1. Introducción

las pruebas de software constituyen un componente fundamental en el ciclo de vida del desarrollo de software, ya que permite identificar defectos, validar requisitos y asegurar que el producto final cumpla con los estándares de calidad esperados [1]. Su implementación sistemática y estructurada contribuye significativamente a la confiabilidad, seguridad y eficiencia del software entregado la correcta clasificación de estas pruebas facilita su planificación, ejecucion y evaluación dentro del proceso de desarrollo [1].

Este informe tiene como finalidad presentar los conceptos fundamentales relacionados con la clasificación de las pruebas de software, ilustrando cada tipo mediante ejemplos específicos, herramientas utilizadas comúnmente y su aplicación en un caso real.

2. Desarrollo

2.1 Clasificación según el Nivel

Esta clasificación se basa en la etapa del ciclo de vida del software en la que se ejecutan las pruebas, comenzando desde el nivel más bajo (unidad) hasta el nivel más alto (aceptación del usuario).

• Pruebas Unitarias

las pruebas unitarias tienen como propósito verificar de manera aislada el correcto funcionamiento de unidades individuales de código, como funciones o métodos. Estas pruebas son generalmente automatizadas y se ejecutan por los desarrolladores en su fase inicial [2]

Ejemplo: En una aplicación bancaria, se prueba que una función de conversión de divisas calcule correctamente el valor según la tasa actual.

- Pruebas de integración: Se encargan de evaluar la interacción entre múltiples módulos o componentes del sistema. Su objetivo es detectar errores en la comunicación o el intercambio de datos entre unidades previamente verificadas de forma individual [5].

Ejemplo: En un sistema de reservas de vuelos, se prueba la conexión entre el módulo de selección de vuelo y el de pago para garantizar una transacción fluida.

- Pruebas de Sistema:

Las pruebas de sistema validan el comportamiento global del software como una unidad completa, asegurando que todos los componentes funcionen correctamente bajo condiciones reales. Estas pruebas comprueban tanto los requisitos funcionales como los no funcionales del sistema [3].

Ejemplo: Evaluar un sistema de comercio electrónico completo, desde la búsqueda de productos hasta el pago y envío.

- Pruebas de aceptación

Las pruebas de aceptación son realizadas por el cliente o usuario final, y tienen como objetivo determinar si el software cumple con los requisitos establecidos y está listo para su liberación. Este tipo de pruebas tiene un enfoque en las expectativas del negocio y en la experiencia de usuario [4]

Ejemplo: Un cliente prueba una app de delivery verificando que puede registrar pedidos, pagar y rastrear el envío satisfactoriamente.

2.2 Clasificación Según el conocimiento del código

Esta clasificación depende del nivel de acceso al código fuente por parte del evaluador, lo que define el enfoque de la prueba.

• Pruebas de caja blanca: Permiten al evaluador acceder al código fuente y diseñar los casos de prueba a partir de las estructuras internas, como condicionales, bucles y rutas de ejecución [5]

Ejemplo: Verificar que todas las ramas de una sentencia if-else hayan sido ejecutadas almenos una vez en una función de validación de datos.

• Pruebas de caja Negra

Se enfoca en los requerimrentos funcionales del software, sin acceso al código fuente. Se enfoca en la entrada y salida, evaluando si el comportamiento del sistema cumple con las expectativas [3].

Ejemplo: Probar una calculadora en línea ingresando datos y comparando el resultado obtenido con el esperado, sin saber cómo se implementaron las operaciones internamente.

• Pruebas de caja gris

Combinan aspectos de la caja blanca y negra. El evaluador posee conocimiento paracial del código fuente, lo cual permite diseñar pruebas más específicas aquellas basadas solo en requerimientos externos [3],[4].

Ejemplo: Probar una API REST teniendo acceso a la estructura interna de las respuestos JSON, pero sin modificar la lógica del servidor.

2.3. Clasificación según el tipo de prueba

• Pruebas funcionales

Evalúan si las funciones de software cumplen con los requisitos especificados. Se concentran en el "que" hace el sistema, sin analizar cómo lo hace internamente [2].

Ejemplo: Verificar que un botón de "registrasse" redirija al usuario a la página de confirmación tras ingresar sus datos.

• Pruebas no funcionales

Estas pruebas miden atributos de calidad como rendimiento, escalabilidad, usabilidad seguridad o compatibilidad. No se enfocan en funcionalidades específicas, sino en cómo se comporta el sistema en su conjunto [4]

• Ejemplo: Medir cuánto tarda una aplicación móvil en cargar una pantalla al iniciar sesión desde diferentes dispositivos.

• Pruebas de regresión

Se aplican después de realizar cambios en el código para garantizar funcionalidades previamente implementadas no hayan sido afectadas negativamente [5].

Ejemplo: Tras corregir un error en el sistema de envíos de correos, se vuelve a probar el formulario de contacto para verificar que sigue funcionando correctamente.

• Pruebas de carga o rendimiento: Estas pruebas miden cómo se comporta el sistema bajo distintos niveles de carga, identificando cuellos de botella, límites de capacidad o degradación de servicio [3].

Ejemplo: Probar el rendimiento de una plataforma de videollamadas con más de 500 usuarios conectados simultáneamente.

2.4 Aplicación práctica y herramientas

Para ilustrar cómo se aplican los distintos tipos de pruebas en un entorno real, se concidera el siguiente ejemplo el desarrollo de una aplicación móvil para gestión de tareas, que permite, crear, editar y eliminar recordatorios, así como recibir notificaciones

1. Prueba Unitaria: Se prueba de forma aislada la función que valida el formato de fecha ingresada por el usuario. Esta validación se realiza antes de guardar el recordatorio en la base de datos. Permite identificar errores lógicos en funciones específicas.

2. Prueba Funcional (Caja Negra): Se simula el uso normal de la aplicación para verificar que al pulsar el botón "Agregar tarea", esta se guarde correctamente y aparezca en la lista. No se considera cómo se implementó internamente la operación, sino si cumple lo requerido.

3. Prueba de rendimiento: Se evalúa el comportamiento de la app al recibir una gran cantidad de tareas programadas al mismo día, verificando si las notificaciones se disparan correctamente y sin retrasos

2.5 Herramientas Utilizadas según el tipo de prueba

- Pruebas Unitarias: JUnit (Java), NUnit (.Net), Pytest (Python), Mocha (JS)
- Pruebas de Integración: TestNG (Java), Postman (Api), SoapUI (servicios web)
- Pruebas de Sistema: Selenium, Cypress, Robot Framework, Test Complete
- Pruebas de aceptación: Cucumber (BDD), FitNesse, Behave (Python)
- Caja banca: JUnit, NUnit, Pytest, Clover
- Caja Negra: Selenium, Cypress, Test Complete, Ranorex
- Caja Gris: Selenium + herramientas de análisis de logs
- Pruebas funcionales: Selenium, Cypress, Postman, Playwright
- Pruebas no funcionales: JMeter (rendimient), OWASP ZAP (seguridad)
- Pruebas de regresión: selenium, GithubActions, Jenkis CI
- Pruebas de carga: Apache JMeter, Gatling, Locust, BlazeMeter

3. Conclusiones

Las pruebas de software representan un proceso esencial para garantizar la calidad, confiabilidad y correcto funcionamiento de los sistemas informáticos. Cada tipo de prueba, ya sea unitaria, de integración, sistema o aceptación, cumple un rol específico dentro del ciclo de vida del software.

• La distinción entre pruebas de caja blanca, negra y gris permite adoptar enfoques diversos según el grado de acceso y compresión del código favoreciendo tanto el análisis estructural cómo el comportamiento estructural.
El uso de herramientas especializadas facilita la automatización, repetibilidad y eficiencia de los procesos de prueba, aspectos especialmente relevantes en entornos ágiles y de integración continua.

4. Recomendaciones

• Aplicar pruebas desde las primeras etapas de desarrollo, esto permite detectar errores tempranamente y reduce el costo de corrección
• Automatizar las pruebas cuando sea posible, especialmente las pruebas de regresión y las pruebas unitarias, utilizando herramientas como JUnit, selenium o Cypress.
Esto mejora la eficiencia del desarrollo y garantiza estabilidad a cambios frecuentes
• Adoptar una combinacion de enfoques (caja negra, blanca y griss), ya que cada uno aporta información complementaria y fortalece la cobertura del sistema bajo prueba