Técnicas de *testing*



Tabla de CONTENIDOS

1. Técnicas y herramientas de testing..... 3
Introducción 3
Técnicas de Testing 4
Herramientas de Testing 6
Definición y Alcance de los Tipos de Testing..... 7
Diseño de Casos de Testing..... 8
Material complementario 10
Cierre e ideas fuerza..... 10

2. Guía: Alcances y definiciones de los tipos de testing en el desarrollo de software..... 11
Introducción 11
Material complementario 15
Cierre e ideas fuerza..... 15

3. Guía sobre cómo detectar la complejidad del testing 16
Introducción 16
Definición de Complejidad en Testing 17
Criterios para Detectar la Complejidad del Testing..... 17
Métodos para Clasificar la Complejidad del Testing..... 17
Clasificación de la Complejidad 18
Herramientas de Apoyo..... 18
Material complementario 19
Cierre e ideas fuerza..... 19

4. Soluciones a problemas y criterios de factibilidad con técnicas de testing..... 20
Introducción 20
Definición y Propósito de las Técnicas de Testing..... 21
Identificación de Problemas Comunes en el Desarrollo de Software 21
Soluciones Tentativas a Problemas..... 21
Criterios de Factibilidad 22
Modelos Existentes en la Industria..... 22
Ejecución de Técnicas de Testing..... 23
Cierre e ideas fuerza..... 24

Bibliografía y referencias utilizadas 25

1. Técnicas y herramientas de testing

Introducción

El testing de software es un proceso crítico en el desarrollo de aplicaciones, garantizando que los productos cumplan con los requisitos especificados y funcionen correctamente bajo diversas condiciones. Existen múltiples técnicas de testing, divididas en estáticas y dinámicas, que permiten a los equipos de desarrollo identificar y corregir defectos en diferentes etapas del ciclo de vida del software. Además, el uso de herramientas de testing especializadas se ha vuelto indispensable en la industria, facilitando la automatización de pruebas y mejorando la eficiencia y precisión de las evaluaciones. En esta unidad, se explorarán las principales técnicas de testing y las herramientas utilizadas para asegurar la calidad del software.



Técnicas de Testing

Las técnicas de testing, son métodos sistemáticos utilizados para diseñar y ejecutar pruebas con el objetivo de encontrar defectos en el software y asegurar su calidad. Analizaremos a continuación las técnicas estáticas y dinámicas, dos enfoques diferentes pero complementarios dentro del proceso de testing de software.

- ## Técnicas Estáticas

Las técnicas estáticas de testing se centran en la revisión de la documentación y el código sin ejecutar el software. Estas técnicas son esenciales para identificar defectos tempranos en el ciclo de vida del desarrollo de software.

- ## Revisión por Pares

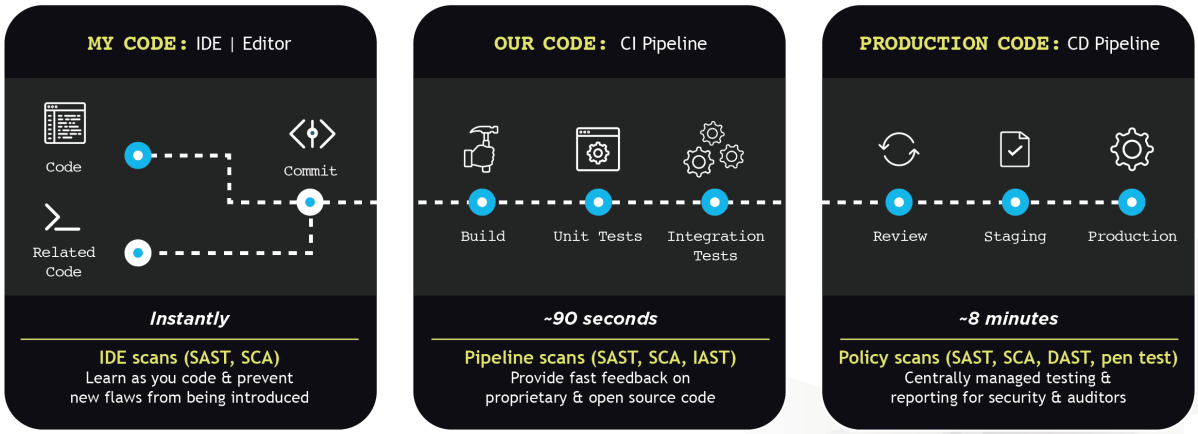
La revisión por pares implica la evaluación del código por otros desarrolladores para identificar posibles errores o mejoras. Es una técnica colaborativa que ayuda a mejorar la calidad del código desde las primeras etapas.

- ## Inspección

Las inspecciones son reuniones formales en las que se revisa el código o la documentación de manera sistemática y detallada, siguiendo un procedimiento predefinido. El objetivo es identificar defectos y garantizar el cumplimiento de estándares.

- ## Análisis Estático

El análisis estático utiliza herramientas automatizadas para examinar el código fuente sin ejecutarlo. Estas herramientas pueden detectar problemas como vulnerabilidades de seguridad, errores de sintaxis y prácticas de codificación ineficientes.



Fuente: Herrera, F. (2020, 14 de abril) *Veracode Static Code Analysis.* CPNnet Security. https://www.cpnnetsecurity.com/2020/04/14/que-es-el-analisis-de-codigo-estatico-de-veracode/

Técnicas Dinámicas

Las técnicas dinámicas de testing implican la ejecución del software para identificar defectos que ocurren durante su funcionamiento.

Testing de Caja Blanca

El testing de caja blanca, o testing estructural, se basa en el conocimiento del código fuente. Los testers diseñan casos de prueba para evaluar las estructuras internas del software, como funciones, loops y condiciones.

Testing de Caja Negra

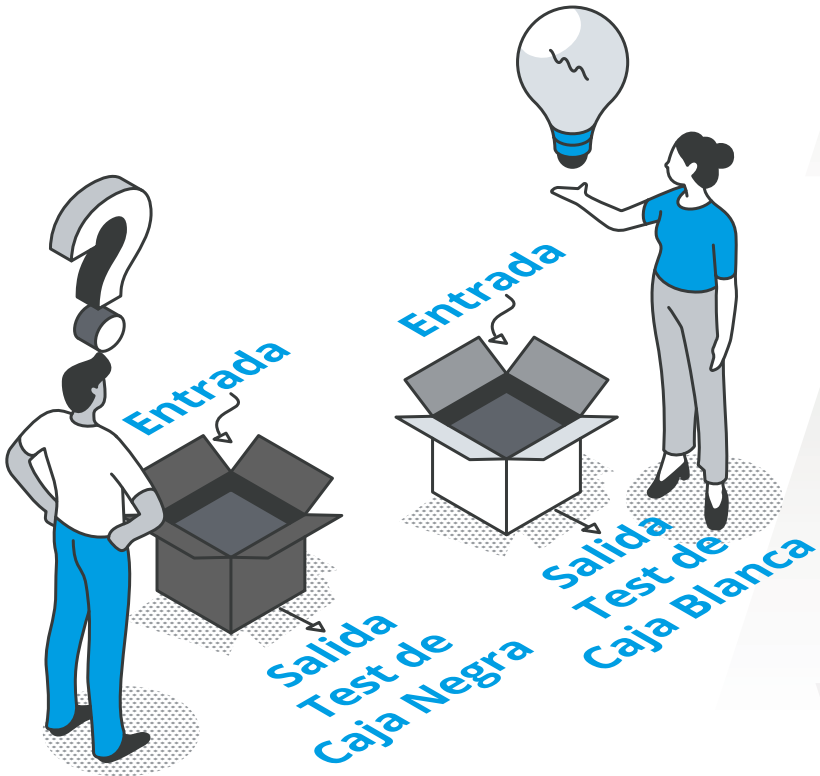
El testing de caja negra se centra en las funcionalidades del software sin conocer su estructura interna. Los testers validan si el software cumple con los requisitos especificados, basándose en las entradas y salidas.

Testing de Regresión

El testing de regresión se realiza para asegurar que las modificaciones en el código no introduzcan nuevos defectos en las partes ya probadas del software. Es crucial después de cada cambio significativo.

Testing de Rendimiento

El testing de rendimiento evalúa cómo se comporta el software bajo diferentes condiciones de carga. Se mide el tiempo de respuesta, la estabilidad y el uso de recursos.



Fuente: TelefónicaTech (2014, 29 de diciembre). https://telefonicatech.com/blog/qa-pruebas-para-asegurar-la-calidad-de

Herramientas de Testing



Las herramientas de testing, son aplicaciones o programas diseñados para ayudar a los equipos de desarrollo de software a evaluar la calidad, el rendimiento y el funcionamiento de sus aplicaciones.

Herramientas para Testing Estático

Las herramientas de testing estático ayudan a analizar el código fuente y la documentación sin ejecutar el software.

- **SonarQube**

SonarQube es una plataforma de código abierto que realiza un análisis estático del código para detectar errores, vulnerabilidades y malos olores en el código. Proporciona informes detallados y métricas de calidad.

- **Checkstyle**

Checkstyle es una herramienta de desarrollo que ayuda a los programadores a escribir código Java de acuerdo con los estándares de codificación. Detecta errores de estilo y posibles bugs.

- **Herramientas para Testing Dinámico**

Las herramientas de testing dinámico permiten la ejecución automatizada de pruebas para identificar defectos durante el funcionamiento del software.

- **Selenium**

Selenium es una suite de herramientas para la automatización de navegadores web. Permite escribir scripts en varios lenguajes de programación para realizar pruebas funcionales de aplicaciones web.

Definición y Alcance de los Tipos de Testing

- **Testing Funcional**

El testing funcional se centra en verificar que cada función del software opere conforme a los requisitos especificados. Incluye pruebas de unidad, integración, sistema y aceptación.

Pruebas Unitarias	Las pruebas unitarias validan el funcionamiento de componentes individuales del software. Utilizan frameworks como JUnit o NUnit.
Pruebas de Integración	Las pruebas de integración verifican la interacción entre componentes del software. Aseguran que los módulos integrados funcionen correctamente juntos.
Pruebas de Sistema	Las pruebas de sistema evalúan el software completo como un todo, verificando su conformidad con los requisitos del sistema.
Pruebas de Aceptación	Las pruebas de aceptación son realizadas por los usuarios finales para asegurar que el software cumple con sus necesidades y requisitos antes de la implementación.



- **Testing No Funcional**

El testing no funcional evalúa aspectos del software que no están relacionados con funciones específicas, como el rendimiento, la seguridad y la usabilidad.

Pruebas de Rendimiento	Las pruebas de rendimiento miden cómo el software responde bajo diferentes condiciones de carga. Incluyen pruebas de carga, estrés y volumen.
Pruebas de Seguridad	Las pruebas de seguridad identifican vulnerabilidades en el software que podrían ser explotadas por atacantes. Incluyen pruebas de penetración y análisis de vulnerabilidades.
Pruebas de Usabilidad	Las pruebas de usabilidad evalúan la facilidad de uso del software, asegurando que los usuarios puedan interactuar con él de manera eficiente y satisfactoria.

Diseño de Casos de Testing

El diseño de casos de testing es una fase crítica en el desarrollo de software, centrada en la planificación meticulosa de las condiciones y escenarios que serán evaluados para asegurar la calidad y fiabilidad del producto final. Para ello se emplean diversas técnicas especializadas, que explicaremos a continuación.

- **Identificación de Casos de Prueba**

La identificación de casos de prueba es el proceso de determinar las condiciones y escenarios bajo los cuales se evaluará el software. Incluye la definición de entradas, acciones y resultados esperados.

• Técnicas de Diseño de Casos de Prueba

Existen varias técnicas para diseñar casos de prueba, como la partición de equivalencia, el análisis de valores límite y las tablas de decisión.

Partición de Equivalencia	La partición de equivalencia divide el conjunto de entradas posibles en clases de equivalencia, de modo que probar una entrada de cada clase sea suficiente para evaluar el comportamiento del software.
Análisis de Valores Límite	El análisis de valores límite se centra en probar los valores en los límites de las clases de equivalencia, ya que es más probable que se encuentren defectos en estos puntos.
Tablas de Decisión	Las tablas de decisión son una técnica sistemática para representar las reglas de negocio que afectan las condiciones y acciones en casos de prueba.



Material complementario

Para complementar sobre los indicadores de rentabilidad, te sugerimos revisar el siguiente material:

Lee, Jihyun & Kang, Sungwon & Lee, D. (2012). Survey on software testing practices. Software, IET. V6. p.275-282. DOI: 10.1049/iet-sen.2011.0066. https://www.researchgate.net/publication/260649940_Survey_on_software_testing_practices

TelefónicaTech (2014, 29 de diciembre). QA pruebas para asegurar la calidad. https://telefonicatech.com/blog/qa-pruebas-para-asegurar-la-calidad-de

Cierre e ideas fuerza

Esta semana hemos analizado las técnicas de testing, diferenciando entre técnicas estáticas y técnicas dinámicas y estudiamos cuáles son las herramientas que se usan en cada técnica.

Ideas Fuerza

- El testing de software no solo detecta defectos, sino que asegura que las aplicaciones cumplen con estándares de calidad y seguridad rigurosos desde el inicio del desarrollo.
- La combinación de técnicas estáticas y dinámicas, respaldadas por herramientas avanzadas, permite identificar y corregir problemas de manera eficiente, optimizando recursos y reduciendo tiempos de desarrollo.
- Al ofrecer productos más robustos y confiables, el testing no solo mejora la satisfacción del cliente, sino que también fortalece la competitividad de las empresas en un mercado tecnológico cada vez más exigente.



2. Guía: Alcances y definiciones de los tipos de testing en el desarrollo de software



Introducción

En el desarrollo de *software*, la implementación de pruebas efectivas es crucial para garantizar la calidad y fiabilidad del producto final. Esto implica comprender y aplicar adecuadamente una variedad de tipos de testing, cada uno diseñado para evaluar diferentes aspectos del *software* en diferentes etapas del ciclo de vida del desarrollo. Desde pruebas estáticas que revisan el código sin ejecutarlo, hasta pruebas dinámicas que evalúan el comportamiento del *software* en tiempo de ejecución, cada técnica de testing juega un papel vital en la detección temprana de errores y en la optimización del rendimiento del sistema.

Además, el uso de herramientas especializadas en testing facilita la ejecución y gestión de casos de prueba, mejorando la eficiencia y la cobertura de las pruebas realizadas. Este documento explora en detalle las técnicas estáticas y dinámicas de *testing* recomendadas por la industria, así como el uso estratégico de herramientas para casos de prueba, proporcionando un marco integral para los profesionales del desarrollo de *software* en la implementación efectiva de estrategias de *testing*.

1. Pruebas Funcionales

Las pruebas funcionales son fundamentales para verificar que el software cumpla con las funciones especificadas en los requisitos del sistema. Estas pruebas pueden ser estáticas, como las revisiones de código y análisis de documentos, o dinámicas, como la ejecución de casos de prueba. En las pruebas de caja negra, se evalúa el software desde una perspectiva externa, probando entradas y salidas sin conocer la estructura interna del sistema. Por otro lado, las pruebas de caja blanca analizan la lógica interna del software mediante la inspección del código fuente y la ejecución de rutas de código específicas.

Ejemplo:

Imagina una aplicación de banca en línea que permite a los usuarios transferir dinero entre cuentas. Una prueba funcional podría verificar que, al ingresar una cantidad y seleccionar las cuentas de origen y destino, el sistema realiza la transferencia y muestra un mensaje de confirmación exitoso.

2. Pruebas No Funcionales

Las pruebas no funcionales evalúan atributos del software que no están relacionados directamente con sus funciones básicas, como la escalabilidad, el rendimiento, la seguridad y la usabilidad. Estas pruebas son cruciales para garantizar que el software pueda manejar cargas de trabajo esperadas, proteger datos sensibles y proporcionar una experiencia de usuario satisfactoria bajo diversas condiciones.

Ejemplo:

Pruebas de rendimiento en una página web. Se mide cuánto tiempo tarda la página en cargar bajo diferentes condiciones (por ejemplo, con 100, 1000, y 10,000 usuarios simultáneos).

3. Pruebas de Regresión

Las pruebas de regresión se realizan para asegurar que los cambios recientes en el software no hayan introducido errores en las funcionalidades que anteriormente funcionaban correctamente. Estas pruebas son especialmente importantes en entornos ágiles donde los desarrollos frecuentes pueden afectar inadvertidamente a otras partes del sistema.

Ejemplo:

Después de actualizar la función de búsqueda en un sitio de comercio electrónico, se ejecutan pruebas de regresión para asegurar que las funciones de búsqueda, filtrado y carrito de compras todavía funcionen correctamente.

4. Pruebas Unitarias

Las pruebas unitarias validan unidades individuales de código, como funciones o métodos, para asegurar que funcionen correctamente de manera aislada. Estas pruebas son críticas para la detección temprana de errores en el código y para facilitar el mantenimiento a largo plazo del software.

Ejemplo:

Un método que calcula el total de una factura en una aplicación de contabilidad. La prueba unitaria verifica que el método devuelva el resultado correcto para diferentes entradas de datos.

5. Pruebas de Integración

Las pruebas de integración verifican la interoperabilidad entre diferentes módulos o subsistemas del software para garantizar que funcionen juntos de manera cohesiva como un sistema completo. Estas pruebas aseguran que las interfaces entre componentes se comuniquen correctamente y que los datos se transmitan sin pérdidas ni errores.

Ejemplo:

En una aplicación de reservas de vuelos, se verifica que el módulo de pago se integre correctamente con el módulo de reservas y que se actualice el estado del vuelo después de un pago exitoso.

6. Pruebas de Aceptación

Las pruebas de aceptación son realizadas por los usuarios finales o stakeholders para validar que el software cumpla con sus necesidades y expectativas. Estas pruebas aseguran que el software sea intuitivo, fácil de usar y cumpla con los requisitos funcionales y no funcionales acordados.

Ejemplo:

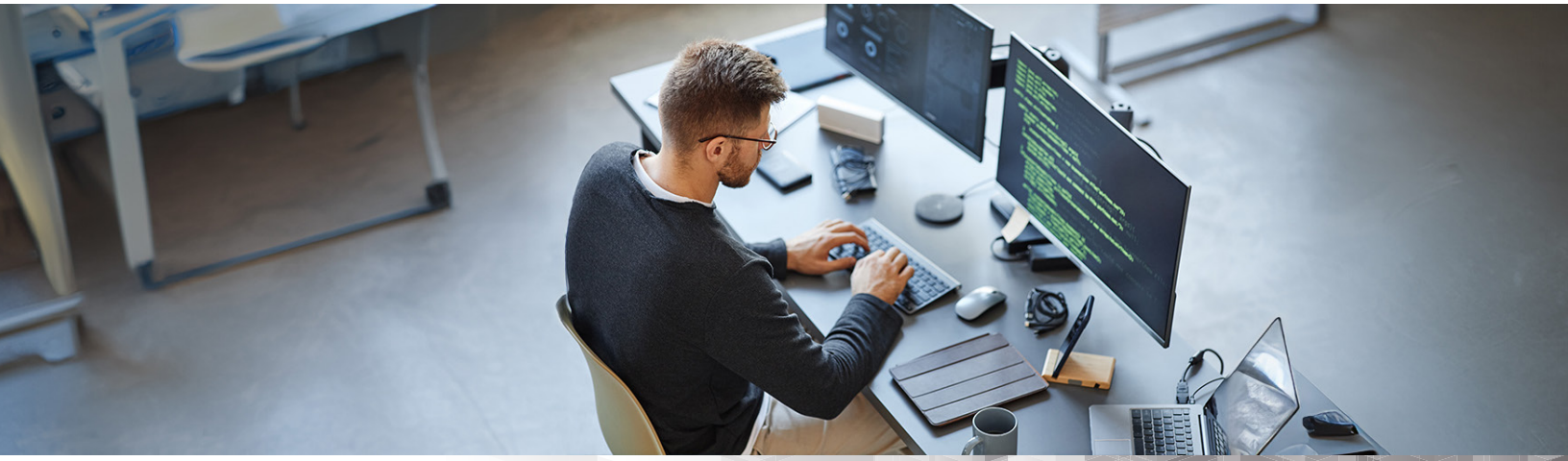
Los usuarios finales prueban una nueva funcionalidad de reporte en una herramienta de gestión de proyectos para asegurarse de que genera informes correctos y útiles.

7. Pruebas Exploratorias

Las pruebas exploratorias se llevan a cabo sin un plan de prueba formal, permitiendo a los testers explorar el software y descubrir errores de manera espontánea. Estas pruebas son valiosas para identificar problemas que pueden no ser evidentes en pruebas estructuradas y para mejorar la calidad general del software.

Ejemplo:

Un tester navega por una nueva aplicación de redes sociales, explorando diferentes funciones sin un plan específico, buscando comportamientos inusuales o errores.



Material complementario

Para profundizar en los tipos de testing te recomendamos acceder al siguiente material complementario:

GeeksforGeeks. (2024, July 12). Difference between Static and Dynamic Testing. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing/

Cierre e ideas fuerza

En esta Guía hemos definido los diferentes tipos de pruebas en el desarrollo de software, enfocando en sus usos posibles y alcances.

Ideas Fuerza

- La aplicación efectiva de diferentes tipos de testing es crucial para garantizar la calidad y fiabilidad del software durante su ciclo de vida.
- Cada tipo de prueba aborda aspectos específicos del software, desde su funcionalidad hasta su rendimiento y usabilidad.
- El uso adecuado de herramientas de testing potencia este proceso, permitiendo una gestión más eficiente y eficaz de los casos de prueba.



3. Guía sobre cómo detectar la complejidad del testing

Introducción

La complejidad del testing en el desarrollo de software es un aspecto crítico que puede influir significativamente en la calidad del producto final. Detectar y clasificar la complejidad del testing permite una planificación más precisa y una asignación adecuada de recursos, asegurando que se aborden todos los aspectos importantes del sistema bajo prueba.

En esta guía, se presentarán los criterios y metodologías para identificar y clasificar la complejidad del testing de acuerdo con el uso y las características del software.



Definición de Complejidad en Testing

Complejidad Intrínseca: Se refiere a la complejidad inherente del software, basada en factores como la lógica del negocio, las dependencias entre módulos y la arquitectura del sistema.

Complejidad Extrínseca: Se relaciona con factores externos que afectan el proceso de testing, como los requisitos del cliente, las regulaciones de la industria y el entorno operativo del software.

Criterios para Detectar la Complejidad del Testing

Número de Funcionalidades: Cuantas más funcionalidades tenga el software, mayor será la complejidad del testing.

Interdependencia de Módulos: La interacción entre múltiples módulos puede aumentar la complejidad debido a las posibles fallas en la integración.

Tecnologías Utilizadas: El uso de tecnologías avanzadas o diversas puede complicar el proceso de testing.

Requisitos No Funcionales: Aspectos como la seguridad, el rendimiento y la usabilidad añaden capas adicionales de complejidad.

Métodos para Clasificar la Complejidad del Testing

Análisis de Puntos de Función: Una técnica que mide la funcionalidad del software desde la perspectiva del usuario final, proporcionando una base para estimar la complejidad del testing.

Modelo de McCabe: Calcula la complejidad ciclomática del software, lo que ayuda a determinar la cantidad de pruebas necesarias para cubrir todas las rutas posibles del código.

Evaluación de Impacto: Analiza cómo los cambios en el software afectan otras partes del sistema, ayudando a priorizar las pruebas en áreas más complejas.

Clasificación de la Complejidad

Baja Complejidad: Sistemas con funcionalidades básicas, pocas interdependencias y uso de tecnologías estándar.

Media Complejidad: Sistemas con un número moderado de funcionalidades, algunas dependencias y tecnologías mixtas.

Alta Complejidad: Sistemas con muchas funcionalidades, alta interdependencia entre módulos y uso de tecnologías avanzadas o personalizadas.

Para Herramientas de Apoyo

Herramientas de Gestión de Pruebas: Software como JIRA, TestRail y Zephyr que facilitan la planificación, ejecución y seguimiento de pruebas.

Herramientas de Automatización: Selenium, QTP y TestComplete que ayudan a manejar pruebas repetitivas y complejas de manera eficiente.

Análisis Estático y Dinámico: Utilización de herramientas como SonarQube y JProfiler para evaluar el código estático y dinámico, identificando áreas de alta complejidad.



Material complementario

Para profundizar en la complejidad en el testing te proponemos visitar el siguiente material complementario:

2023 IEEE 16th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) : 16-20 April 2023, Dublin, Ireland : Proceedings. Los Alamitos, California: IEEE, 2023. https://inacap.primo.exlibrisgroup.com/permalink/56INACAP_INST/it4brd/alma991000677794506726

TEDx Talks (2019, 9 de mayo) El testing de software nos salva cada día | Eneritz Zubizarreta | TEDxVitoriaGasteiz. [Video] YouTube. https://youtu.be/WfkVDkT4_Vg?si=3lizXw7dYvqjSer5

Cierre e ideas fuerza

En esta Guía hemos desarrollado los criterios y metodologías para clasificar la complejidad del testing en función de su uso y características del software. Los equipos de testing utilizan estrategias y metodologías para manejar esta complejidad y asegurar que el software cumpla con los estándares de calidad y funcionalidad esperados.

Ideas Fuerza

- Detectar y clasificar la complejidad del testing es esencial para garantizar un proceso de pruebas efectivo y eficiente.
- Al identificar correctamente los factores que contribuyen a la complejidad y utilizar herramientas y técnicas adecuadas, las organizaciones pueden mejorar la calidad de sus productos de software y reducir los riesgos asociados con la implementación.
- La planificación adecuada y la utilización de métricas de complejidad ayudan a priorizar los esfuerzos de testing, asegurando que se cubran todas las áreas críticas del sistema.



4. Soluciones a problemas y criterios de factibilidad con técnicas de testing

Introducción

En el desarrollo de software, identificar y resolver problemas mediante técnicas de testing es fundamental para asegurar la calidad del producto final. La aplicación de estas técnicas debe considerar soluciones tentativas y atender criterios de factibilidad que se alineen con los estándares y modelos de la industria. Esta guía aborda la ejecución de técnicas de testing, la identificación de soluciones a problemas y la evaluación de su factibilidad.



Definición y Propósito de las Técnicas de Testing

El propósito fundamental de las pruebas de testing es garantizar que el software sea funcional, seguro, confiable y cumpla con las expectativas del usuario y los requisitos del negocio. Esto se logra a través de la aplicación de diversas técnicas de prueba, que se adaptan al ciclo de vida del desarrollo del software.

Técnicas Estáticas: Incluyen revisiones, inspecciones y análisis estático de código sin ejecutar el software.

Técnicas Dinámicas: Involucran la ejecución del software y comprenden pruebas de caja negra y caja blanca.

Identificación de Problemas Comunes en el Desarrollo de *Software*

Defectos Funcionales: Fallas en las funcionalidades previstas del software.

Defectos de Rendimiento: Problemas relacionados con la velocidad, capacidad de respuesta y eficiencia del software.

Problemas de Usabilidad: Dificultades que enfrentan los usuarios al interactuar con el software.

Problemas de Seguridad: Vulnerabilidades que pueden ser explotadas para comprometer el sistema.

Soluciones Tentativas a Problemas

Pruebas Unitarias: Detectar y solucionar errores en componentes individuales del software.

Pruebas de Integración: Identificar y resolver problemas en la interacción entre diferentes módulos.

Pruebas de Sistema: Asegurar que el sistema completo funciona correctamente bajo condiciones específicas.

Pruebas de Aceptación: Verificar que el software cumple con los requisitos del usuario y los criterios de aceptación.

Criterios de Factibilidad

Disponibilidad de Recursos: Evaluar si se cuenta con las herramientas, tiempo y personal necesarios para realizar las pruebas.

Costo-Beneficio: Analizar el costo de implementar la solución frente al beneficio que se obtendrá.

Impacto en el Proyecto: Considerar cómo la solución afectará el cronograma, la calidad y el alcance del proyecto.

Compatibilidad con el Entorno de Desarrollo: Verificar si las técnicas de testing son compatibles con las tecnologías y procesos utilizados.

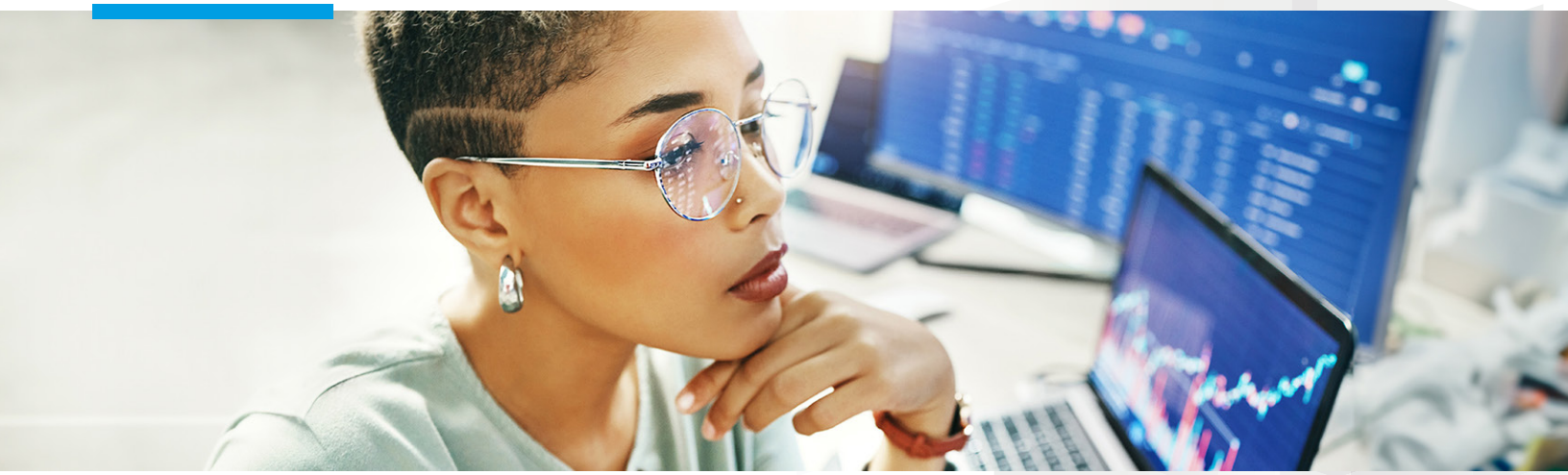
Modelos Existentes en la Industria

Existen varios modelos y enfoques que guían las prácticas de testing para asegurar la calidad y eficiencia en la entrega de software. Estos modelos representan diferentes filosofías y prácticas adaptadas a las necesidades y exigencias específicas de cada proyecto y organización en la industria del software.

V-Model: Un enfoque de testing que se realiza en paralelo con el desarrollo, donde cada fase de desarrollo tiene una fase de testing correspondiente.

Agile Testing: Enfoque iterativo que integra pruebas continuas a lo largo del ciclo de desarrollo, promoviendo la colaboración entre equipos de desarrollo y testing.

DevOps: Combina desarrollo y operaciones para mejorar la colaboración y la entrega continua, integrando testing automatizado en el proceso de entrega.



Ejecución de Técnicas de Testing

Planificación de Pruebas: Definir los objetivos, alcance, enfoque y recursos necesarios para las pruebas.

Diseño de Casos de Prueba: Crear escenarios detallados que cubran todos los requisitos y especificaciones del software.

Ejecución de Pruebas: Realizar las pruebas según lo planificado y registrar los resultados.

Análisis de Resultados: Evaluar los resultados de las pruebas para identificar defectos y áreas de mejora.

Informe de Pruebas: Documentar los hallazgos, conclusiones y recomendaciones para futuras acciones.



Cierre e ideas fuerza

En esta guía hemos analizado cómo la correcta ejecución de técnicas de testing, junto con la consideración de soluciones tentativas y criterios de factibilidad, es crucial para asegurar la calidad del software. La selección y aplicación de técnicas adecuadas, basadas en modelos establecidos y criterios de factibilidad, permite enfrentar de manera efectiva los desafíos del testing en el desarrollo de software, contribuyendo así a la entrega de productos confiables y de alta calidad.

Ideas Fuerza

- La clave para unas pruebas efectivas reside en una planificación meticulosa que defina objetivos claros, recursos adecuados y un enfoque estratégico.
- Evaluar los resultados de las pruebas de manera rigurosa es fundamental para identificar y corregir defectos, asegurando la calidad del software.
- El informe de pruebas bien estructurado no solo documenta hallazgos y recomendaciones, sino que también sirve como guía para futuras iteraciones y mejoras del producto.



Bibliografía

IEEE. (1990). IEEE Standard for Software Test Documentation (IEEE Std 829-1990).

IEEE 16th International Conference on Software Testing, Verification and Validation Workshops (ICSTW) : 16-20 April 2023, Dublin, Ireland : Proceedings. Los Alamitos, California: IEEE, 2023. https://inacap.primo.exlibrisgroup.com/permalink/56INACAP_INST/it4brd/alma991000677794506726

GeeksforGeeks. (2024, July 12). Difference between Static and Dynamic Testing. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-static-and-dynamic-testing/

Kaner, C. & Bach, J. (2004) The Nature of Exploratory Testing. Cem Kaner, J.D., Ph.D. https://kaner.com/pdfs/NatureOfExploratoryTest.pdf

Lee, Jihyun & Kang, Sungwon & Lee, D. (2012). Survey on software testing practices. Software, IET. V6. p.275-282. DOI: 10.1049/iet-sen.2011.0066. https://www.researchgate.net/publication/260649940_Survey_on_software_testing_practices

McCabe, T. (1976). A Complexity Measure. IEEE Transactions on Software Engineering, SE-2(4), p. 308-320. https://ieeexplore.ieee.org/document/1702388

Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). John Wiley & Sons.

Pressman, R. S. y Maxim, B. R. (2014). Software Engineering: A Practitioner's Approach (8.a ed.). McGraw Hill Education.

Sommerville, I. (2011). Software Engineering (9th ed.). Addison-Wesley.

TEDx Talks (2019, 9 de mayo) El testing de software nos salva cada día | Eneritz Zubizarreta | TEDxVitoriaGasteiz. [Video] YouTube. https://youtu.be/WfkVDkT4_Vg?si=3lizXw7dYvqjSer5

TelefónicaTech (2014, 29 de diciembre). QA pruebas para asegurar la calidad. https://telefonicatech.com/blog/qa-pruebas-para-asegurar-la-calidad-de