TESTING

¿Qué es el Testing?

Es un proceso destructivo que trata de encontrar defectos (cuya presencia se asume) en el código. Los testers deben adoptar una perspectiva crítica y buscar activamente las fallas en el software. Deberían tratar de "romper" el sistema. El éxito de las pruebas de se mide por la cantidad de errores que se detectan. Cuantos más errores se encuentren y se corrijan, más confiable será el software final. Las pruebas de software son una actividad costosa y que consume mucho tiempo. Se estima que entre el 30% y el 50% del costo total de desarrollo de un software se destina a las pruebas.

Asegurar la calidad vs Controlar la calidad

La calidad no se puede inyectar en las últimas etapas del desarrollo. Es decir, no se puede esperar a tener un producto casi terminado y luego realizar pruebas para arreglar lo que esté mal. La calidad se construye desde el principio y en cada etapa del proceso.

Cada tarea que se realiza durante el desarrollo, desde el diseño hasta la implementación, influye en la calidad final del producto. Por lo tanto, el testing debe ser una actividad continua y no solo una fase asilada al final.

Encontrar y corregir errores al principio del desarrollo es mucho más eficiente que hacerlo al final. Cuanto más tarde se detecte un error, más costos y complejo será solucionarlo.

La calidad no solo se refiere a las características del producto final, sino también a los procesos utilizados para desarrollo. Es decir, debemos buscar formas de mejorar los procesos para prevenir futuros errores y mejorar la eficiencia.

Aunque el testing es una herramienta fundamental para asegurar la calidad, no puede garantizar por sí solo que el software sea perfecto. Siempre habrá algún riesgo residual.

Error vs Defecto

Un error es algo que está mal hecho en el momento de producción, una acción humana que produce un resultado incorrecto. Es la causa raíz de un problema.

Un defecto es un error encontrado en etapas posteriores a la de producción. Es la manifestación de ese error en el software. Es la consecuencia tangible del error, la parte del código que no funciona como debería debido a ese error.

Los defectos son más costosos y graves que los errores. El peor defecto vendría siendo el que se origina en la etapa de requerimientos. A medida que pasa el tiempo, más costoso se vuelve.

Severidad y Prioridad

La severidad de un defecto se refiere al impacto que tiene este en el funcionamiento del software. Es una medida de cuán grave es el problema. Se clasifican de la siguiente manera:

1. Bloqueante: impide completamente el uso del software. Es un error crítico que debe ser corregido inmediatamente.
2. Crítico: afecta significativamente la funcionalidad del software y puede causar la falla de un proceso importante.
3. Mayor: causa un problema notable en el funcionamiento del software, en caminos alternativos frecuentados.
4. Menor: causa un problema que no afecta significativamente la experiencia de usuario, en caminos poco usados.
5. Cosmético: no afecta a la funcionalidad del software, pero que puede ser considerar como un error estético.

La prioridad de un defecto, por otro lado, se refiere a la urgencia con la que debe sr corregido. Es decir, determina el orden en el que los defectos serán arreglados. Se clasifican de la siguiente manera:

1. Urgencia: debe ser corregido inmediatamente.
2. Alta: debe ser corregido lo antes posible.
3. Media: debe ser corregido en un plazo razonable.
4. Baja: puede ser corregido en una futura versión o iteración.

Niveles de Testing

- Testing unitario: se prueba cada componente individualmente tras su construcción. Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración. Los errores se suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.
- Testing de integración: test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto. Cualquier estrategia de prueba de versión o de integración debe ser incremental, para lo que existen dos esquemas principales:
    - Integración de arriba hacia abajo
    - Integración de abajo hacia arriba

  Lo ideal es tener una combinación de ambos esquemas. Tener en cuenta que los módulos más críticos deben ser probado lo más tempranamente posible. Los puntos de integración son simples:
    - Conectar de a poco las partes más complejas.
    - Minimizar la necesidad de programas auxiliares.
- Testing de sistema: es la prueba realizada cuando una aplicación está funcionando como un todo. Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.) El entorno de prueba debe corresponder al entorno de producción tanto como sea posible para reducir al mínimo el riesgo de incidentes debidos al ambiente específicamente y que no se encontraron en las pruebas. Deben investigar tanto requerimientos funcionales y no funcionales del sistema.
- Testing de aceptación: es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades. La meta es el establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema. Encontrar defectos no es el foco principal en las pruebas de aceptación. Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

Ambientes para construcción del software

- Ambiente de desarrollo: es donde el equipo escribe, modifica y depura el código. Características:
    - Flexibilidad: se permite realizar cambios constantes al código.
    - Personalización: cada desarrollador puede configurar su entorno de trabajo según sus preferencias.
    - Herramientas: editores de código, debuggers, sistema de control de versiones.
- Ambiente de prueba: aquí se realizan las pruebas del software para identificar y corregir errores antes de que llegue a producción. Características:
    - Similitud al ambiente de producción: debe ser lo más parecido posible al ambiente real para garantizar que los resultados de las pruebas sean confiables.
    - Herramientas: frameworks de prueba, herramientas de automatización.
- Ambiente preproducción: es un entorno intermedio entre el desarrollo y la producción, donde se realizan pruebas finales y se preparan las últimas configuraciones antes del lanzamiento. Características:
    - Configuración casi idéntica a producción: debe ser lo más similar posible al ambiente de producción para evitar sorpresas.
    - Pruebas exhaustivas: se realizan pruebas finales para asegurar la calidad del software.
- Ambiente de producción: es el entorno donde el software está disponible para los usuarios finales. Características:
    - Estabilidad: debe ser altamente confiable y seguro.
    - Monitoreo constante: se monitorea el rendimiento y la disponibilidad del software.

Casos de Prueba

Un caso de prueba es la unidad de la actividad de la prueba. Consta de tres partes:

- Objetivo: la característica del sistema a comprobar.
- Datos de entrada y de ambiente: datos a introducir al sistema que se encuentra en condiciones preestablecidas.

- Comportamiento esperado: la salida o la acción esperada en el sistema de acuerdo los requerimientos del mismo.

El objetivo es encontrar errores de forma completa y con el mínimo de esfuerzo y tiempo.

Se pueden derivar casos de prueba:

 Desde documentos del cliente:

- **¿Qué implica?** Los casos de prueba se generan directamente a partir de los documentos que el cliente proporciona, como los requisitos funcionales, las especificaciones de usuario o los casos de uso.
- **Por qué es importante:** Garantiza que el software cumple con las expectativas y necesidades del cliente desde el principio.
- **Ejemplo:** Si el cliente especifica que un sistema debe permitir el registro de nuevos usuarios, un caso de prueba podría ser "Verificar que un nuevo usuario se pueda registrar con un correo electrónico válido y una contraseña segura".

 Desde información de relevamiento:

- **¿Qué implica?** Se derivan casos de prueba a partir de la información recopilada durante el análisis de los requisitos, como entrevistas con los usuarios, talleres y documentación existente.
- **Por qué es importante:** Permite identificar posibles escenarios de uso y casos límite que no estén explícitamente documentados.
- **Ejemplo:** Si durante una entrevista con un usuario se descubre que a menudo comete errores al ingresar su dirección, se puede crear un caso de prueba para verificar que el sistema maneje correctamente los errores de formato en la dirección.

 Desde requisitos:

- **¿Qué implica?** Los casos de prueba se diseñan basados en los requisitos funcionales y no funcionales del software.
- **Por qué es importante:** Asegura que todas las funcionalidades del sistema estén cubiertas por las pruebas.
- **Ejemplo:** Si un requisito establece que el sistema debe ser capaz de generar reportes en formato PDF, un caso de prueba podría ser "Verificar que se pueda generar un reporte en formato PDF con los datos correctos".

 Desde especificaciones de programación:

- **¿Qué implica?** Los casos de prueba se derivan del diseño técnico del software, como diagramas de flujo, diagramas de clases y código fuente.
- **Por qué es importante:** Permite verificar la correcta implementación de las funcionalidades del software a nivel técnico.
- **Ejemplo:** Si una función está diseñada para calcular el área de un círculo, un caso de prueba podría verificar que la función devuelva el valor correcto para diferentes radios.

 Desde el código:

- **¿Qué implica?** Los casos de prueba se generan directamente a partir del código fuente, utilizando técnicas como la cobertura de código.
- **Por qué es importante:** Permite identificar áreas del código que no están cubiertas por las pruebas y mejorar la calidad del código.
- **Ejemplo:** Si una sección de código no está cubierta por ninguna prueba, se puede crear un caso de prueba específico para esa sección.

Estrategias de Testing

Característica	Caja Negra	Caja Blanca
Visión	Externa	Interna
Enfoque	Funcionalidad	Estructura de código
Diseño de casos de prueba	Basado en requisitos	Basado en el código
Técnicas	Particiones de equivalencia Análisis de valores límite Tablas de decisión	Cobertura de código Análisis de flujo de control Pruebas de bifurcaciones
Ventajas	Independiente de la implementación Fácil de realizar	Identificar errores a nivel de código Mejora la calidad del código
Desventajas	No garantiza cobertura completa Difícil detectar errores internos	Requiere conocimientos técnicos Costoso

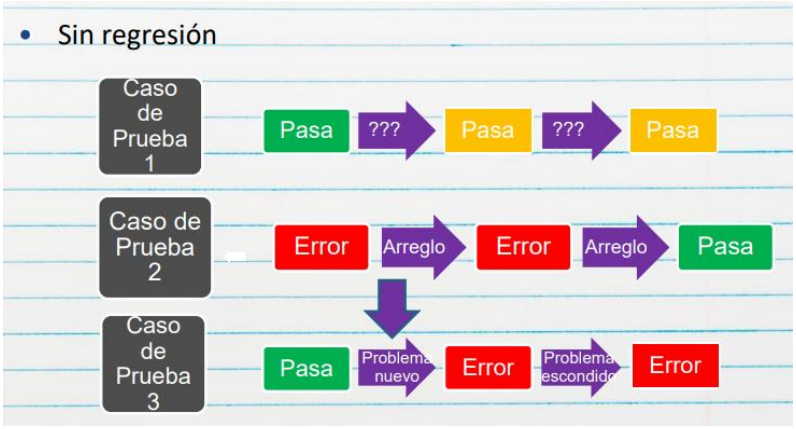
Ambas técnicas son complementarias y se utilizan en conjunto para garantizar la calidad del software.

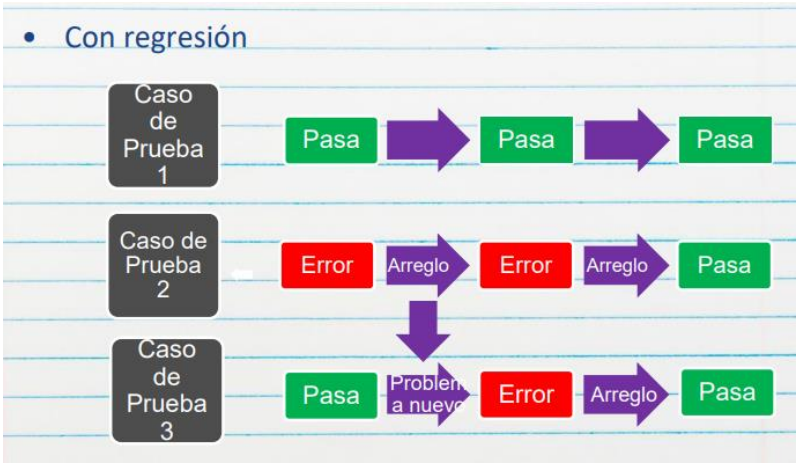
Ciclo de Test

Es una forma de definir hasta dónde hacer testing. Por ejemplo: “Haremos N cantidad de ciclos de prueba”. La cantidad de pruebas que se harán depende del tipo de software que se esté realizando. Abarca la ejecución de la totalidad de los casos de prueba establecidos y aplicados a una misma versión del sistema a probar.

Regresión

Se basa en que cuando se corrige un defecto, se produce uno nuevo. Al concluir un ciclo de pruebas, y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.}





Proceso Definido del Testing

- Planificación y Control:

Es la actividad de verificar que se entienden las metas y los objetivos del cliente, las partes interesadas, el proyecto y los riesgos de las pruebas que se pretende abordar.

 Construcción del Test Plan:

- **Riesgos y Objetivos del Testing:** Se identifican los posibles riesgos que pueden afectar la calidad del software y se establecen los objetivos específicos que se quieren alcanzar con las pruebas.
- **Estrategia de Testing:** Se define el enfoque general que se seguirá para realizar las pruebas, incluyendo los tipos de pruebas que se llevarán a cabo (funcionales, no funcionales, etc.), el nivel de detalle de las pruebas y la metodología a utilizar.
- **Recursos:** Se identifican los recursos necesarios para llevar a cabo las pruebas, como personal, herramientas, equipos y entornos de prueba.
- **Criterio de Aceptación:** Se establecen los criterios que determinarán si el software está listo para ser lanzado, es decir, qué condiciones deben cumplirse para considerar que el software es de calidad suficiente.

 Control**:**

- **Revisar los resultados del testing:** Se evalúan los resultados de las pruebas para identificar defectos y verificar si se cumplen los criterios de aceptación.
- **Test coverage y criterio de aceptación:** Se verifica si se ha cubierto adecuadamente el software con las pruebas y si se han cumplido los criterios de aceptación establecidos.
- **Tomar decisiones:** En función de los resultados de las pruebas, se toman decisiones sobre si el software está listo para ser lanzado, si se necesitan realizar pruebas adicionales o si es necesario corregir defectos.

- Identificación y Especificación

Esta fase es fundamental para establecer las bases sólidas sobre las cuales se construirán las pruebas. Se trata de definir claramente qué se va a probar, cómo se va a probar y qué se espera obtener de las pruebas.

 Revisión de la base de pruebas:

- Se verifica que toda la documentación relacionada con las pruebas esté completa y actualizada.
- Esto incluye los planes de prueba, los casos de prueba existentes, los scripts de automatización, etc.
- Se asegura que la información sea consistente y esté alineada con los requisitos del software.

 Verificación de las especificaciones para el software bajo pruebas:

- Se revisan detalladamente los requisitos funcionales y no funcionales del software.
- Se verifica que estén claros, completos y sin ambigüedades.
- Se identifican cualquier inconsistencia o falta de información.

 Evaluar la testabilidad de los requerimientos y el sistema:

- Se analiza si los requisitos son testables, es decir, si se pueden diseñar casos de prueba para verificar su cumplimiento.
- Se evalúa la complejidad del sistema y se identifican las áreas que podrían presentar mayores desafíos para las pruebas.

 Identificar los datos necesarios:

- Se determinan los datos de entrada y salida que se utilizarán en las pruebas.
- Se consideran los datos válidos, inválidos y límite para cubrir una amplia gama de escenarios.
- Se pueden utilizar datos de prueba ficticios o reales.

 Diseño y priorización de los casos de pruebas:

- Se crean los casos de prueba detallados, especificando los pasos a seguir, los datos de entrada esperados y los resultados esperados.
- Se priorizan los casos de prueba en función de su importancia y riesgo.
- Se pueden utilizar diferentes técnicas de diseño de casos de prueba, como particiones de equivalencia, análisis de valores límite, etc.

 Diseño del entorno de prueba:

- Se define el entorno en el que se ejecutarán las pruebas, incluyendo el hardware, software, redes y datos necesarios.
- Se configura el entorno para que sea lo más similar posible al entorno de producción.

➢ Ejecución

Esta fase es donde se ponen en práctica todos los preparativos realizados en etapas anteriores. Es aquí donde se ejecuta cada caso de prueba diseñado y se comparan los resultados obtenidos con los esperados.

 Desarrollar y dar prioridad a nuestros casos de prueba:

- Se elaboran los casos de prueba detallados, especificando los pasos a seguir, los datos de entrada esperados y los resultados esperados.
- Se priorizan los casos de prueba en función de su importancia y riesgo.

 Crear los datos de prueba:

- Se preparan los datos necesarios para ejecutar los casos de prueba. Estos pueden ser datos válidos, inválidos o límite.

 Automatizar lo que sea necesario:

- Se automatizan los casos de prueba repetitivos o complejos para aumentar la eficiencia y reducir el error humano.

 Creación de conjuntos de pruebas de los casos de prueba para la ejecución de la prueba eficientemente:

- Se organizan los casos de prueba en conjuntos lógicos para facilitar la ejecución y el seguimiento.

 Implementar y verificar el ambiente:

- Se configura el entorno de prueba para que sea lo más similar posible al entorno de producción.
- Se verifica que el entorno esté listo para ejecutar las pruebas.

 Ejecutar los casos de prueba:

- Se ejecutan los casos de prueba siguiendo los pasos definidos.
- Se registran los resultados obtenidos.

 Registrar el resultado de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de pruebas:

- Se documentan los resultados de cada caso de prueba, incluyendo cualquier error o falla encontrada.
- Se registra la información relevante sobre el software y el entorno de prueba.

 Comparar los resultados reales con los resultados esperados:

- Se analizan los resultados obtenidos y se comparan con los resultados esperados definidos en los casos de prueba.
- Se identifican las discrepancias y se registran como defectos.

➢ Evaluación y Reporte:

Esta fase es fundamental para determinar si el software está listo para su lanzamiento y para aprender de las lecciones del proceso de testing.

 Evaluar los criterios de Aceptación:

- Se verifica si el software cumple con todos los criterios de aceptación establecidos en la fase de planificación.
- Se asegura que el software cumple con los requisitos funcionales y no funcionales.

 Reporte de los resultados de las pruebas para los stakeholders:

- Se elabora un informe detallado de los resultados de las pruebas, incluyendo los defectos encontrados, la cobertura de las pruebas y los resultados de las pruebas de rendimiento.
- El informe se presenta a los stakeholders (clientes, equipo de desarrollo, etc.) para informarles sobre el estado del proyecto.

 Recolección de la información de las actividades de prueba completadas para consolidar:

- Se recopila toda la información relevante sobre las pruebas realizadas, como los casos de prueba ejecutados, los defectos encontrados y las soluciones implementadas.
- Esta información se consolida en un repositorio central para su análisis posterior.

 Verificación de los entregables y que los defectos hayan sido corregidos:

- Se verifica que todos los entregables del proyecto estén completos y que los defectos identificados durante las pruebas hayan sido corregidos.
- Se asegura que el software esté listo para su lanzamiento.

 Evaluación de cómo resultaron las actividades de testing y se analizan las lecciones aprendidas:

- Se evalúa la eficacia de las pruebas realizadas y se identifican las áreas de mejora.
- Se analizan las lecciones aprendidas para mejorar los procesos de testing futuros.

Por qué el testing es necesario? Quick Quiz

- Porque la existencia de defectos en el software es inevitable ✓
- Para llenar el tiempo entre fin del desarrollo y el día del release ✗
- Para probar que no hay defectos ✗
- Porque el testing está incluido en el plan del proyecto ✗
- Porque debuggear mi código es rápido y simple ✗

Por qué el testing es necesario? Quick Quiz

- Para evitar ser demandado por mi cliente ✓
- Para reducir riesgos ✓
- Para construir la confianza en mi producto ✓
- Porque las fallas son muy costosas ✓
- Para verificar que el software se ajusta a los requerimientos y validar que las funciones se implementan correctamente. ✓

¿Cuánto testing es suficiente?

- El testing exhaustivo es imposible.
- Decidir cuánto testing es suficiente depende de:
  - Evaluación del nivel de riesgo
  - Costos asociados al proyecto
- Usamos los riesgos para de terminar:
  - Que testear primero
  - A qué dedicarle más esfuerzo de testing
  - Que no testear (por ahora)

¿Cuánto testing es suficiente?

- El **Criterio de Aceptación** es lo que comúnmente se usa para resolver el problema de determinar cuándo una determinada fase de testing ha sido completada.
- Puede ser definido en términos de:
  - Costos
  - % de tests corridos sin fallas
  - Fallas predichas aún permanecen en el software
  - No hay defectos de una determinada severidad en el software

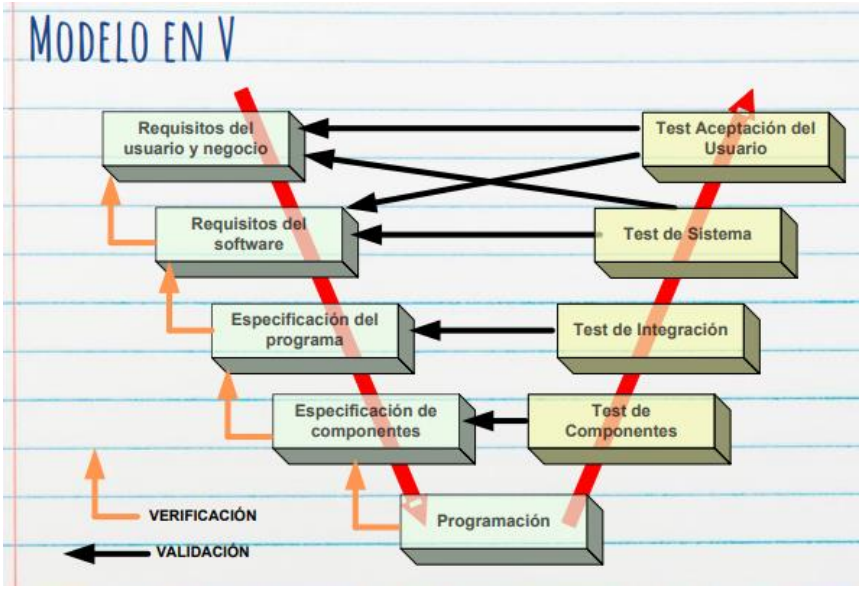
Principios del Testing

- El testing muestra presencia de defecto
- El testing exhaustivo es imposible
- Testing temprano
- Agrupamiento de Defectos
- Paradoja del Pesticida
- El testing es dependiente del contexto
- Falacia de la ausencia de errores

Principios del Testing

- Un programador debería evitar probar su propio código.
- Una unidad de programación no debería probar sus propios desarrollos.
- Examinar el software para probar que no hace lo que se supone que debería hacer es la mitad de la batalla, la otra mitad es ver que hace lo que no se supone que debería hacer.
- No planificar el esfuerzo de testing sobre la suposición de que no se van a encontrar defectos.

Modelo en V



Smoke Test

El Smoke Testing, o pruebas de humo en español, es un tipo de prueba funcional que se realiza para comprobar rápidamente si un nuevo build o versión de un software es lo suficientemente estable como para pasar a pruebas más exhaustivas.

Imagina que acabas de construir un prototipo de un avión. Antes de realizar vuelos de prueba completos, harías una inspección rápida para asegurarte de que los motores encienden, los controles responden y no hay fugas evidentes. Si todo parece funcionar bien, entonces procederías a las pruebas más detalladas. Así funciona el Smoke Testing en el desarrollo de software.

Tipos de Pruebas

Pruebas Funcionales

- ¿Qué evalúan? Verifican si el software se comporta como se espera según los requisitos funcionales definidos. Esto significa que evalúa si cada función o característica del software realiza las tareas para las que fue diseñada.
- Ejemplos:
    - Comprobar si un botón al ser presionado ejecuta la acción esperada.
    - Verificar si un formulario calcula correctamente los totales.
    - Asegurarse de que un sistema de login permita el acceso solo a usuarios autorizados.
- Enfoque: Se centra en la lógica interna del software y en las funcionalidades específicas del sistema.

Pruebas No Funcionales

- ¿Qué evalúan? Evalúan aspectos del software que no están directamente relacionados con su funcionalidad específica, pero que son cruciales para la experiencia del usuario y el rendimiento del sistema.
- Ejemplos:
    - Rendimiento: Medir la velocidad de respuesta, la capacidad de manejar cargas altas, etc.
    - Usabilidad: Evaluar la facilidad de uso de la interfaz, la intuitividad de las funciones, etc.
    - Seguridad: Verificar que el sistema esté protegido contra ataques externos y que los datos estén seguros.
    - Fiabilidad: Evaluar la capacidad del sistema de funcionar sin fallos durante un período de tiempo determinado.
    - Compatibilidad: Comprobar que el software funciona correctamente en diferentes plataformas, navegadores o dispositivos.
- Enfoque: Se centra en características como la usabilidad, el rendimiento, la seguridad y la confiabilidad del sistema en su conjunto.