nacap

Gestión de defectos, combinación de valores de prueba y exploración



Tabla de CONTENIDOS

1. Requerimientos del desarrollo de software 3
Introducción 3
Plan de pruebas 3
Introducción al plan de pruebas 3
Objetivos del Plan de Prueba 4
Alcance del Plan de Prueba 4
Estrategia de Pruebas 4
Criterios de Aceptación y Finalización 4
Recursos y Cronograma 5
Gestión de Defectos 5
Evaluación y Revisión 5
PDCA (Planificar, Hacer, Verificar, Actuar) 6
Material ampliatorio 8
Cierre e ideas Fuerza 8
Ideas Fuerza 8

2. Técnicas para pruebas exploratorias 9
Introducción 9
Fundamentos de las pruebas exploratorias 10
Técnicas utilizadas en pruebas exploratorias 10
Beneficios y desafíos de las pruebas exploratorias 11
Ejemplo práctico 11
Material complementario 12
Cierre e ideas fuerza 12

3. Ciclo de vida de un defecto de software 13
Introducción 13
¿Qué es el ciclo de vida de un defecto? 13
Detección del defecto 14
Clasificación y priorización 14
Asignación y resolución 14
Verificación 14
Cierre 14
Reapertura (si es necesario) 14
Diagrama de flujo: ciclo de vida de un defecto de software 15
Mejores prácticas en la gestión de defectos 16
Material complementario 17
Cierre e ideas fuerza 17
Ideas fuerza 17

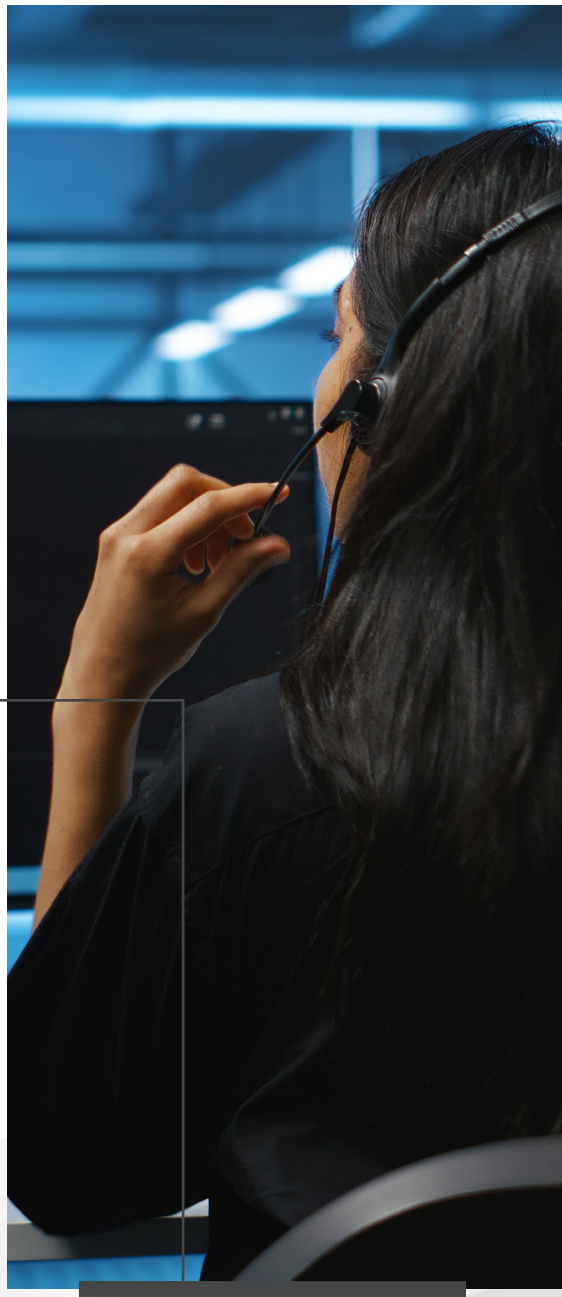
4. Defectos, errores y fallos según criterios de factibilidad de Plan de Prueba 18
Introducción 18
Conceptos básicos 18
Defectos 19
Errores 19
Fallos 19
Ciclo de Vida: De Error a Fallo 19
Análisis y documentación de hallazgos 20
Uso de imágenes o diagramas 21
Cierre e ideas fuerza 22
Ideas Fuerza 22

5. Guía: Procesos de seguimiento en el monitoreo 23
Introducción 23
Seguimiento de fallos, errores y defectos 24
Ejemplo 25
Material complementario 26
Cierre e ideas fuerza 26
Ideas Fuerza 26

Bibliografía 27

1. Requerimientos del desarrollo de software

Introducción



La definición de un plan de prueba según los requerimientos del desarrollo de software es una etapa crítica en el ciclo de vida del desarrollo. Este plan sirve como una hoja de ruta detallada que guía a los equipos de pruebas en la verificación y validación de un producto de software, asegurando que cumple con las especificaciones y necesidades del cliente. El plan de prueba debe abordar todos los aspectos del sistema, desde las pruebas de unidad y de integración hasta las pruebas de aceptación del usuario. Incluirá objetivos claros, estrategias de prueba, criterios de entrada y salida, cronograma, y los recursos necesarios. Al hacerlo, se garantiza que las pruebas sean exhaustivas y efectivas, minimizando los riesgos y mejorando la calidad del software final.

Plan de pruebas

Introducción al plan de pruebas

Un plan de prueba es un documento que describe el alcance, enfoque, recursos y calendario de las actividades de prueba previstas. Este documento es fundamental para garantizar que el producto de software se pruebe de manera adecuada y completa, asegurando su calidad y funcionalidad.

Objetivos del Plan de Prueba

Los objetivos del plan de prueba incluyen:

- Verificar que el software cumple con los requisitos especificados.
- Detectar defectos en el software antes de su liberación.
- Asegurar que el software sea confiable y funcione correctamente bajo diversas condiciones.

Alcance del Plan de Prueba

El alcance del plan de prueba debe detallar las características del software que se probarán y aquellas que no se incluirán en las pruebas. Esto incluye:

- **Características a probar:** Funcionalidades principales, integraciones, rendimiento, seguridad, etc.
- **Características que no se prueban:** Funcionalidades de baja prioridad o aquellas que se probarán en fases posteriores.

Estrategia de Pruebas

La estrategia de pruebas define el enfoque general para las pruebas y puede incluir:

- **Técnicas de Prueba:** Pruebas estáticas (revisión de documentos, análisis estático) y dinámicas (pruebas funcionales y no funcionales).
- **Tipos de Prueba:** Pruebas unitarias, pruebas de integración, pruebas de sistema, pruebas de aceptación.
- **Criterios de Prioridad:** Basados en la criticidad y el impacto de los defectos.

Criterios de Aceptación y Finalización

Los criterios de aceptación y finalización establecen las condiciones bajo las cuales las pruebas se considerarán satisfactorias y completadas:

- **Criterios de Entrada:** Condiciones que deben cumplirse antes de iniciar una fase de prueba (por ejemplo, disponibilidad del entorno de prueba, casos de prueba preparados).
- **Criterios de Salida:** Condiciones para finalizar una fase de prueba (por ejemplo, porcentaje de casos de prueba ejecutados y aprobados, severidad de defectos).

Recursos y Cronograma

El plan de prueba debe identificar los recursos necesarios, incluidos el equipo de pruebas, las herramientas y los entornos de prueba, así como un cronograma detallado:

- **Equipo de Pruebas:** Roles y responsabilidades de los miembros del equipo.
- **Herramientas de Pruebas:** Software y hardware necesarios.
- **Cronograma:** Calendario detallado de actividades de prueba.

Gestión de Defectos

La gestión de defectos es crucial para asegurar la calidad del software. Esto incluye la identificación, registro, clasificación y seguimiento de defectos hasta su resolución:

- **Registro de Defectos:** Herramientas y procesos para registrar y rastrear defectos.
- **Clasificación de Defectos:** Priorización basada en la severidad y el impacto.
- **Resolución y Verificación:** Procesos para la corrección y revalidación de defectos.

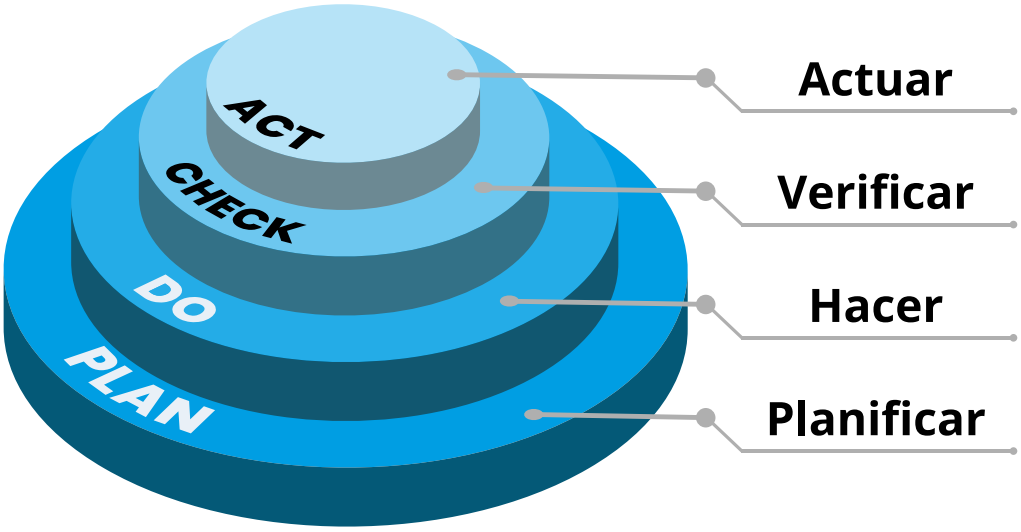
Evaluación y Revisión

La evaluación y revisión del plan de prueba aseguran su efectividad y relevancia a lo largo del proyecto:

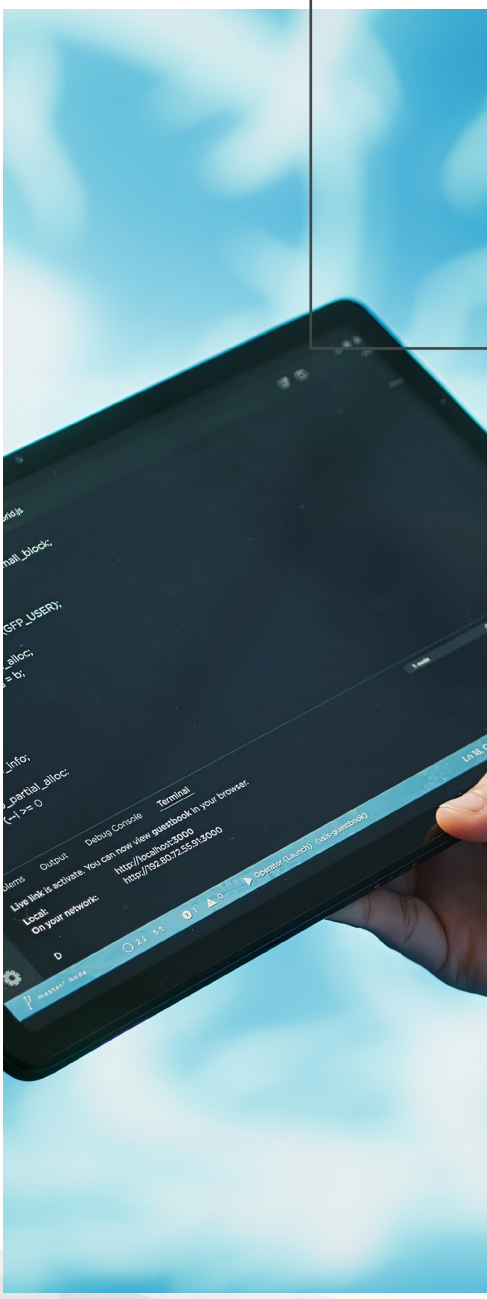
- **Revisiones Regulares:** Revisión periódica del plan de prueba para adaptarse a cambios en los requisitos o el entorno.
- **Mejora Continua:** Identificación y aplicación de mejoras basadas en la experiencia de pruebas previas.

PDCA (Planificar, Hacer, Verificar, Actuar)

La rueda de Deming, también conocida como ciclo PDCA (Plan-Do-Check-Act) se utiliza como método para resolver problemas e implementar soluciones de forma rigurosa y metódica. Estos principios son aplicables en la gestión del plan de pruebas de software, y podrían sintetizarse de la siguiente manera:



Fuente: elaboración propia.



Planificar (Plan):

Definir el alcance, identificar los recursos necesarios (personal, herramientas de prueba, entornos, etc.) y elaborar un plan detallado de pruebas que incluya la estrategia de pruebas, los casos de prueba y los criterios de aceptación.

Hacer (Do):

Ejecutar las actividades de prueba según el plan establecido. Esto incluye la ejecución de casos de prueba, la recolección de datos y la documentación de los resultados obtenidos.

Verificar (Check):

Evaluar los resultados obtenidos, analizar si los resultados cumplen con los criterios de aceptación definidos, identificar defectos y áreas de mejora, y recopilar métricas de calidad relevantes (por ejemplo, tasa de defectos encontrados, cobertura de pruebas, etc.).

Actuar (Act):

Basándose en los hallazgos y análisis de las pruebas, se toman acciones correctivas o preventivas. Esto puede incluir ajustes en los casos de prueba, mejoras en la estrategia de pruebas, corrección de defectos encontrados, actualización de la documentación, entre otros. software, y podrían sintetizarse de la siguiente manera:



Material complementario

Para profundizar en el plan de pruebas de software te proponemos acceder al siguiente material complementario:

MindTools | Home. (s.f.). PDCA (Plan Do Check Act) https://www.mindtools.com/as2l5i1/pdca-plan-do-check-act

Kumar, Sandeep, and Santosh Singh Rathore. (2018) Software Fault Prediction: A Road Map. 1st ed. Singapore: Springer Singapore. https://inacap.primo.exlibrisgroup.com/permalink/56INACAP_INST/di6hvk/cdi_askewsholts_vlebooks_9789811087158

Cierre e ideas fuerza

Hasta aquí hemos presentado la estructura del plan de pruebas de software y su importancia para el éxito de cualquier proyecto en la industria. Destacamos los elementos clave para alcanzar la máxima calidad y confiabilidad del software desarrollado.

Ideas Fuerza

- Un plan de prueba efectivo es crucial para garantizar el éxito de cualquier proyecto de desarrollo de software.
- Una planificación meticulosa, una estrategia de pruebas robusta y una gestión eficaz de recursos y defectos son fundamentales para asegurar que el producto de software cumpla con los requisitos y expectativas del cliente.
- La implementación de un plan de prueba bien estructurado no solo asegura la funcionalidad del software, sino que también contribuye a su calidad y fiabilidad, esenciales para satisfacer las demandas del mercado y la satisfacción del usuario final.



2. Técnicas para pruebas exploratorias



Introducción

Las pruebas exploratorias son una técnica dinámica de testing que se enfoca en la simultaneidad de la ejecución y el diseño de las pruebas. A diferencia de las pruebas tradicionales que siguen scripts predefinidos, las pruebas exploratorias se adaptan sobre la marcha, permitiendo a los testers descubrir problemas de manera más intuitiva y creativa. Esta técnica es especialmente útil cuando se exploran nuevas áreas del software o se intenta identificar defectos que no han sido previstos en los planes de prueba iniciales.

Fundamentos de las pruebas exploratorias

Las pruebas exploratorias se basan en la experiencia y el conocimiento del tester. En lugar de seguir un conjunto de pasos predefinidos, los testers exploran el software de manera libre, adaptando sus pruebas en función de lo que encuentran. Este enfoque se apoya en varios principios fundamentales:

1. **Simultaneidad de diseño y ejecución:** El diseño y la ejecución de las pruebas ocurren al mismo tiempo. Esto permite una mayor flexibilidad y adaptabilidad durante el proceso de prueba.
2. **Ciclos cortos de feedback:** Las pruebas exploratorias generan ciclos rápidos de retroalimentación, permitiendo a los testers ajustar sus estrategias de prueba en tiempo real.
3. **Creatividad y adaptabilidad:** Los testers utilizan su creatividad y adaptan sus pruebas sobre la marcha, basándose en su experiencia y en los resultados obtenidos.

Técnicas utilizadas en pruebas exploratorias

Existen varias técnicas que se pueden aplicar durante las pruebas exploratorias para maximizar su efectividad:

1. Tours de Exploración: Esta técnica implica recorrer diferentes áreas del software como si se estuviera en una excursión, cada "tour" se enfoca en un aspecto específico del software, como la interfaz de usuario, la funcionalidad de back-end, o la integración de sistemas.
2. Listas de Verificación (Checklists): Las listas de verificación proporcionan una guía general sobre qué áreas del software deben ser exploradas, pero no dictan cómo hacerlo. Esto permite a los testers mantener un enfoque flexible mientras aseguran una cobertura adecuada.
3. Matriz de Riesgos: Utilizar una matriz de riesgos ayuda a identificar y priorizar áreas del software que son más propensas a tener defectos. La matriz considera factores como la complejidad del código, la frecuencia de uso y el impacto potencial de los defectos.
4. Notaciones de Exploración: Registrar observaciones y resultados durante las pruebas es crucial. Se utilizan notaciones para documentar los descubrimientos, problemas encontrados y áreas que requieren mayor atención.
5. Heurísticas de Prueba: Las heurísticas son reglas prácticas que guían a los testers en la exploración del software. Algunas heurísticas comunes incluyen: "Cada entrada debe tener una salida", "Pruebas negativas deben ser realizadas", y "Pruebas de límites son esenciales".

Beneficios y desafíos de las pruebas exploratorias

Beneficios:

- **Mayor flexibilidad y adaptabilidad:** Permiten ajustes rápidos y dinámicos en el enfoque de pruebas.
- **Descubrimiento de defectos inesperados:** Al no estar limitado a scripts predefinidos, los testers pueden descubrir problemas que de otro modo pasarían desapercibidos.
- **Mejora continua:** Los testers aprenden y mejoran sus estrategias en cada ciclo de prueba.



Desafíos:

- **Dependencia de la habilidad del tester:** La efectividad de las pruebas exploratorias depende en gran medida de la experiencia y creatividad del tester.
- **Documentación:** Registrar adecuadamente las pruebas y sus resultados puede ser más complicado que con pruebas basadas en scripts.
- **Cobertura de pruebas:** Asegurar una cobertura adecuada puede ser un desafío, ya que no se sigue un plan de prueba predefinido.

Ejemplo práctico

Para ilustrar cómo se aplican estas técnicas en un entorno real, consideremos una aplicación de gestión de proyectos. Un tester podría iniciar con un tour de exploración enfocándose en la interfaz de usuario, verificando que todas las funciones básicas (como crear, editar y eliminar proyectos) funcionen correctamente. Utilizando una lista de verificación, el tester se asegura de cubrir todas las áreas importantes, como la gestión de usuarios y la integración con otras herramientas. Una matriz de riesgos ayuda a priorizar la exploración de áreas complejas y críticas, mientras que las notaciones de exploración documentan los hallazgos y problemas en tiempo real.

Material complementario

Para profundizar en las pruebas exploratorias te proponemos acceder al siguiente material complementario:

Gonzalez, Marta (s/f) Testing Exploratorio: Una guía práctica. Marta González Desarrollo Front-End https://martagonzalez.dev/blog/testing-exploratorio-una-guia-integral/

Kaner, C. (2006, November 17). Exploratory Testing Explained. Cem Kaner, J.D., Ph.D. Keynote at QAI https://kaner.com/pdfs/ETatQAI.pdf

Cierre e ideas fuerza

Hasta acá hemos presentado los fundamentos de las técnicas exploratorias, las principales técnicas utilizadas, sus beneficios y desafíos.

Ideas Fuerza

- Las pruebas exploratorias son una técnica esencial en el arsenal de cualquier tester, ofreciendo una forma flexible y dinámica de descubrir defectos.
- Las pruebas exploratorias van más allá de la simple ejecución de pruebas. El tester debe ser creativo y utilizar su intuición para identificar patrones, anomalías y oportunidades de mejora en el software.
- Aplicando técnicas como tours de exploración, listas de verificación, matrices de riesgos, notaciones de exploración y heurísticas de prueba, los testers pueden maximizar la efectividad de sus esfuerzos y garantizar un software de alta calidad.



3. Ciclo de vida de un defecto de software

Introducción

El ciclo de vida de un defecto de software es un proceso esencial en el desarrollo y mantenimiento de software que ayuda a rastrear, gestionar y resolver defectos encontrados durante las pruebas. Comprender este ciclo es crucial para garantizar la calidad del producto final y para identificar áreas de mejora en el proceso de desarrollo.



¿Qué es el ciclo de vida de un defecto?

El desarrollo de software, al igual que cualquier proceso creativo, no está exento de imperfecciones. Es en este contexto donde surge el concepto de defecto de software, definido como cualquier falla o error en el código que impide que el programa funcione correctamente, cumpla con sus requisitos o satisfaga las necesidades de los usuarios.

Para abordar estos defectos de manera eficiente y sistemática, se ha establecido el Ciclo de Vida de un Defecto de Software, un proceso estructurado que define las distintas etapas por las que atraviesa un defecto desde su detección hasta su resolución y cierre. Este ciclo de vida no es lineal, sino que puede ser iterativo, es decir, que un defecto puede volver a una etapa anterior si se requiere más análisis o pruebas.

A continuación, se presenta un resumen general de las fases que componen el Ciclo de Vida de un Defecto de Software.

Detección del defecto

El ciclo de vida de un defecto comienza cuando se detecta un error durante la fase de pruebas. Los testers documentan el defecto en una herramienta de seguimiento, proporcionando detalles como la descripción del problema, los pasos para reproducirlo, el entorno de pruebas y la gravedad del defecto.

Clasificación y priorización

Una vez documentado, el defecto es revisado por el equipo de desarrollo para clasificar su tipo (p. ej., funcional, UI, rendimiento) y asignarle una prioridad basada en su severidad e impacto en el sistema.

Asignación y resolución

El defecto es asignado a un desarrollador específico para su resolución. El desarrollador analiza el defecto, identifica la causa raíz y realiza los cambios necesarios en el código para corregirlo.

Verificación

Después de que el defecto ha sido corregido, se somete a una verificación para asegurarse de que la solución funciona correctamente y no introduce nuevos problemas. Esto generalmente implica una nueva ronda de pruebas, incluyendo pruebas de regresión.

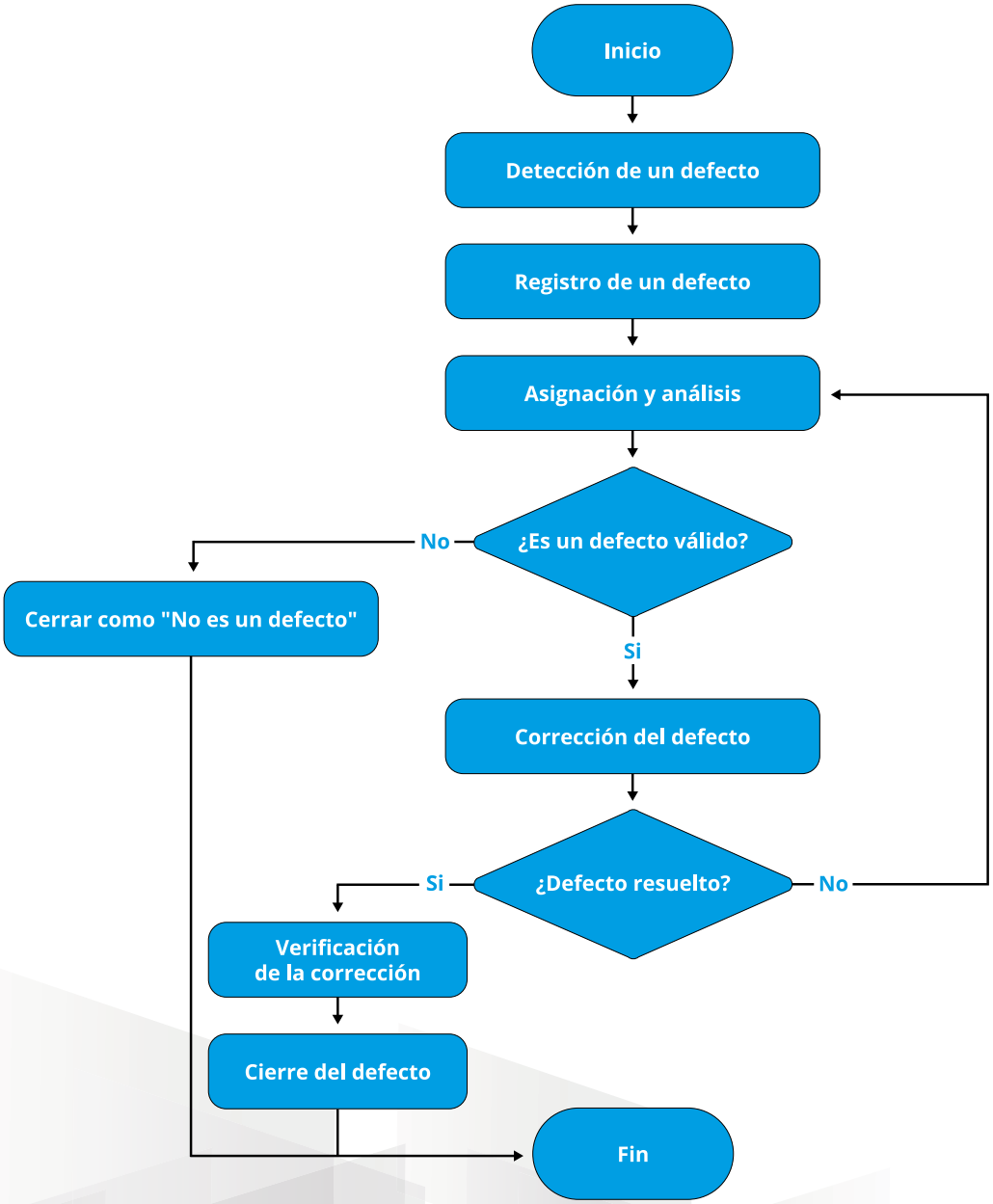
Cierre

Si el defecto ha sido resuelto satisfactoriamente y no se encuentran más problemas durante la verificación, el defecto se marca como cerrado. Esta etapa también incluye la documentación de la resolución para referencia futura.

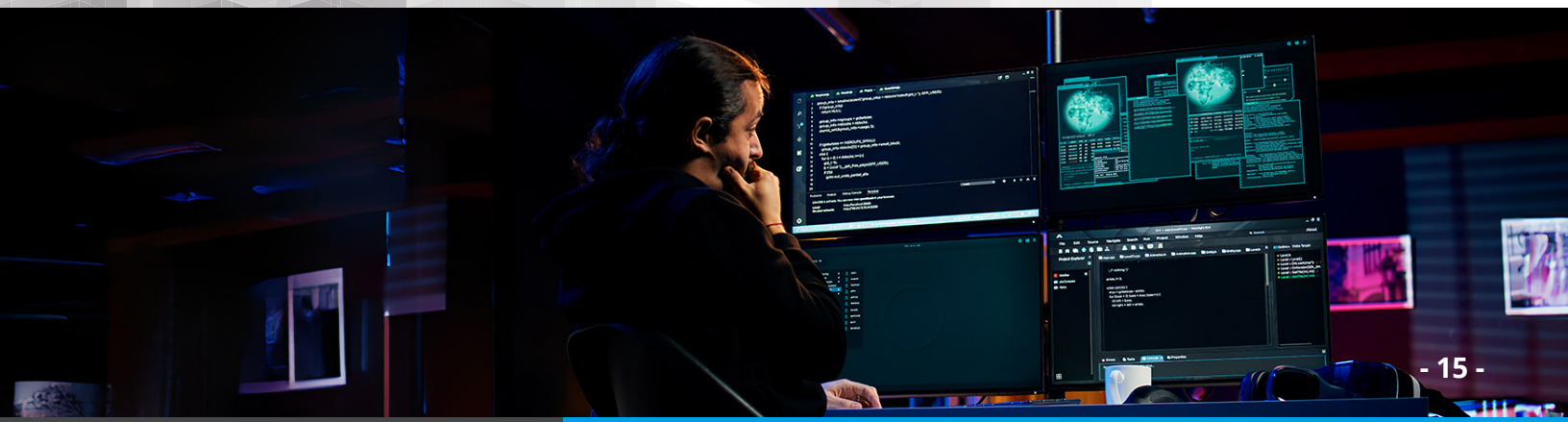
Reapertura (si es necesario)

En algunos casos, los defectos pueden reabrirse si se descubre que no se han solucionado completamente o si el problema persiste bajo ciertas condiciones.

Diagrama de flujo: ciclo de vida de un defecto de software



Elaboración propia.



Mejores prácticas en la gestión de defectos

Algunas de las mejores prácticas para la gestión eficaz de defectos son:

- **Descripción clara y concisa:** Proporcionar toda la información necesaria para reproducir el defecto.
- **Categorización adecuada:** Asignar correctamente la severidad y prioridad de cada defecto.
- **Evitar duplicados:** Verificar si un defecto similar ya ha sido reportado antes de crear uno nuevo.
- **Comunicación efectiva:** Mantener una comunicación clara entre testers, desarrolladores y gestores de proyecto.
- **Seguimiento constante:** Monitorear regularmente el estado de los defectos abiertos.
- **Análisis de tendencias:** Revisar periódicamente los patrones de defectos para identificar áreas problemáticas.
- **Documentación actualizada:** Mantener la documentación del proyecto actualizada con la información de los defectos y sus soluciones.
- **Pruebas de regresión:** Realizar pruebas exhaustivas después de corregir un defecto para evitar efectos secundarios.



Material complementario

Para profundizar en el ciclo de vida de un defecto de software, te proponemos acceder al siguiente material complementario:

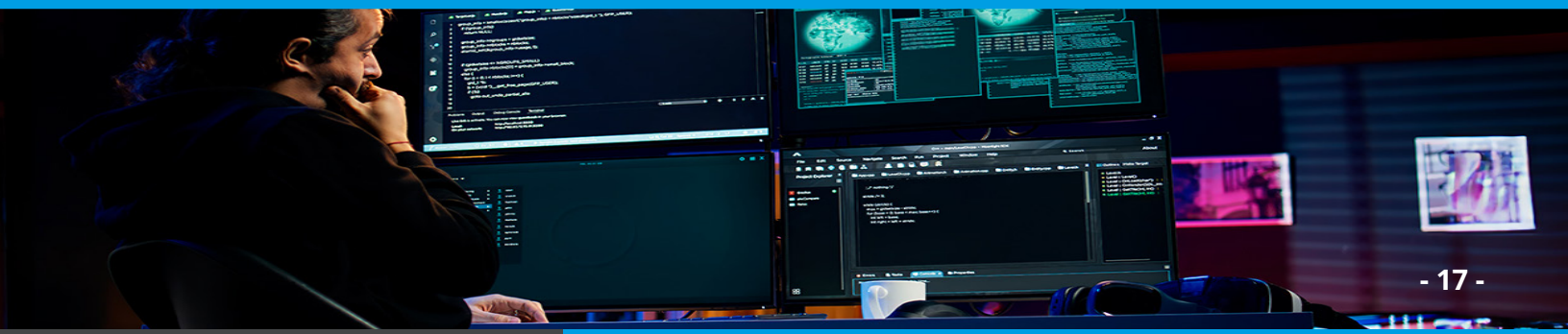
Fillottrani, P. R. (2007). Calidad en el Desarrollo de Software. Métricas de procesos de software. https://controldecalidadcue1.wordpress.com/wp-content/uploads/2011/04/clase5.pdf

Cierre e ideas fuerza

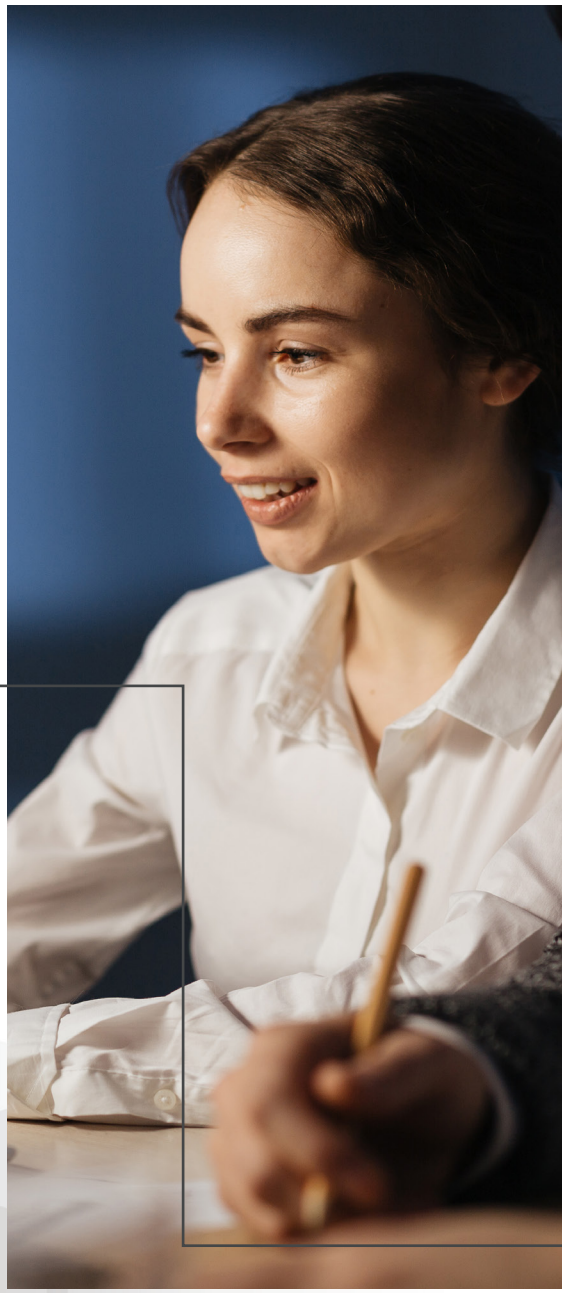
Esta semana hemos aprendido sobre el ciclo de vida de un defecto de software, analizando cada una de sus fases. Remarcamos la importancia de una gestión eficaz de defectos no solo para mejorar la calidad del producto final, sino también para optimizar el proceso de desarrollo al identificar y corregir problemas de manera estructurada y sistemática. Al entender y aplicar el ciclo de vida del defecto, los equipos aseguran la estabilidad y funcionalidad del software y fortalecen su capacidad para responder de manera ágil y eficiente a los desafíos que surjan durante el ciclo de desarrollo.

Ideas Fuerza

- La gestión efectiva de defectos es un componente crucial para el desarrollo de software de alta calidad.
- Un ciclo de vida de defectos bien definido y documentado ayuda a garantizar que los defectos se identifiquen, reporten, analicen, reparen, verifiquen y cierren de manera eficiente.
- La implementación de buenas prácticas en la gestión de defectos, junto con herramientas y métricas adecuadas, puede reducir significativamente el número de defectos y mejorar la calidad del software.



4. Defectos, errores y fallos según criterios de factibilidad de Plan de Prueba



Introducción

Para abordar eficazmente la gestión de calidad en el desarrollo de software, es crucial entender la diferencia entre defectos, errores y fallos. Los defectos representan desviaciones del comportamiento esperado del software, mientras que los errores son las acciones incorrectas o faltas de precisión que conducen a estos defectos. Los fallos, por otro lado, son las manifestaciones observables de los errores durante la ejecución del software. En este contexto, analizar y documentar estos hallazgos dentro del marco de un plan de prueba no solo facilita la identificación temprana de problemas, sino que también permite la aplicación de soluciones viables que aseguren la calidad y la funcionalidad del producto final.

Conceptos básicos

Comprender las diferencias entre defectos, errores y fallos no es solo una cuestión de terminología; es una habilidad fundamental que impacta directamente en la eficacia del proceso de desarrollo y en la calidad del software. Esta distinción permite a los profesionales comunicarse con claridad, priorizar adecuadamente los problemas, y aplicar las estrategias de resolución más apropiadas.

Defectos

Los defectos en el software son problemas o desviaciones en el código que afectan negativamente su funcionamiento. Estos pueden surgir debido a errores humanos durante la programación o a problemas en la interpretación de los requisitos del software. Es crucial identificar y documentar cada defecto de manera precisa, utilizando herramientas y técnicas adecuadas de gestión de pruebas. Esto no solo incluye la detección de errores en el código, sino también la captura de problemas potenciales que podrían afectar la calidad del producto final.

Errores

Los errores son las acciones incorrectas o la falta de precisión en el desarrollo del software que conducen directamente a la aparición de defectos. Estos errores pueden surgir en cualquier etapa del ciclo de vida del desarrollo de software, desde la concepción inicial hasta las pruebas finales. Identificar los errores implica revisar de manera exhaustiva el diseño y la implementación del software para garantizar que cumpla con los requisitos especificados y las expectativas del usuario.

Fallos

Los fallos son las manifestaciones observables de los errores durante la ejecución del software. Estos pueden incluir comportamientos inesperados, errores de funcionamiento o incluso la interrupción completa del programa. La detección de fallos es crucial durante las pruebas funcionales y de aceptación del software, donde se simulan situaciones reales para evaluar su comportamiento bajo diversas condiciones de uso.

Ciclo de Vida: De Error a Fallo

Comprender cómo un error inicial puede llevar a un defecto y finalmente manifestarse como un fallo es crucial para prevenir y manejar problemas en el desarrollo de software. Este ciclo ilustra la progresión de los problemas y ayuda a identificar puntos de intervención.

Ciclo: Error → Defecto → Fallo

- El desarrollador malinterpreta los requisitos (Error)
- Codifica una fórmula incorrecta en el programa (Defecto)
- El programa se ejecuta con datos reales (Activación del defecto)
- El usuario recibe un cálculo de intereses incorrecto (Fallo)

Análisis y documentación de hallazgos

El análisis de defectos, errores y fallos implica documentar cuidadosamente cada hallazgo identificado durante las pruebas. Esto incluye detalles como la severidad y la prioridad de cada problema, lo cual ayuda a priorizar las acciones correctivas y preventivas necesarias.

Un informe de defecto bien estructurado debe incluir:

a) Identificador único: Para facilitar el seguimiento y referencia.

b) Título descriptivo: Resumen conciso del problema.

c) Descripción detallada:

- Comportamiento esperado vs. comportamiento observado.
- Pasos para reproducir el problema.
- Entorno (versión del software, sistema operativo, etc.).

d) Severidad y prioridad: Indica el impacto y la urgencia de la resolución.

e) Estado: (e.g., Nuevo, En progreso, Resuelto, Cerrado).

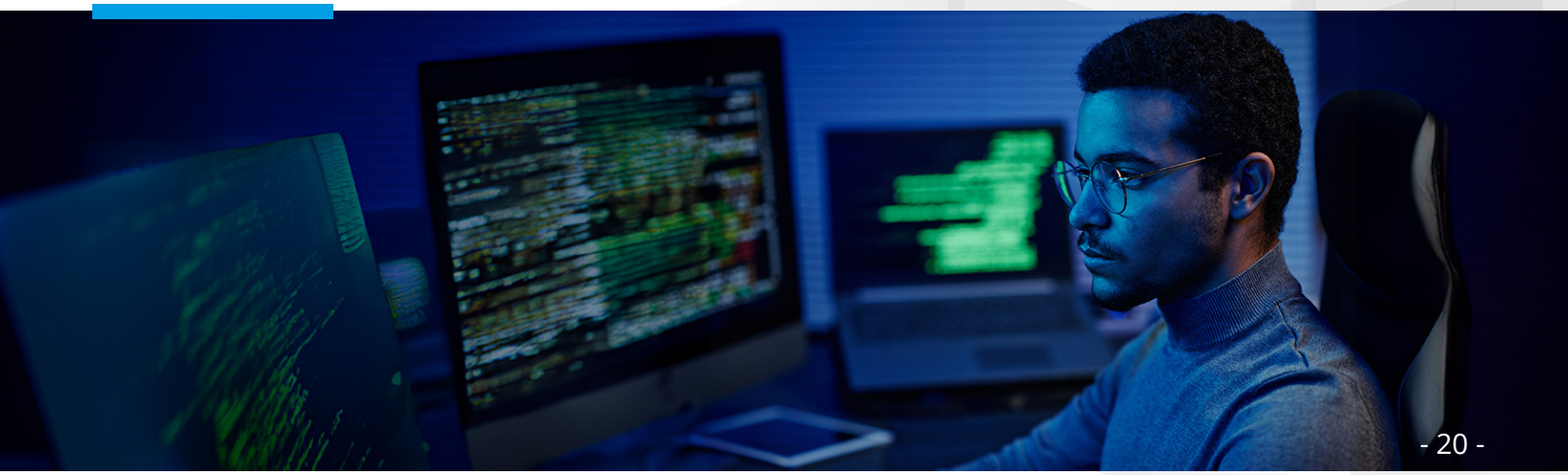
f) Asignado a: Responsable de la investigación o resolución.

g) Información adicional:

- Capturas de pantalla o videos del fallo.
- Logs relevantes.
- Datos de prueba utilizados.

h) Historial de cambios: Registro de actualizaciones y comentarios.

Una vez desarrollado el análisis, se plantean las soluciones tentativas considerando criterios de factibilidad, que abarcan aspectos como el impacto potencial del problema en el usuario final, el esfuerzo requerido para la corrección y la viabilidad dentro de los plazos establecidos por el plan de prueba.



Uso de imágenes o diagramas

Las imágenes o diagramas pueden ser útiles para ilustrar defectos, errores y fallos en el desarrollo de software y para representar gráficamente el ciclo de vida de un defecto. Se recomienda incluirlos para ayudar a clarificar procesos secuenciales, como en el ciclo de vida de un defecto o documentar casos de fallos durante la ejecución del software.

A modo de ejemplo, podemos mencionar:

1. Documentación de defectos:

- **Capturas de pantalla:** Útiles para mostrar el comportamiento erróneo de la interfaz de usuario.
- **Diagramas de flujo:** Para ilustrar la secuencia de pasos que llevan al fallo.

Pueden ser útiles durante el registro inicial del defecto y en la documentación detallada.

2. Análisis de causa raíz:

- **Diagramas de Ishikawa (espina de pescado):** Para visualizar las posibles causas de un problema.
- **Árboles de decisión:** Para mapear el proceso de toma de decisiones que lleva a un error.

Pueden ser útiles durante la fase de análisis del defecto, para comprender mejor su origen.

3. Planificación de soluciones:

- **Diagramas UML:** Para mostrar cómo se podría modificar la arquitectura del software.
- **Wireframes:** Para proponer cambios en la interfaz de usuario.

Pueden ser útiles al diseñar y proponer soluciones para el defecto.

4. Comunicación con el equipo:

- Infografías: Para resumir datos complejos sobre tendencias de defectos.
- Gráficos de Gantt: Para planificar y visualizar el proceso de corrección.

Pueden ser útiles en reuniones de equipo, informes de estado y planificación de sprints.

Cierre e ideas fuerza

Hasta aquí pusimos el foco en la distinción entre defectos, errores y fallos, su registro y documentación como parte del plan de pruebas de software. Hemos señalado la importancia de documentar cuidadosamente los hallazgos lo cual ayuda a priorizar las acciones correctivas y preventivas necesarias.

Ideas Fuerza

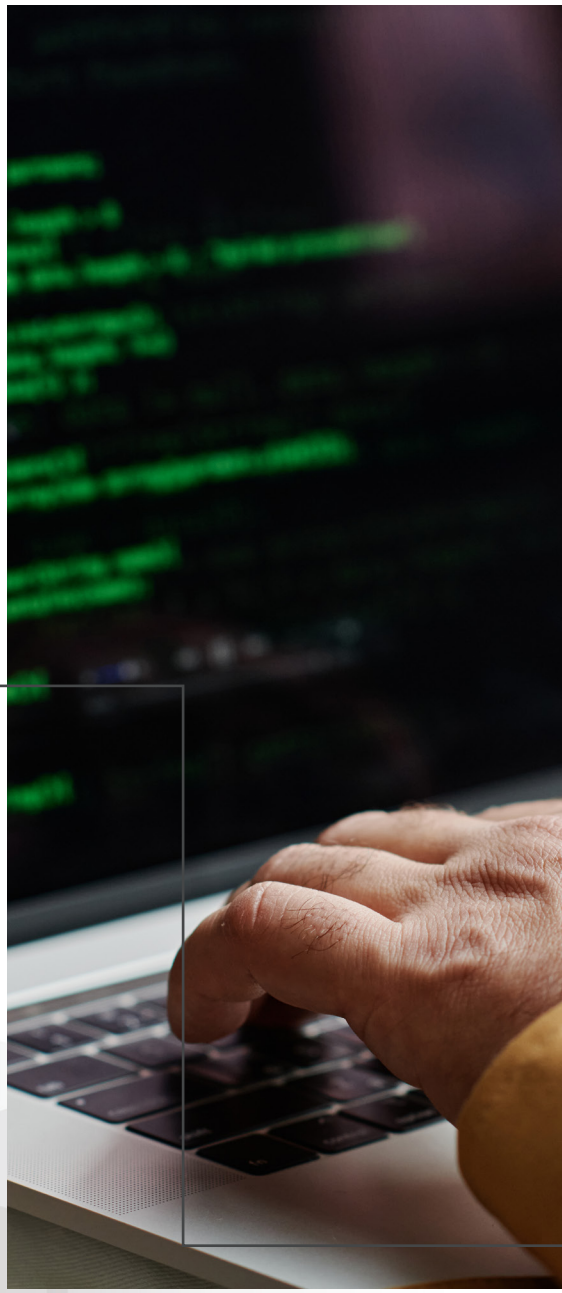
- La gestión efectiva de defectos en el desarrollo de software no solo asegura la calidad del producto final, sino que también optimiza el proceso de entrega y satisfacción del cliente.
- La identificación temprana, evaluación meticulosa y resolución eficiente de los defectos son pilares para mantener la integridad y funcionalidad del software, garantizando así su éxito en el mercado y la confianza de los usuarios finales.



5. Guía: Procesos de seguimiento en el monitoreo

Introducción

Llevar a cabo un seguimiento efectivo de fallos, errores y defectos durante el desarrollo de software es crucial para garantizar la calidad y la integridad del producto final. Este proceso no solo implica la identificación y documentación detallada de los problemas encontrados, sino también la implementación de medidas correctivas adecuadas. El monitoreo constante según el plan de pruebas establecido permite una gestión proactiva de los riesgos, facilitando así la toma de decisiones informadas para mantener el proyecto en línea con los estándares de calidad y los objetivos del negocio.



Seguimiento de fallos, errores y defectos

Llevar a cabo el seguimiento de fallos, errores y defectos durante el desarrollo de software implica un proceso estructurado y continuo para garantizar la calidad del producto final. Este proceso se realiza según el plan de pruebas establecido, que incluye varias etapas clave:

1. **Identificación de Fallos y Defectos:** Inicialmente, se identifican y registran todos los fallos y defectos encontrados durante las pruebas. Esto puede incluir problemas de funcionamiento, comportamientos inesperados, o incumplimientos de los requisitos especificados.
2. **Documentación Detallada:** Cada fallo o defecto identificado se documenta de manera detallada, incluyendo información relevante como la descripción del problema, el contexto en el que ocurrió, los pasos para reproducirlo, la severidad y la prioridad asignada para su corrección.
3. **Clasificación y Priorización:** Los fallos y defectos se clasifican según su gravedad y su impacto en el sistema. La clasificación ayuda a priorizar cuáles problemas deben ser resueltos primero, considerando factores como el impacto en la funcionalidad, la criticidad del componente afectado y los plazos de entrega del proyecto.
4. **Análisis de Causa Raíz:** Para abordar los fallos de manera efectiva, se realiza un análisis de causa raíz. Esto implica investigar las razones subyacentes que han llevado a la aparición del problema. Por ejemplo, un fallo podría deberse a un error en la lógica de programación, a una integración incorrecta entre módulos o a una interpretación incorrecta de los requisitos.
5. **Implementación de Soluciones:** Una vez identificada la causa raíz, se diseñan y aplican soluciones correctivas. Esto puede incluir modificaciones en el código, ajustes en la configuración del sistema, actualizaciones de documentación o cambios en los procedimientos de prueba.
6. **Pruebas de Verificación:** Después de implementar las soluciones, se realizan pruebas de verificación para asegurarse de que los fallos y defectos han sido corregidos correctamente y que no se han introducido nuevos problemas en el proceso.
7. **Seguimiento y Monitoreo:** Durante todo este proceso, se realiza un seguimiento continuo y monitoreo de los fallos corregidos y de cualquier nueva incidencia que pueda surgir. Esto ayuda a asegurar que el sistema se mantenga estable y funcional a lo largo del tiempo.

Un monitoreo efectivo se basa en métricas definidas y en herramientas adecuadas que faciliten la detección y el seguimiento de defectos.

Establecimiento de Métricas

Definir métricas claras es fundamental para el monitoreo eficaz. Algunas métricas útiles incluyen:

- **Tasa de defectos:** Número de defectos encontrados en relación con el tamaño del software.
- **Tiempo de resolución:** Tiempo promedio para resolver defectos.
- **Frecuencia de fallos:** Cuántas veces se repiten los mismos fallos en un período determinado.
- **Severidad de los Incidentes:** Distribución de los defectos según su severidad (crítico, alto, medio, bajo) para priorizar la atención.
- **Cumplimiento de Plazos:** Porcentaje de incidencias resueltas dentro de los plazos establecidos en el plan de acción.

Estas métricas proporcionan información valiosa sobre la calidad del proceso de seguimiento y permiten identificar áreas de mejora.

Herramientas de Monitoreo

Seleccionar las herramientas adecuadas colaborará con un seguimiento exitoso. Algunas opciones incluyen:

- **Sistemas de seguimiento de errores (bug trackers):** Permiten registrar y gestionar fallos y errores.
- **Herramientas de monitoreo de rendimiento:** Ayudan a identificar problemas de rendimiento en tiempo real.
- **Dashboards de métricas:** Visualizan información clave sobre el estado del software y los problemas detectados.

Ejemplo

Supongamos que durante las pruebas de un sistema de gestión de inventario se encuentra un fallo en el módulo de generación de informes, donde ciertos datos no se muestran correctamente en el reporte final. Tras identificar el fallo, se documenta detalladamente con capturas de pantalla y pasos para reproducirlo. Luego, se prioriza su corrección debido a su impacto en la precisión de los informes financieros del cliente. Tras un análisis, se descubre que el problema radica en un error de lógica en el cálculo de ciertos datos, y se implementa una solución que implica una actualización del algoritmo de generación de informes. Después de las pruebas de verificación, se confirma que el fallo ha sido corregido y se sigue monitoreando para asegurar que no vuelva a ocurrir.

Este enfoque metódico y estructurado en el seguimiento de fallos y defectos asegura que el software entregado cumpla con los estándares de calidad requeridos y satisfaga las expectativas del cliente.

Material complementario

Para profundizar en el contenido abordado, te proponemos consultar el siguiente material complementario:

Ever Curiel (2024, 29 de febrero). Explorando los Principios del Software Testing. [Podcast] La escuela del testing. https://www.laescueladeltesting.com/podcast/episode/23b19a8b/principios-fundamentales-del-software-testing-or-parte-2

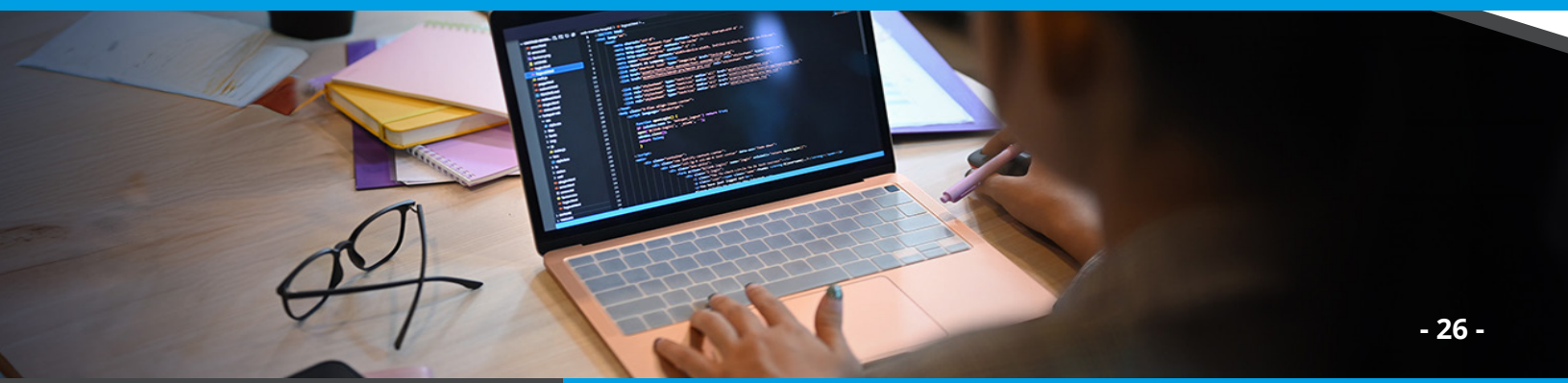
Fossati, Matías. (2016) Testing: Convertite en un experto probando software. Assurance Quality. https://www.google.cl/books/edition/Testing/_kQ1DgAAQBAJ?hl=en&gbpv=1

Cierre e ideas fuerza

Al reflexionar sobre los conceptos clave abordados en la gestión de defectos y pruebas de software, es fundamental reconocer cómo cada fase del ciclo de vida contribuye a la mejora continua del producto. La identificación temprana de defectos, la aplicación de técnicas adecuadas de testing y la gestión efectiva del cambio son pilares para asegurar la calidad del software. Este enfoque no solo optimiza los recursos y tiempos de desarrollo, sino que también fortalece la confianza en la entrega de productos robustos y funcionales.

Ideas Fuerza

- La identificación temprana de problemas es clave para evitar fallos críticos.
- Un registro detallado ayuda en la resolución y análisis de futuros problemas.
- Los datos recogidos informan sobre mejoras necesarias en el proceso de desarrollo.



Bibliografía

Ever Curiel (2024, 29 de febrero). Explorando los Principios del Software Testing. [Podcast] La escuela del testing. https://www.laescueladeltesting.com/podcast/episode/23b19a8b/principios-fundamentales-del-software-testing-or-parte-2

ISO/IEC/IEEE. (2013). ISO/IEC/IEEE 29119-1:2013 - Software and systems engineering -- Software testing -- Part 1: Concepts and definitions. International Organization for Standardization.

IEEE. (2008). IEEE Standard 829-2008 - IEEE Standard for Software and System Test Documentation. Institute of Electrical and Electronics Engineers

Fillottrani, P. R. (2007). Calidad en el Desarrollo de Software. Métricas de procesos de software. https://controldecalidadcue1.wordpress.com/wp-content/uploads/2011/04/clase5.pdf

Fossati, Matías. (2016) Testing: Convertite en un experto probando software. Assurance Quality. https://www.google.cl/books/edition/Testing/_kQ1DgAAQBAJ?hl=en&gbpv=1

Gonzalez, Marta (s/f) Testing Exploratorio: Una guía práctica. Marta González Desarrollo Front-End https://martagonzalez.dev/blog/testing-exploratorio-una-guia-integral/

Kaner, C, Bach, James & Pettichord, Bret. (2002). Lessons Learned in Software Testing: A Context-Driven Approach. Wiley.

Kaner, C. (2006, November 17). Exploratory Testing Explained. Cem Kaner, J.D., Ph.D. Keynote at QAI https://kaner.com/pdfs/ETatQAI.pdf

Kumar, Sandeep, and Santosh Singh Rathore. (2018) Software Fault Prediction : A Road Map. 1st ed. 2018. Singapore: Springer Singapore. Web. https://inacap.primo.exlibrisgroup.com/permalink/56INACAP_INST/di6hvk/cdi_askewsholts_vlebooks_9789811087158

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

Naik, K., & Tripathy, P. (2008). Software Testing and Quality Assurance: Theory and Practice. John Wiley & Sons.

Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach (8th ed.). McGraw-Hill Education.

Sommerville, I. (2015). Software Engineering (10th ed.). Pearson.

Tarlinder, A. (2016). Developer Testing: Building Quality into Software. Addison-Wesley Professional. Copeland, L. (2003). A Practitioner's Guide to Software Test Design. Artech House.

Whittaker, J. A. (2009). Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design. Addison-Wesley.