Material descargable

Tipos de Pruebas

Introducción

Las pruebas de software se pueden diferenciar en dos categorías esenciales para evaluar y garantizar la calidad de un sistema: **las pruebas funcionales y no funcionales.** Esta clasificación se centra principalmente en los diferentes enfoques que estas pruebas toman para evaluar el software. Las pruebas funcionales se centran en la funcionalidad explícita, mientras que las pruebas no funcionales se centran en atributos de calidad y comportamientos más amplios del sistema.

Estas pruebas, desplegadas en todos los niveles, desde las fases iniciales hasta las pruebas de aceptación, desempeñan un papel crítico en la detección temprana de defectos y el aseguramiento de la robustez del software.

A continuación, tendrás los elementos primordiales que las diferencian .

Pruebas Funcionales

Las pruebas funcionales se llevan a cabo en todos los niveles del proceso de desarrollo de software, desde las pruebas unitarias hasta las pruebas de aceptación. Este tipo de pruebas se centra en evaluar la funcionalidad del software y asegurarse de que cumpla con los requisitos especificados.

Diseñar pruebas funcionales generalmente requiere un sólido dominio del problema, así como conocimiento del negocio y del entorno en el que se insertará el sistema. A menudo, se utilizan documentos como la documentación funcional, historias de usuario y requisitos como base para la planificación de pruebas.

Es crucial que los testers analicen críticamente estos insumos para identificar posibles contradicciones, ambigüedades, omisiones y errores de diversa

naturaleza. Por ejemplo, entender las leyes impositivas y flujos de facturación es esencial para diseñar pruebas efectivas en sistemas relacionados con la gestión financiera en diferentes países.

Algunos ejemplos de pruebas funcionales incluyen:

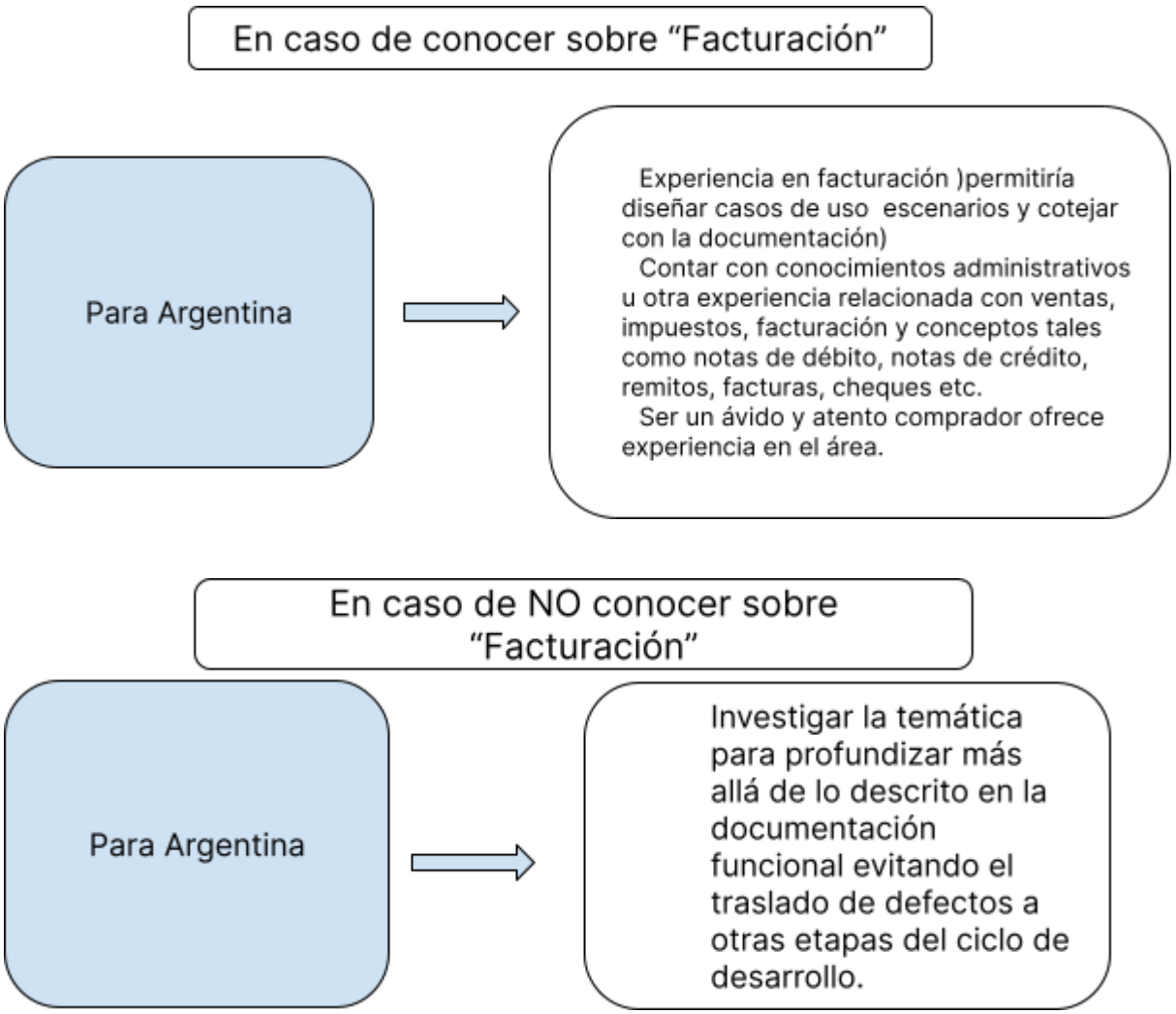
- **Pruebas unitarias**: Estas pruebas evalúan individualmente unidades o componentes aislados del software para asegurarse de que funcionen correctamente. Generalmente, son ejecutadas por los desarrolladores durante la fase de desarrollo para validar que cada unidad de código realiza su tarea prevista.
- **Pruebas de aceptación:** También conocidas como pruebas de aceptación del usuario (UAT, por sus siglas en inglés), estas pruebas se realizan para asegurarse de que el software cumple con los requisitos del usuario final. Por lo general, son llevadas a cabo por usuarios reales para validar que el software satisface sus necesidades y expectativas.
- **Pruebas de integración:** Verifican la interacción entre módulos o componentes del software.
- **Pruebas de regresión:** Aseguran que las nuevas modificaciones no afecten las funcionalidades existentes.

¿Necesitas un ejemplo?

Diseñar casos de prueba para un sistema de facturación de uso en **Argentina**

Diseñar casos de prueba para un sistema de facturación de uso en **Panamá**

Entonces: ¿Cómo podría llevar a cabo esta tarea de análisis si no conozco la ley impositiva de Argentina y la ley impositiva de Panamá? ¿Y si desconozco los flujos de facturación de cada país?



Pruebas No Funcionales

Las pruebas no funcionales se centran en aspectos no funcionales del comportamiento del sistema, como rendimiento, accesibilidad, usabilidad y seguridad. Al igual que las pruebas funcionales, se llevan a cabo en todos los niveles del desarrollo.

A diferencia de las pruebas funcionales, las pruebas no funcionales requieren un nivel de conocimiento técnico. Evaluar el rendimiento, garantizar la seguridad y optimizar la usabilidad del sistema son ejemplos de áreas en las que el conocimiento técnico es crucial para diseñar y ejecutar pruebas efectivas.

Algunos ejemplos de pruebas no funcionales incluyen:

- **Pruebas de carga:** Estas pruebas evalúan la capacidad del software para manejar una carga específica o un número de usuarios concurrentes. Se centran en verificar el rendimiento y la estabilidad del sistema bajo condiciones de carga extrema.
- **Pruebas de seguridad:** Evalúan la resistencia del sistema frente a posibles amenazas y vulnerabilidades.
- **Pruebas de usabilidad:** Se centran en la experiencia del usuario y la facilidad de uso del software.
- **Pruebas de rendimiento:** Evalúan la velocidad, capacidad y eficiencia del sistema bajo diversas condiciones.
- **Pruebas de compatibilidad:** Garantizan que el software funcione correctamente en diferentes entornos y dispositivos.
- **Pruebas de estrés**: Determinan cómo se comporta el sistema bajo cargas extremas o condiciones límite.

¿Necesitas un ejemplo?

- El conocimiento técnico sobre posibles vulnerabilidades que podrían afectar a un sistema
- El conocimiento técnico para evaluar y medir la performance del sistema, su accesibilidad o usabilidad.

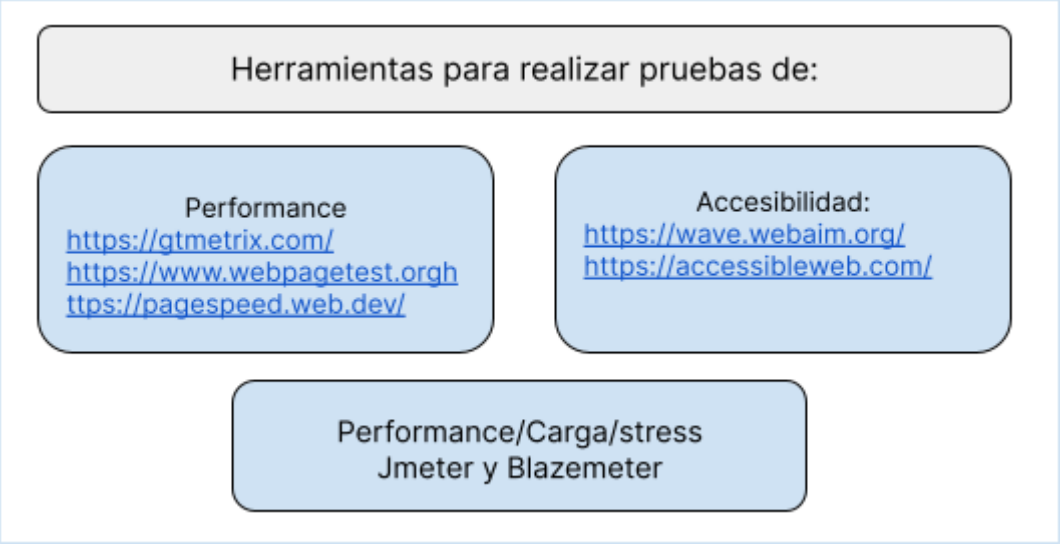
En general, los resultados de las pruebas no funcionales suelen ser medibles y cuantificables.

Estas pruebas son las que verifican los requerimientos NO funcionales:
Repasemos algunos de ellos:

- Instalabilidad
- Mantenibilidad (y la capacidad del sistema de recibir cambios)
- Performance, tiempos de respuesta
- Manejo de carga (respuesta del sistema cuando la carga aumenta)
- Manejo de estrés (comportamiento cuando se alcanzan los límites más altos de capacidad)

- Portabilidad (comportamiento en diferentes sistemas operativos)
- Recuperabilidad (procedimiento de recuperación ante fallos)
- Fiabilidad (habilidad de realizar las funciones esperadas a lo largo del tiempo)
- Usabilidad (facilidad de uso e interacción)

Existen distintas herramientas para realizar las pruebas no funcionales. Algunas de estas son:



Pruebas de caja blanca (WBT – White-Box Testing / structural testing)

El testing de "caja blanca", también conocido como pruebas estructurales, refiere al hecho de que testeamos sabiendo cómo está construido el código: conocemos la estructura del código.
Este tipo de prueba funcional se enfoca en la estructura interna del sistema, su código, arquitectura y los flujos de datos del sistema.
Las pruebas de caja blanca se llevan a cabo en el nivel de las pruebas unitarias y de integración, niveles en los cuales se desarrollan las pruebas.

Esto significa que su creación requiere conocimientos técnicos tales como capacidad de interpretación y redacción de código, conocimiento sobre bases de datos y herramientas de desarrollo.

Pruebas relacionadas a cambios

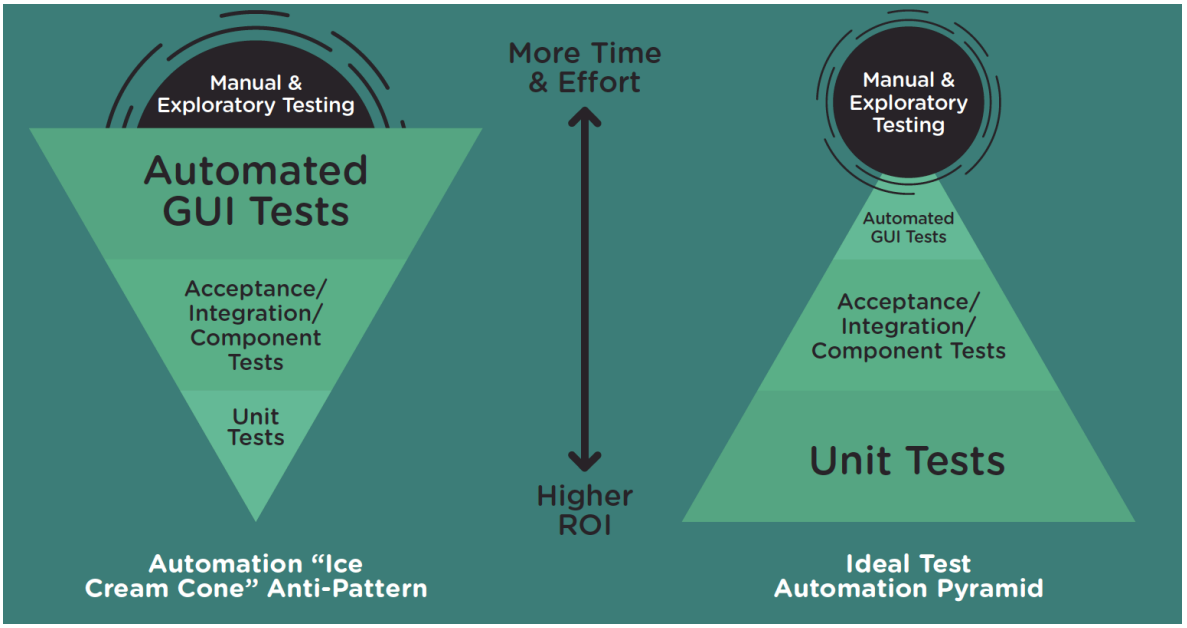
Los defectos pueden manifestarse en cualquier fase del proceso de prueba, y su corrección es esencial para mejorar la calidad del sistema. Una vez que se ha corregido un defecto, resulta crucial someter la modificación aplicada al software a pruebas exhaustivas para confirmar su funcionamiento conforme a las expectativas. En el caso de que surjan problemas durante estas pruebas, es necesario reportar.

Además de testear que el bug reportado se corrigió, es necesario asegurarse que esa corrección no haya roto ninguna parte del sistema.

No solo se trata de verificar que el error reportado se haya corregido; también es esencial garantizar que la corrección no haya introducido nuevas fallas en otras áreas del sistema. Para llevar a cabo esta validación, se recurre a las pruebas de regresión (regression testing). Dependiendo del tipo de sistema, es probable que las pruebas de regresión se realicen de manera automatizada para asegurar una cobertura eficiente y exhaustiva. Estas pruebas desempeñan un papel crucial en mantener la integridad y estabilidad del sistema a lo largo de su evolución.

¿Necesitas un ejemplo?

Si se trata de un sistema que se desarrolla con metodología ágil sin fecha de finalización, la decisión sobre automatizar o no (cuánto y qué) dependerá de los recursos, el tipo de proyecto, las ventajas y desventajas de invertir el tiempo para crear el framework necesario para la automatización y los scripts para las pruebas automatizadas.
Cuando los tests de regresión no están automatizados hay que ejecutarlos de manera manual.



Fuente: https://medium.com/@wc.testing.qa/la-famosa-pir%C3%A1mide-de-cohn-y-la-dura-realidad-e1250dfbe5f3

Enfoque estático y dinámico

Los enfoques estático y dinámico son dos perspectivas complementarias utilizadas en el proceso de prueba de software para evaluar distintos aspectos de un sistema.

Para introducir el tema, veamos el siguiente **video** explicativo

1. **Pruebas estáticas (static testing)**

Las técnicas de pruebas estáticas testean el sistema o software **sin ejecutarlo**. Su objetivo es encontrar errores y defectos antes de que se construya y ejecute el código debido a que la detección temprana de defectos permite que su corrección sea menos costosa.

Las técnicas empleadas se basan principalmente en revisión y análisis. Algunas de las actividades utilizadas para llevar a cabo estas tareas incluyen la revisión de código, inspecciones, análisis estático del código fuente, así como la revisión de documentos y especificaciones.

Las **técnicas de revisión** se enfocan en identificar errores en la documentación antes de utilizarlas como insumo para el desarrollo. De esta manera se reducen las posibilidades de errores que se trasladen al código.

Las **técnicas de análisis** (aka análisis técnico / technical analysis) analizan el código en búsqueda de defectos estructurales o problemas en la lógica de programación que podrían causar defectos. Este análisis técnico suele ser realizado por desarrolladores (P2P review) o por personas con el conocimiento técnico suficiente (technical review team) según sea la organización del equipo de desarrollo de la empresa.

Todo lo escrito puede ser objeto de análisis técnico: los insumos disponibles para la construcción del sistema, la base para el diseño de los tests, el código, los programas de especificaciones, las guías de usuario, contratos, modelos de diseño, diagramas, entre otros.

Se revisa para no encontrar contradicciones, incorrecciones, ambigüedades u omisiones.

Es importante tomar nota de todas las observaciones y preguntas que surjan de esta etapa de revisión y canalizarlas con la persona responsable.

Desde el punto de vista del análisis técnico se busca evitar defectos de diseño (como el alto acoplamiento o la baja cohesión), defectos de programación (código duplicado, variables mal declaradas o no declaradas, o no usadas, métodos muy largos o con demasiadas responsabilidades), desviaciones de las políticas o estándares de coding (coding policies o coding standards) y todo tipo de incorrecciones en la construcción del código.

2. Pruebas dinámicas (Dynamic testing)

El propósito fundamental de las pruebas dinámicas es similar al del testing estático: descubrir defectos de manera temprana. No obstante, a diferencia de este último, las pruebas dinámicas se llevan a cabo ejecutando el código, es decir, utilizando el sistema construido.

El enfoque dinámico implica poner en marcha el sistema y examinar los comportamientos observables durante la ejecución del código. Este tipo de prueba se realiza cuando el sistema está operativo, permitiendo así evaluar su rendimiento, fiabilidad y otros aspectos dinámicos. Algunos de los objetivos clave de las pruebas dinámicas incluyen:
:

- Evaluación del Rendimiento: Analizar el desempeño del sistema en términos de velocidad, eficiencia y escalabilidad.
- Identificación de Defectos durante la Ejecución: Descubrir y corregir posibles problemas que solo se manifiestan durante la operación del sistema.
- Verificación del Cumplimiento de Requisitos: Confirmar que el software funciona de acuerdo con los requisitos especificados.

Las pruebas dinámicas ofrecen una perspectiva activa al evaluar el comportamiento real del sistema en funcionamiento, contribuyendo así a garantizar que el software cumpla con los estándares de rendimiento y fiabilidad esperados.