UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

Principios y prevención de defectos 1 y 2

04/Septiembre/2023

Hector Emmanuel Garcia Garcia

217585177

Michel Emanuel López Franco

Computación tolerante a fallas 2023B

Introducción

Para la actividad de principios y prevención de defectos debíamos investigar métodos para la prevención de defectos e investigar que es la clasificación de defectos ortogonales (ODC), aunque normalmente hago dos documentos, uno para cada parte en este caso decidí hacer uno solo para ambas partes pues creo que ambas partes de complementan muchísimo para poder llegar a una conclusión puesto que la ODC no solo se encarga de clasificar defectos sí que implementa diferentes métodos y técnicas para prevenirlos y detectarlos, para una vez detectados poder clasificarlos de una mejor forma para ayudar a encontrar una mejor solución.

¿Qué es Orthogonal Defect Classification (ODC)?

La clasificación de defectos ortogonales (ODC) convierte la información semántica en el flujo de defectos de software en una medición del proceso. Las ideas fueron desarrolladas a finales de los 80 y principios de los 90 por Ram Chillarege en IBM Research. Esto ha llevado al desarrollo de nuevos métodos analíticos utilizados para el desarrollo de software y el análisis de procesos de prueba. ODC es independiente del modelo de proceso, el lenguaje y el dominio. Aplicaciones de la ODC han sido reportados por varias empresas en una variedad de plataformas y procesos de desarrollo, que van desde la cascada, espiral, cerrada, y ágil procesos de desarrollo.

Actividad para la prevención de defectos

- **Inspección de Código**: Es una técnica formal de revisión de código cuyo objetivo principal es detectar e identificar anomalías en un producto de software. Consiste en el examen sistematizado por parte de pares del trabajo elaborado por una persona. Se buscan errores comunes, por lo cual se examina para detectar la presencia de errores más que simular la ejecución. Se utiliza una lista de errores a detectar, en la cual se incluyen los errores de programación clásica.
- **Inspección de Diseño:** Similar a la inspección de código, pero se realiza sobre los documentos de diseño.
- **Prueba Unitaria:** La prueba unitaria es una forma de verificar una unidad de software de forma aislada. Una unidad puede ser un método, una clase o un conjunto pequeño de clases que forman un módulo. Las pruebas unitarias son generalmente realizadas por los desarrolladores de la unidad.
- **Prueba de Integración:** Es aquella que se realiza una vez que se han aprobado las pruebas unitarias. El objetivo es verificar la comunicación entre dos o más componentes. Las pruebas de integración se focalizan en los defectos de las interfaces, y se utiliza en la integración de módulos, subsistemas o sistemas. Generalmente las pruebas de integración son realizadas por los desarrolladores de los componentes.

- **Prueba de Sistema:** Las pruebas de sistema tienen como objetivo verificar el sistema software en su totalidad, para comprobar si el mismo cumple con sus requerimientos. Abarca las pruebas funcionales, de usabilidad, de rendimiento, de seguridad, entre otras. Las pruebas pueden ser realizadas por los desarrolladores del sistema o por un conjunto de verificadores expertos en el área con gran conocimiento de la especificación del software a testear

Tipo de defecto

- **Asignación:** Un defecto de este tipo implica el cambio de unas pocas líneas de código, como la inicialización de un bloque de control o de estructuras de datos.
- **Chequeo**: Se refiere a lógica que ha fallado en la validación de datos, chequeo de valores antes de ser utilizados o condición de iteraciones.
- **Algoritmo**: Son defectos de eficiencia o correctitud que pueden ser corregidos mediante la reimplementación del algoritmo o mediante un cambio de la estructura de los datos locales, sin necesidad de un cambio en el diseño.
- **Función:** Un defecto de este tipo es aquel que afecta significativamente la capacidad, las características requeridas para el usuario final, la API, las interfaces con hardware o estructuras globales. Requiere un cambio formal en el diseño.
- **Timing/Serialization:** Son aquellos defectos que se corrigen mediante la mejora en el manejo de recursos compartidos y de tiempo real.
- **Interfaz:** Corresponde a defectos en la interacción con otros componentes, módulos, controladores de dispositivos, llamados a funciones, bloques de control o lista de parámetros.
- **Build/Package/Merge:** Se refiere a defectos que ocurren debido a defectos en bibliotecas, control de cambios o control de versión.
- **Documentación:** Son defectos que pueden afectar a las publicaciones o las notas de mantenimiento.

Calificador de defecto

- **Faltante:** El defecto es causado por algo que se omite. Por ejemplo, falta una sentencia de asignación.
- **Incorrecto:** El defecto es causado por algo realizado incorrectamente. Por ejemplo, el chequeo de datos realizado usa valores incorrectos.
- **Extraño:** El defecto se debe a algo que no es relevante o pertinente al documento o al código en sí. Por ejemplo, en el documento de diseño hay una sección que no es necesaria para el producto actual, por lo que debería ser quitada

Triggers

Hay tres clases de triggers, según las distintas actividades de verificación que son comúnmente empleadas en el desarrollo de software: Inspecciones, Pruebas Unitarias/Funcionales, Pruebas de Sistema.

Triggers en inspecciones:

- **Compatibilidad con versiones**: Está relacionado con la comprensión de cómo la versión actual del producto funciona con versiones anteriores.
- **Compatibilidad Lateral:** Está relacionado con cómo el producto debe trabajar con otros productos con la misma configuración de software.
- **Conformidad del diseño:** Está relacionado con la completitud del diseño del producto respecto a los requerimientos y objetivos del producto.
- **Concurrencia:** Está relacionado con la comprensión de las cuestiones de timing y serialización relacionadas a la implementación del producto. Por ejemplo, mecanismos de bloqueo, regiones compartidas y secciones críticas.
- **Semántica Operacional:** Relacionado a la comprensión del flujo de la lógica respecto a la implementación de un diseño.
- **Consistencia/Completitud de Documento:** Está relacionado con la completitud de un diseño y asegurar la consistencia entre las distintas partes del diseño propuesto y la implementación.
- **Situaciones Raras:** Relacionado con implementaciones inusuales, idiosincrasias o información específica de un dominio que no es común.

Trigger en Pruebas Unitarias o Funcionales:

- **Prueba de Cubrimiento:** Se refiere a la ejecución de una función a través de varias entradas para cubrir la mayor cantidad de casos posibles del dominio de parámetros. Es un trigger de prueba de caja negra.
- **Prueba Secuencial:** Estos son casos de pruebas que apuntan a ejecutar múltiples secciones del código con diferentes secuencias. También es de caja negra.
- **Prueba de Interacción:** Son pruebas que exploran interacciones más complicadas que generalmente no son cubiertas por secuencias simples.
- Prueba de Variación: Son pruebas que apuntan a ejecutar una ´única función usando múltiples entradas.
- **Cobertura de Camino:** Simple Son pruebas de caja blanca que apuntan a ejecutar los distintos caminos del código.
- **Cobertura de Combinación de Caminos:** Prueba de caja blanca que apunta a caminos de código más completos.

Trigger en Pruebas de Sistema:

- **Recuperación/Manejo de Excepciones:** El defecto surge debido a un mal manejo de excepciones o no se ha ejecutado un proceso de recuperación.

- **Arranque y Reinicio del Sistema:** Esto se refiere a un producto siendo inicializado o reiniciado. Estos procedimientos pueden estar muy ligado a aplicaciones, como la base de datos.
- **Volumen de trabajo:** Esto indica que el producto alcanzó la capacidad máxima de alguno de los recursos.
- **Configuración de Hardware y Software:** Estos triggers son aquellos que son generados por cambios en el entorno, ya sea en hardware o software.
- **Modo Normal:** Esta categoría significa que para capturar estos triggers, no tuvo que suceder nada inusual. El producto falla cuando se suponía que debía funcionar normalmente.

Conclusiones

La ODC es definitivamente una muy buena manera de clasificar errores es bastante amplia y completa la clasificación que utiliza por lo que no solo es buena para clasificar fallos si no para prevenirlos y resolverlos, además de que las estrategias que maneja para encontrar los errores no son realmente complicadas y son algo que cualquiera de nosotros puede hacer como el hecho de inspeccionar el código o el diseño, para encontrar errores o realizar pruebas a partes aisladas del código y diseño para verificar su correcto desempeño son aspectos que ya se contemplan dentro de la ODC y que son realmente útiles y fáciles de hacer, por eso decidí mantener este documento como uno solo pues aparte de la clasificación los métodos que utiliza para encontrarlos facilitan su solución porque dependiendo de donde se encontró el error y como se clasifica solucionarlo es realmente sencillo.

Bibliografía

- De código cuyo objetivo principal es detectar e identificar, I. de C. E. U. T. F. de R. (s/f). Clasificación Ortogonal de Defectos Atributo Valores. Www.uv.mx. Recuperado el 4 de septiembre de 2023, de https://www.uv.mx/personal/ermeneses/files/2017/08/Clasificacion-Ortogonal-de-Defectos.pdf
- Clasificación de defectos ortogonales. (s/f). Hmong.es; tok.wiki. Recuperado el 4 de septiembre de 2023, de https://hmong.es/wiki/Orthogonal_Defect_Classification