Tipos de pruebas de Software

En este documento vamos a introducirnos en una de las fases más importantes del desarrollo de software, la fase de pruebas. Introduciremos los diferentes tipos de pruebas que se pueden realizar y más adelante profundizaremos en ellas.

Las pruebas de software dentro del desarrollo de un sistema

El desarrollo de software sigue una serie de fases para garantizar que el producto final sea funcional, eficiente y libre de errores. Una de las fases fundamentales es la **prueba y depuración del software**, donde se verifica que el código cumple con los requisitos y funciona correctamente en diferentes escenarios.

Las pruebas de software permiten detectar errores antes de que el sistema llegue a producción, reduciendo los costos y riesgos asociados a fallos en entornos reales. Para ello, existen diversas metodologías y enfoques que buscan garantizar la calidad del software en diferentes niveles.

Las pruebas de software son una parte fundamental del desarrollo, ya que permiten garantizar la calidad, seguridad y funcionalidad de una aplicación antes de que llegue a los usuarios finales. Un error no detectado a tiempo puede provocar fallos graves, pérdidas económicas y problemas de reputación para una empresa. Es por ello que el proceso de testeo es clave en cualquier proyecto de software.

Objetivos y metodologías de testeo

El objetivo principal del testeo de software es **detectar y corregir errores** para asegurar la calidad del producto final. Además, el testeo permite verificar que el software cumple con los requisitos funcionales y no funcionales establecidos en la fase de diseño.

Para alcanzar estos objetivos, se emplean diferentes metodologías de prueba:

- **Pruebas de caja blanca**: Se basan en el conocimiento interno del código y su estructura. Se analizan las rutas de ejecución, estructuras de control y dependencias internas del programa.
- **Pruebas de caja negra**: Se enfocan en evaluar la funcionalidad del software sin conocer su implementación interna. Se prueban los casos de entrada y salida para validar que el sistema responde correctamente.
- **Pruebas manuales y pruebas automatizadas**: Las pruebas pueden realizarse de forma manual, con la intervención de testers que evalúan el sistema, o de forma automatizada, usando herramientas como JUnit para ejecutar test de manera programada.

Etapas del testeo

El proceso de testeo se realiza en varias etapas, desde las pruebas iniciales en el código hasta la validación final del producto:

1. **Pruebas de unidad**: Se prueban componentes individuales del código, como funciones o clases, para verificar que funcionan de manera aislada.
2. **Pruebas de integración**: Se verifica la interacción entre módulos o componentes del software para asegurar su correcto funcionamiento en conjunto.
3. **Pruebas de sistema**: Se prueba el software en su totalidad, evaluando su comportamiento en diferentes escenarios y condiciones.
4. **Pruebas de aceptación y validación**: Se comprueba que el software cumple con los requisitos del cliente y está listo para su despliegue.

Pruebas de unidad

Las **pruebas de unidad** son el nivel más básico de testeo y se centran en evaluar el correcto funcionamiento de componentes individuales del código, como funciones, métodos o clases. Estas pruebas se realizan de manera aislada, sin depender de otros módulos del sistema, lo que permite detectar errores en partes específicas del código antes de integrarlas con el resto del software.

Características clave

- Se enfocan en probar unidades pequeñas e independientes.
- Se ejecutan de forma rápida y frecuente.
- Permiten identificar errores en etapas tempranas del desarrollo.
- Son generalmente automatizadas con herramientas de testing.

En **Java**, la herramienta más utilizada para realizar pruebas de unidad es **JUnit**, que permite definir pruebas automatizadas y ejecutarlas repetidamente. Un ejemplo básico de test con JUnit sería:

```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class CalculadoraTest {
    @Test
    public void testSuma() {
        Calculadora calc = new Calculadora();
        assertEquals(10, calc.sumar(5, 5));
    }
}
```

En este caso, el test verifica que el método `sumar()` de la clase `Calculadora` devuelve el valor esperado. Si el resultado no coincide, la prueba fallará, indicando un error en la implementación.

Pruebas de integración

Las **pruebas de integración** verifican la interacción entre diferentes módulos o componentes del software. Aunque cada unidad puede haber pasado sus pruebas individuales, es fundamental asegurarse de que, al trabajar juntas, se comuniquen correctamente y no generen errores inesperados.

Características clave

- Se enfocan en la interacción entre componentes.
- Pueden incluir bases de datos, API externas y otros módulos del sistema.
- Se realizan después de las pruebas de unidad.
- Ayudan a detectar problemas de compatibilidad y dependencias.

Tipos de pruebas de integración

- **Big Bang**: Se integran todos los módulos a la vez y se prueba el sistema completo.
- **Top-down**: Se prueban primero los módulos de alto nivel y luego los de menor nivel.
- **Bottom-up**: Se prueban primero los módulos de bajo nivel y luego los de mayor nivel.
- **Sandwich (híbrido)**: Mezcla de top-down y bottom-up.

En Java, herramientas como **Mockito** permiten simular dependencias y realizar pruebas de integración sin necesidad de depender de servicios externos.

Pruebas de CI/CD

Las **pruebas en entornos de Integración Continua (CI) y Entrega Continua (CD)** permiten garantizar que los cambios en el código se integren y desplieguen sin introducir errores.

Características clave

- Permiten detectar fallos automáticamente en cada cambio del código.
- Se integran con herramientas de desarrollo como GitHub Actions, Jenkins o GitLab CI/CD.
- Se ejecutan en servidores automatizados.
- Mejoran la estabilidad del software y reducen riesgos en la entrega.

Flujo de trabajo

1. Un desarrollador sube cambios al repositorio de código (Git).
2. Se ejecutan pruebas de unidad y de integración automáticamente (CI).
3. Si las pruebas pasan, el software se despliega en un entorno de prueba o producción (CD).

Pruebas de aceptación

Las **pruebas de aceptación** verifican que el software cumple con los requisitos funcionales establecidos por el cliente o los usuarios finales. Son una parte crucial antes del lanzamiento del producto.

Características clave

- Se centran en la experiencia del usuario y en la funcionalidad esperada.
- Pueden ser realizadas manualmente o automatizadas.
- Incluyen escenarios de uso reales.
- Generalmente son ejecutadas por un equipo de QA o por el cliente.

Tipos de pruebas de aceptación

- **Pruebas Alfa**: Se realizan en un entorno de desarrollo, con usuarios internos o testers.
- **Pruebas Beta**: Se realizan con usuarios reales en un entorno de producción limitado.

Herramientas como **Cucumber** permiten escribir pruebas en lenguaje natural usando la metodología **BDD (Behavior-Driven Development)**.

Cucumber emplea una sintaxis especial llamada Gherkin. Gherkin está traducido a 70 idiomas o sabores. Algunos son serios, otros son más en código de broma. Aquí tienes cómo traducir las palabras clave a diferentes lenguajes.

> **Actividad:** Lee el siguiente ejemplo en Gherkin para cucumber. Sus palabras clave están en inglés. Tradúcelas al castellano siguiendo la sintaxis facilitada en la web. Tradúcelas también a catalán, a inglés en jerga pirata y a inglés en jerga de texas profundo. Parece de broma, y en parte lo es, pero todas estas traducciones son posibles.

```
Feature: Login de usuario
  Scenario: Usuario ingresa credenciales válidas
    Given el usuario está en la página de login
    When ingresa su usuario "juan" y contraseña "1234"
    Then debe ver la página de inicio
```

Este tipo de pruebas facilita la validación con clientes y usuarios no técnicos.

Pruebas de validación

Las **pruebas de validación** aseguran que el software cumple con todas las especificaciones y normativas antes de su despliegue. Son un paso esencial para garantizar la calidad del producto final.

Características clave

- Verifican que el software cumpla con los requisitos técnicos y legales.
- Aseguran la estabilidad y seguridad del sistema.
- Se realizan antes del lanzamiento oficial del software.

Tipos de pruebas de validación

- **Pruebas de seguridad**: Evalúan vulnerabilidades y riesgos.
- **Pruebas de rendimiento**: Analizan la eficiencia del sistema bajo carga.
- **Pruebas de compatibilidad**: Verifican el funcionamiento en diferentes dispositivos y entornos.
- **Pruebas de usabilidad**: Evalúan la experiencia del usuario.

Para pruebas de seguridad, por ejemplo, se pueden usar herramientas como **OWASP ZAP** (Zed Attack Proxy) , que permite analizar vulnerabilidades en aplicaciones web.

Aquí tienes el **manual para empezar a trabajar con ZAP**.

> **Actividad:** Busca herramientas para pruebas de rendimiento, de compatibilidad y usabilidad. Agrega el enlace a la página web principal de la herramienta, al getting-started si lo tiene y a algún videotutorial (en Youtube u otras plataformas) que te parezca interesante, para tener guardado el recurso para cuando te sea de utilidad.

Pruebas exploratorias

Hasta ahora, las pruebas que hemos visto siguen un plan estructurado. Sin embargo, en muchas ocasiones los testers utilizan un enfoque más libre e intuitivo llamado **pruebas exploratorias**.

Las pruebas exploratorias son un tipo de prueba donde el tester **investiga** la aplicación sin seguir un conjunto de pasos predefinidos, buscando errores de manera creativa. Se basan en la experiencia y la intuición del tester. Sus principales características son:

- No requieren casos de prueba predefinidos.
- Permiten descubrir errores inesperados.
- Son útiles cuando no hay suficiente documentación.
- Son complementarias a las pruebas automatizadas.

Supongamos que estamos probando una aplicación de banca en línea. Un tester exploratorio podría intentar:

- Ingresar caracteres extraños en el campo de usuario (`!@#$%^&*`).
- Intentar iniciar sesión con un usuario incorrecto varias veces seguidas para ver si bloquea la cuenta.
- Modificar manualmente la URL después de iniciar sesión (`www.banco.com/admin`).

En el ámbito académico, los profesores de programación son expertos *destruyendo* los programas de los alumnos a base de pruebas exploratorias. Son una forma excelente de simular el comportamiento caótico de miles de clientes interactuando con tu aplicación.

> **Actividad:**
> Realiza una sesión de pruebas exploratorias sobre una aplicación web de tu elección. Registra los errores que encuentres y clasifícalos según su gravedad.

Estándares de pruebas de software

Existen dos estándares importantes relacionados con las pruebas de software, el **ISO 25010** y el **ISTQB**.

ISO 25010 - Modelo de Calidad del Software

La norma **ISO/IEC 25010** define un modelo de calidad que evalúa el software en base a **8 características principales**:

1. **Adecuación funcional**: ¿El software cumple con sus funciones correctamente?
2. **Eficiencia de rendimiento**: ¿Responde rápido y usa bien los recursos?
3. **Compatibilidad**: ¿Funciona en diferentes entornos y sistemas?
4. **Usabilidad**: ¿Es fácil de usar para los usuarios finales?
5. **Fiabilidad**: ¿Es estable y maneja bien los errores?
6. **Seguridad**: ¿Protege los datos y evita accesos no autorizados?
7. **Mantenibilidad**: ¿Es fácil de modificar y mejorar?
8. **Portabilidad**: ¿Puede ejecutarse en distintos entornos sin problemas?

Las pruebas de software ayudan a garantizar estas características, dependiendo del tipo de prueba que se aplique.

> **Actividad:**
> Relaciona cada una de las 8 características de ISO 25010 con los tipos de pruebas de software vistos en clase.

ISTQB - Certificación en Testing de Software

El **ISTQB (International Software Testing Qualifications Board)** es un organismo internacional que define estándares y promueve buenas prácticas en el ámbito de las pruebas de software. Su principal objetivo es certificar a profesionales en el área del testing, ofreciendo una estructura de certificación que cubre distintos niveles de conocimiento y habilidad. A través de esta certificación, ISTQB busca mejorar la calidad del software y la profesionalización de los testers a nivel global.

Niveles de Pruebas

Dentro del marco del ISTQB, se definen **diferentes niveles de pruebas** para garantizar la calidad del software en cada fase del ciclo de vida del desarrollo, que son las que ya conocemos:

- **Pruebas de Unidad**
- **Pruebas de Integración**
- **Pruebas de Sistema**
- **Pruebas de Aceptación**

Tipos de Testing

En el ISTQB, se reconocen varios **tipos de testing**, cada uno con un enfoque particular para evaluar distintas características del software:

Tipo de test	Descripción
Testing Funcional	Se centra en evaluar la funcionalidad del software y comprobar que el sistema haga lo que se espera según los requisitos definidos. Este tipo de pruebas está orientado a verificar la correcta ejecución de las funciones del sistema
Testing No Funcional	Examina características que no están directamente relacionadas con la funcionalidad, como el rendimiento, la seguridad, la accesibilidad o la usabilidad del sistema. Aquí se incluyen las pruebas de carga, de estrés y de seguridad.
Testing Estructural	Se basa en la estructura interna del software, como el código fuente, para asegurar que los componentes estén correctamente implementados. A menudo, se utiliza la técnica de *white-box testing* (pruebas de caja blanca), en la que se tiene acceso al código y la estructura interna.
Testing de Regresión	Se realizan para asegurar que las nuevas modificaciones o mejoras del software no hayan introducido errores en las funcionalidades previamente implementadas. Este tipo de pruebas son esenciales en cada ciclo de desarrollo y mantenimiento.

Principios del Testing

El ISTQB también establece una serie de **principios fundamentales del testing** que guían las mejores prácticas en la ejecución de pruebas de software:

- **No es posible probar todo**: Dado que no se puede probar exhaustivamente todo el sistema, es importante definir un enfoque de testing basado en riesgos y prioridades, seleccionando los escenarios más relevantes.
- **Las pruebas tempranas ahorran costos**: Detectar errores en etapas tempranas del desarrollo reduce significativamente los costos, ya que se evitan fallos mayores que podrían surgir más adelante en el ciclo de vida del software.
- **Las pruebas exhaustivas son imposibles**: En la práctica, nunca es posible probar todos los posibles escenarios. Por lo tanto, el testing debe enfocarse en las áreas más críticas y de mayor impacto.
- **El testing demuestra la presencia de defectos, no la ausencia de ellos**: Las pruebas de software no garantizan que el sistema esté libre de defectos, sino que buscan identificar posibles errores en el software.

> **Actividad**
> Contesta a las siguientes preguntas tipo test:

Pregunta 1: ¿Cuál de estas pruebas verifica la funcionalidad sin conocer el código interno?
A) Caja blanca
B) Caja negra
C) Prueba de integración
D) Prueba de carga

Pregunta 2: ¿Cuál de estas herramientas se usa para pruebas unitarias en Java?
A) JUnit
B) Selenium
C) Jenkins
D) Postman

Pregunta 3: ¿Qué tipo de prueba se usa en entornos de CI/CD?
A) Pruebas manuales
B) Pruebas exploratorias
C) Pruebas automatizadas
D) Pruebas alfa

Documentación de Pruebas

La **documentación de pruebas** es el registro organizado y sistemático de todo el proceso de prueba de software. No es solo un trámite, sino una parte fundamental para garantizar la **calidad**, la **trazabilidad** y el **mantenimiento** de cualquier aplicación. Permite que cualquier miembro del equipo, o incluso futuros equipos, comprenda qué se probó, cómo se probó, por qué se probó y cuáles fueron los resultados.

Factores de importancia para la documentación de las pruebas

- **Trazabilidad:** Conecta los requisitos del software con los casos de prueba, asegurando que cada funcionalidad esperada sea validada.
- **Reusabilidad:** Facilita la ejecución repetida de pruebas (pruebas de regresión) a lo largo del ciclo de vida del desarrollo, ahorrando tiempo y recursos.
- **Análisis de Fallos:** Proporciona un registro detallado para depurar errores y entender por qué un defecto se manifestó.
- **Conocimiento Compartido:** Actúa como una base de conocimiento para el equipo, reduciendo la dependencia de individuos y facilitando la incorporación de nuevos miembros.

- **Mejora Continua:** Permite analizar el proceso de prueba, identificar cuellos de botella y optimizar futuras estrategias de testeo.
- **Cumplimiento Normativo:** En muchos sectores (ej. médico, financiero), la documentación de pruebas es un requisito legal o de auditoría.

Elementos clave de la documentación de pruebas

Aunque puede variar según el proyecto, la documentación de pruebas comúnmente incluye:

- **Plan de Pruebas:** Describe el alcance, los objetivos, la estrategia, el cronograma, los recursos y los roles del equipo de pruebas.
- **Casos de Prueba:** Especifican los pasos detallados para ejecutar una prueba, los datos de entrada, los resultados esperados y el criterio de éxito/falla.
- **Guiones/Scripts de Prueba:** Para pruebas automatizadas, es el código y los pasos para ejecutar la automatización.
- **Informes de Defectos (Bugs):** Documentan los errores encontrados, incluyendo pasos para reproducirlos, gravedad, prioridad y estado.
- **Informes de Resumen de Pruebas:** Presentan un resumen ejecutivo de los resultados de la prueba, métricas clave (cobertura, defectos encontrados), riesgos y recomendaciones.

> **Actividad:**
> Imagina que trabajas como tester en una empresa que ha desarrollado una aplicación para reservar hoteles. Debes diseñar la documentación de pruebas necesaria, que incluya:
>
> 1. **Pruebas unitarias** (ej.: verificar que el cálculo del precio total funcione correctamente).
> 2. **Pruebas de integración** (ej.: validar que la conexión con la pasarela de pagos funcione bien).
> 3. **Pruebas de aceptación** (ej.: asegurar que un usuario pueda buscar un hotel y completar una reserva sin problemas).
> 4. **Pruebas de seguridad** (ej.: evitar que los usuarios puedan modificar los precios desde el frontend).
>
> Debate los diferentes puntos en clase y crea un documento con los casos de prueba para esta aplicación y especifica qué herramientas usarías en cada prueba.