equipo que no forma parte de los dos roles anteriores).

**Stakeholders:** Son los interesados en el proyecto, como los usuarios finales o los clientes.

**Product Owner:** Es el representante de los stakeholders dentro del equipo, escribe los ítems o user stories del Product Backlog y es el único con autoridad para cancelar o cambiar un sprint.

**Scrum master:** Es el “coach” o “manager” de un equipo Scrum, que gestiona el equipo.

**Developers:** Es el resto del equipo los cuales participan en todo el proceso de llevar el proyecto a la realidad.

A grandes rasgos, Scrum consiste en tomar un Product Backlog (lista de requerimientos), seleccionar algunos ítems de esta, y en base a estos ítems seleccionados trabajar el “mini-proyecto” en un Sprint, que es una ventana de tiempo que va desde 2 a 4 semanas, en la que al final de esta se entregará un incremento o entregable al cliente con funcionalidad parcial que alcanza el Minimo Producto Viable, gracias a lo cual podrá dar feedback y en caso de que se requiera algún cambio no tener que empezar todo el proceso desde cero.

A grandes rasgos, Scrum funciona así:

Requerimientos (Product Backlog)

|

Seleccionar los ítems o user stories más prioritarios (Sprint backlog)ç

|

Planeación del Sprint

|

Sprint (De 2 a 4 Semanas)

Daily Scrums (Reuniones de máximo 15 minutos diarias para saber que ha hecho ayer, que hará hoy y los problemas que ha tenido cada developer)

|

Review del Sprint (8 horas máximo) para analizar el resultado del Sprint y presentar el entregable a los stakeholders

|

Sprint Retrospective para buscar maneras de incrementar la efectividad y calidad en los Sprints futuros

Una vez terminado un Sprint el proceso se repite inmediatamente.

**Modelos de desarrollo de Software:**

**Secuencial:** Es el modelo que ya vimos anteriormente, cada etapa va después de la anterior y se entrega al cliente una sola vez: Al final del proyecto.

**Incremental:** Se entrega al cliente pequeñas funcionalidades especificas continuamente, sin embargo, por sí solas no son útiles o usables. Por ejemplo, se entrega la funcionalidad de cargar fotos, otra de registrar usuario, etc.

**Iterativa:** Se entrega al cliente un “mini-software” completo, que requirió un ciclo de vida de desarrollo completo, se entrega una aplicación “completa” solo con las funcionalidades más críticas, es el modelo usado por las metodologías ágiles como Scrum. Una vez entregada, pues basado en el feedback recibido, se incrementa la aplicación añadiéndole mejoras y las funcionalidades faltantes. Un ejemplo práctico de esto, por así decirlo, son las diferentes versiones o fases de los juegos: Fase Alpha, Fase Beta, etc.

**Proceso de Pruebas (Test Process):**

**Planeación:** Escribimos nuestro plan de pruebas

**Monitoreo y control:** Verificamos nuestro progreso esperado con el progreso real que hemos tenido según el plan.

**Análisis:** Analizamos los requerimientos para escribir nuestros escenarios de prueba y nuestros casos de prueba.

* **Casos de pruebas:** Es una descripción detallada, paso por paso y diciéndonos COMO debemos probar lo que debemos, es más detallado que los escenarios de prueba y requieren más tiempo para ejecutarse. Se usa generalmente cuando tenemos más tiempo para ejecutarlos.

Ejemplo explicado aquí: https://www.youtube.com/watch?v=Yg9P2KnUiEM

* **Escenarios de pruebas:** Son una descripción simple y general de lo QUE vamos a probar, generalmente se escriben en una oración simple. Un escenario de prueba es la descripción de una serie de casos de prueba relacionados a una funcionalidad. Se usa generalmente cuando tenemos poco tiempo, como cuando trabajas con metodología agiles, cuando las funcionalidades cambian mucho y cuando ya estas ambientado en el proyecto.

Como dijimos antes, suelen requerir o conformarse por varios casos de prueba:

Ejemplo: Hacer una búsqueda de un producto en MercadoLibre

Para esto, primero debemos verificar una serie de casos o sub-funcionalidades que el cliente esperaría al hacer una búsqueda en Mercado Libre:

* 1. Poder entrar a la página principal
  2. Ver una barra de búsqueda
  3. Obtener sugerencias de búsqueda al hacer clic en la barra de búsqueda
  4. Al hacer clic en el botón de buscar obtener un listado de productos que sean lo que se busco en la barra de búsqueda.

**Diseño:** Diseñamos nuestros casos de pruebas.

**Implementación:** Organizamos nuestros casos de prueba en “Test Suites”, diseñamos nuestro ambiente de pruebas en el cual ejecutaremos las pruebas (por ejemplo, un emulador, un navegador como Chrome o Bing, un Sistema Operativo como Windows o Android, etc), SE DEBE INTENTAR HACER LAS PRUEBAS EN UNA AMBIENTE IGUAL O AL MENOS SIMILAR AL AMBIENTE DE PRODUCCIÓN, ES DECIR, SI EL CLIENTE UTILIZARÁ EL SOFTWARE EN WINDOWS, SE DEBERÍAN HACER LAS PRUEBAS EN WINDOWS, SI USARA EL SOFTWARE EN UN CELULAR DE 7 PULGADAS CON ANDROID 13.5 SE DEBERÍAN HACER LAS PRUEBAS EN UN CELULAR O EMULADOR DE ANDROID 13.5 CON 7 PULGADAS DE PANTALLA.

* **Test Suites:** Son colecciones organizadas de casos de prueba que se agrupan y se ejecutan de forma conjunta para probar una funcionalidad, componente o módulo de un software. Se diferencian de los escenarios de prueba en que los escenarios son simplemente descripciones de alto nivel de una situación especifica que se debe probar.

**Ejecución:** Ejecutamos los distintos tipos de pruebas que necesitemos ejecutar (pruebas de rendimiento, regresión, etc.), si encontramos defectos los reportamos

**Terminación:** Terminamos el trabajo escribiendo reportes de progreso de pruebas y reportes de resúmenes de pruebas.

**Tipos de pruebas:**

Los dos tipos de pruebas principales son:

**Según funcionalidad:**

**Funcionales:** Son las pruebas MÁS COMUNES, en las que verificamos y nos aseguramos de que el software cumple con las funcionalidades esperadas, por ejemplo, poder registrarme e iniciar sesión en un software, y entrar a un perfil que contenga fotos que solo yo y mis contactos podamos ver.

Se pueden resumir en una respuesta de Sí o No: ¿Cumple el software con X requerimiento? ¿Sí o No?

**No Funcionales:** Son las pruebas en las que verificamos otros aspectos al margen de la funcionalidad, generalmente no son preguntas que se pueden responder con un sí o un no, sino que las pruebas arrojan ciertos resultados, que en base al tipo de software que estemos construyendo y a los requerimientos entonces interpretaremos y podemos decir si los resultados son buenos o malos, las pruebas no funcionales cubren aspectos como:

* **Seguridad:** Son pruebas de seguridad del software, que tantas vulnerabilidades tiene, etc.
* **Rendimiento:** Son pruebas del rendimiento del software, como los recursos que consume para hacer alguna operación o por ejemplo, el tiempo de respuesta de una aplicación, el estándar aceptado puede variar de acuerdo al tipo de aplicación que estemos desarrollando, ya que por ejemplo, 1 segundo de respuesta puede ser bueno para una funcionalidad o aplicación especifica como una red social, pero ese mismo tiempo podría ser un tiempo de respuesta pésimo para por ejemplo un chat en tiempo real o un videojuego online.
* **Usabilidad:** Son pruebas de que tan fácil de utilizar es el software, que tan atractiva e intuitiva es el UI, etc.
* **Compatibilidad:** Son pruebas a la capacidad del software de funcionar adecuadamente en diferentes entornos como en los diferentes navegadores existentes (Chrome, Bing, Firefox, etc.), sistemas operativos, dispositivos y configuraciones.

**Según el conocimiento del código:**

**Black Box Testing:** Son pruebas en las que no tenemos ningún conocimiento de cómo funciona el código del módulo o funcionalidad que estamos probando, simplemente ejecutamos la prueba, y sabemos las entradas y salidas que da el software.

**White Box Testing:** Son pruebas en las que se nos ha proporcionado el código del software o de la parte del software que estamos probando, realizamos las pruebas mirando la estructura interna del código, que pasa en el código, la API y en la base de datos mientras realizamos las pruebas. En este tipo de pruebas requerimos saber o entender el lenguaje de programación en el cual está escrito el software que estamos probando.

**Según la ejecución del programa:**

**Dynamic Testing:** Son las pruebas en las cuales ejecutamos el programa al hacer las pruebas. Es decir, este tipo de pruebas incluyen la ejecución del software.

**Static Testing:** Son las pruebas las cuales no incluyen la ejecución del software, es decir, es todo lo que hacemos fuera de la ejecución de las pruebas, como por ejemplo revisar las historias de usuario, los diseños del UI, revisar código, etc.

**Según el momento:**

**Testing:** Son las pruebas que ejecutamos inicialmente.

**Re-Testing o Confirmation Testing:** Son las pruebas que ejecutamos una vez ya ejecutamos una prueba de algún aspecto del software previamente y esta no cumplió satisfactoriamente con lo esperado o se encontraron defectos, por lo cual se devolvió a los desarrolladores para que lo reparasen. Al ejecutarse nuevamente pruebas sobre la misma funcionalidad o componente se esta haciendo un Re-Testing para asegurarse que esta vez si se cumple con lo esperado.

**Regression Testing:** Son las pruebas que hacemos sobre toda la aplicación después de hacer algún cambio en el software como agregar alguna nueva funcionalidad, remover algo o actualizarlo. Este tipo de prueba se ejecuta para asegurarse que al agregar o hacer algún cambio en el software no se rompieron cosas que ya funcionaban.

**Smoke Testing:** Son pruebas en las que probamos rápida y superficialmente el software, probando las funcionalidades críticas o principales del software (como el inicio o arranque de la aplicación, la navegación por las diferentes vistas o páginas de la aplicación y las funcionalidades fundamentales) y buscando errores críticos y graves que impidan que el software sea útil o funcional. En caso de que el software pase la prueba de humo, se procede con otras pruebas más detalladas y minuciosas, en caso de que el software no pase la prueba de humo, se reportan los errores encontrados para que sean reparados.