Informe sobre Testing y Pruebas de Código en el Desarrollo de Software

Introducción

El testing y las pruebas de código son aspectos cruciales en el ciclo de vida del desarrollo de software. Estas prácticas no solo ayudan a identificar y corregir errores antes de que el software llegue al usuario final, sino que también garantizan que el producto cumpla con los requisitos y estándares de calidad establecidos. El testing efectivo mejora la confiabilidad, seguridad y rendimiento del software, lo que a su vez conduce a una mayor satisfacción del cliente y un ciclo de desarrollo más eficiente.

Conceptos Básicos

Definición y Diferencia entre Testing y Pruebas de Código

- Testing: Proceso de verificar y validar la funcionalidad de un programa o una aplicación de software para garantizar que esté libre de defectos y coincida con los requisitos esperados, entregando así un producto de calidad.
- Pruebas de Código: Conjunto de técnicas y procesos utilizados durante el testing para validar que el código cumple con los requisitos y funciona correctamente.

Objetivos y Beneficios de Realizar Pruebas

- Detectar defectos y asegurar la calidad.
- Verificar que el software cumple con los requisitos especificados.

- Reducir costos a largo plazo al encontrar y solucionar problemas tempranamente.
- Mejorar la confianza en la estabilidad del software.

Tipos de Pruebas

Pruebas Unitarias:

- Descripción: Pruebas de bajo nivel realizadas cerca de la fuente de la aplicación para probar métodos y funciones individuales de clases, componentes o módulos.
- Ventajas: Automatización barata y rápida ejecución.
- Herramientas: JUnit (Java), NUnit (.NET), Pytest (Python).

Pruebas de Integración:

- Descripción: Verifican que los distintos módulos o servicios de la aplicación funcionen bien en conjunto. Por ejemplo, se puede probar la interacción con la base de datos.
- Herramientas: TestNG (Java), XUnit (.NET).

Pruebas de Sistema:

- Descripción: Evaluación del sistema completo para verificar que cumple con los requisitos especificados.
- Herramientas: Selenium, QTP.

Pruebas de Aceptación:

- Descripción: Pruebas formales que verifican si un sistema satisface los requisitos empresariales. Este requiere que se esté ejecutando toda la aplicación durante las pruebas y se centren en replicar las conductas del usuario.
- Herramientas: Cucumber, FitNesse.

Pruebas de Rendimiento:

- Descripción: Evalúan el rendimiento de un sistema bajo una carga de trabajo específica. Ayudan a medir la fiabilidad, la velocidad, la escalabilidad y las capacidades de respuesta de una aplicación.
- Herramientas: JMeter, LoadRunner.

Pruebas de Humo:

- Descripción: Pruebas básicas para comprobar el funcionamiento básico de la aplicación. Concebidas para ejecutarse rápidamente y su objetivo es ofrecerte la seguridad de que las principales funciones de tu sistema funcionen según lo previsto.
- Herramientas: Selenium, Appium, Testlink.

Pruebas de Extremo a Extremo:

- Descripción: Replican el comportamiento de un usuario con el software en un entorno de aplicación completo. Verifican que diversos flujos de usuarios funcionen según lo previsto, y pueden ser tan sencillos como cargar una página web o iniciar sesión.
- Herramientas: Selenium, Katalon, Watir.

Técnicas de Testing

Test Driven Development (TDD):

- Descripción: Práctica de programación que consiste en escribir primero las pruebas, luego el código fuente que pase la prueba satisfactoriamente, y por último, refactorizar el código escrito.
- Beneficios: Código más robusto, seguro, mantenible y desarrollo rápido.
- Retos: Cambio de mentalidad y posible lentitud inicial.
- Proceso: Escritura de historias de usuario, definición de criterios de aceptación, traducción a pruebas unitarias, desarrollo de código para pasar la prueba, refactorización y repetición del ciclo.

Behavior Driven Development (BDD):

- Descripción: Proceso ágil que busca la colaboración entre desarrolladores, gestores de proyecto y equipo de negocio.
- Lenguaje Gherkin y Principios de BDD: Uso de un lenguaje de dominio específico con elementos como Given-When-Then para definir funcionalidades y escenarios.
- Beneficios: Colaboración mejorada, enfoque en el valor de negocio, reducción de riesgo y metodologías ágiles.
- Retos: Involucramiento constante de todas las partes.

Automatización de Pruebas

- **Introducción**: Aplicación de herramientas de software para automatizar el proceso manual de revisión y validación de un producto de software.
- Tipos de Pruebas a Automatizar: Pruebas de extremo a extremo, unitarias, de integración y de rendimiento.
- Ventajas: Mayor eficiencia, consistencia y cobertura de pruebas.
- Herramientas y Frameworks: Selenium (web), Appium (móvil), Robot Framework.

Casos de Uso y Ejemplos

Ejemplo TDD: Desarrollo de una Calculadora

- Criterio de Aceptación: Sumar dos números y mostrar el resultado.
- Proceso: Definición del funcionamiento del algoritmo de suma, creación de pruebas, desarrollo de la clase Calculadora, ejecución de pruebas, refactorización y repetición para casos más complejos.

Ejemplo de Prueba Unitaria en Python con Pytest:

- Función a Probar: sumar(a, b)
- Proceso: Instalación de Pytest, escritura de pruebas para diferentes casos, ejecución de pruebas y verificación de resultados.

Conclusión

Las pruebas y el testing son indispensables para garantizar la calidad y confiabilidad del software. A través de diversas técnicas y herramientas, los desarrolladores pueden identificar y corregir errores de manera eficiente, mejorando así la calidad del producto final. La adopción de prácticas como TDD y BDD, junto con la automatización de pruebas, representa una inversión significativa en la calidad y sostenibilidad a largo plazo del desarrollo de software.

Referencias:

- Martínez Canelo, M. (2021) *Qué es el testing de software*. Profile Software Services. Disponible en: https://profile.es/blog/que-es-el-testing-de-software/

- Atlassian. (s.f.) *Los distintos tipos de pruebas en software*. Disponible en: https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing
- Singureanu, C. (2023) *Pruebas de humo: tipos, proceso, herramientas y mucho más*. Zaptest. Disponible en: https://www.zaptest.com/es/smoke-testing-profundizacion-en-tipos-proceso-herramientas-de-software-de-smoke-test-y-mas
- Singureanu, C. (2023) *Pruebas de extremo a extremo: Profunda inmersión en los tipos de pruebas E2E, procesos, enfoques, herramientas y mucho más*. Zaptest. Disponible en: https://www.zaptest.com/es/pruebas-de-extremo-a-extremo-profunda-inmersion-en-los-tipos-de-pruebas-e2e-procesos-enfoques-herramientas-y-mucho-mas
- Herranz, J.I. (2023) *TDD como metodología de diseño de software*. Paradigma. Disponible en: https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/
- Zapater, S. (2022) *BDD Testing. ¿Cómo funciona el Behavior Driven Development?*. Hiberus. Disponible en: https://www.hiberus.com/crecemos-contigo/bdd-behavior-driven-developement/
- Atlassian. (s.f.) *Pruebas de software automatizadas para la entrega continua*. Disponible en: https://www.atlassian.com/es/continuous-delivery/software-testing/automated-testing
- McMullin, B. (2021) *Cómo escribir casos de prueba para software: ejemplos y tutorial*. Parasoft. Disponible en: https://es.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/
- Como corrector tanto de forma como gramaticalmente: OpenAI. (2023) ChatGPT, versión 4. [software de inteligencia artificial]. Disponible en: https://www.openai.com/