



**Arellano Granados Angel Mariano**

**218123444**

**Computación Tolerante a Fallas**

**D06 2023B**

# Principios de prevención de defectos

## Introducción

Al crear cualquier tipo de software por motivos humanos, de software o de hardware se pueden presentar varios errores o defectos en la ejecución de nuestro sistema, es imposible prevenir el cien por ciento de los problemas dentro de un programa, por ello existen varios metodologías para la prevención de defectos las cuales nos proporcionan estrategias y hábitos que podemos seguir para localizar, atacar y resolver los diferentes defectos que encontremos.

## Desarrollo

A continuación, se enumeran algunos métodos tradicionales y comunes que se han utilizado durante mucho tiempo para la prevención de defectos.

**Revisión e inspección:** Este método incluye la revisión por parte de un miembro individual del equipo (autocomprobación), revisiones por pares e inspección de todos los productos de trabajo.

**Tutorial:** Esto es más o menos como una revisión, pero se relaciona principalmente con comparar el sistema con el prototipo, lo que le dará una mejor idea sobre la corrección y / o la apariencia del sistema.

**Registro y documentación de defectos:** Este método proporciona información clave, argumentos / parámetros que se pueden utilizar para respaldar el análisis de defectos.

**Análisis de la causa raíz:** El análisis de la causa raíz incluye dos enfoques principales:

I) Análisis de Pareto:

El análisis de Pareto es una técnica formal y simple que ayuda a priorizar el orden de resolución del problema para lograr el máximo impacto. Afirma que el 80% del problema surge por razones del 20%. Por lo tanto, los problemas una vez identificados se priorizan de acuerdo con la frecuencia y se realiza un análisis estadístico detallado para encontrar qué 20% de las razones

atribuyen al 80% de los problemas. Simplemente centrándose en esas razones del 20% y eliminándolas, los resultados están garantizados mientras se optimiza la extensión del trabajo involucrado.

II) Análisis de espina de pescado:

También conocido como Análisis de Ishikawa este método es una técnica de análisis de causa raíz más visual. No hay estadísticas involucradas ya que este método se basa en una lluvia de ideas de todo el equipo.

**Nivel de TMM y manejo de defectos por organización de prueba**

TMM (Testing Maturity Model) se basa en CMM, es decir; Modelo de Capacidad de Madurez.

La Prevención de Defectos involucra a muchos miembros del personal y su esfuerzo colaborativo en varias etapas, razón por la cual juega un papel destacado en el nivel 5 de TMM. Si ocurre un defecto con frecuencia en cualquier caso de prueba o procedimiento, la organización puede asignar un grupo de miembros del personal para analizar el defecto y desarrollar el plan que contiene acciones para cambios en el proceso con el problema.

Algunos de los beneficios del programa de prevención de defectos son:

- El personal se motiva y es más consciente
- Satisfacción de los clientes
- Mayor confiabilidad, capacidad de administración y previsibilidad
- Mejora continua mejorada del proceso

**¿Qué es Orthogonal Defect Classification (ODC)?**

ODC es un esquema para capturar rápidamente la semántica de cada defecto de software. Es la definición y captura de los atributos de los defectos lo que hace posible el análisis y el modelado matemático. El análisis de los datos ODC proporciona un valioso método de diagnóstico para evaluar las distintas fases del ciclo de vida del software (diseño, desarrollo, prueba y

servicio) y la madurez del producto. ODC hace posible llevar la comprensión y el uso de los defectos mucho más allá de la calidad.

**ODC v5.2 para diseño y código:**

Durante el desarrollo de un software en curso, la información sobre defectos está disponible en dos momentos específicos. Cuando se abre un registro de defectos, normalmente se conocen las circunstancias que llevaron al descubrimiento del defecto y el probable impacto para el usuario. Tres atributos capturados en este momento son:

**Actividad ODC:** Revisión de diseño, Prueba unitaria, Prueba de función, etc.

**Activador ODC:** El entorno o la condición que provocó la exposición de los defectos, normalmente capturado en escenarios recreados.

**Impacto ODC:** El impacto potencial del defecto en el usuario previsto.

Cuando el registro de defectos se cierra después de aplicar la corrección, se conocen la naturaleza exacta del defecto y el alcance de la corrección. Cinco atributos capturados al momento del cierre son:

**Objetivo ODC:** representa el artefacto de alto nivel que se solucionó. Cuando hay un defecto en la implementación del código en relación con el Diseño o los Requisitos, los desarrolladores corrigen el código y, por lo tanto, el "Código" será el objetivo adecuado.

**Tipo de defecto ODC:** el alcance de la solución (asignación, verificación, algoritmo, etc.)

**Calificador ODC:** la solución requirió la adición de algún código faltante o la corrección del código incorrecto existente o la eliminación de código superfluo.

**ODC Source:** captura el origen del código que tenía el defecto (desarrollado internamente, reutilizado de una biblioteca, etc.)

**Antigüedad ODC:** Indica la antigüedad del código que tuvo el defecto en términos de su historial de desarrollo (Código base de una versión anterior, Código nuevo para la versión actual, etc.),

Estos ocho atributos de defecto, en conjunto, capturan la semántica de un defecto desde todas las perspectivas relevantes. El documento ODC para Diseño y Código brinda detalles relacionados con el esquema ODC original, los pasos reales involucrados en la clasificación de defectos y recomendaciones para la implementación del proceso de recolección de defectos.

## Conclusión

Existen muchas metodologías para la prevención de defectos que se especializan en diferentes tipos de software, entre ellos podemos encontrar la metodología ODC la cual es uno de los modelos más formales el cual su uso haciende más allá del software como en las investigaciones científicas.

Conocer estas metodologías nos ofrece una ventaja sobre los demás programadores, pues no importa nuestra experiencia o talento nunca podremos crear un programa exentó de defectos, por ello la mejor de las estrategias no es evitar fallar, sino resolver la mayor cantidad de defectos en el menor tiempo posible.

## Referencias


- Santhanam, P. (2023). Orthogonal defect classification (Archival) - IBM. IBM. https://researcher-watson-ibm-com.translate.goog/researcher/view_group_subpage.php?id=5020&_x_tr_sl=auto&_x_tr_tl=es&_x_tr_hl=es-419
- My server name. (n.d.). Defect Prevention Methods. myservername.com. https://spa.myservername.com/important-software-test-metrics#Defect_Prevention_Methods_and_Techniques