Clase 08/06 Anti-Patrones

Antipatrón

Forma de describir un tipo de solución comúnmente utilizada para resolver un problema que trae consigo resultados contraproducentes/negativos.

Blob

Una sola clase se encarga de la mayoría de las responsabilidades mientras que otras clases son simples contenedores de datos.

Cómo reconocerlo

- Una clase con un número extremadamente alto de atributos y métodos.
- Falta de cohesión en los atributos y operaciones de la clase.
- Las demás clases sirven principalmente para almacenar datos y tienen poca o ninguna lógica.

Consecuencias

- Dificulta mantenibilidad y comprensión del código.
- Reduce reusabilidad de clases.
- Puede afectar el rendimiento por la carga excesiva de una clase.

Evitar/Resolver

- **Refactorizar:** Dividir clases gigantes en clases más pequeñas y especializadas
- **Redistribuir responsabilidades (SRP)**
    - Asegurar que c/objeto tiene un propósito claro.

Lava Flow

Refiere a piezas de código que quedan en el sistema, aunque ya no son útiles, y se vuelven difíciles de eliminar

Características

- Código que se mantiene a pesar de que ya no es necesario

- Código que se ha vuelto obsoleto debido a cambios en el diseño o requisitos
- Código que se evita modificar por miedo a romper funcionalidades existentes

Formación

- Cambios rápidos en el proyecto
- Falta de documentación/comprensión del código.
- Temor a modificar por posibles problemas en el funcionamiento del sistema.

Consecuencias

- Aumento de complejidad del trabajo
- Infla tamaño de código
- Reduce la mantenibilidad

Evitar/Resolver

- Documentar bien el código y los cambios realizados
- Implementar revisiones de código periódicas
- Refactorizar el código obsoleto
- Eliminar código innecesario después de pruebas rigurosas y revisiones de código

Golden Hammer

Cuando se depende de una herramienta o tecnología familiar a expensas de mejores alternativas posibles.

Características

- Subreutilización de una herramienta o tecnología en particular
- Reluctancia para considerar o adoptar alternativas.
- Creencia de que la herramienta familiar puede resolver todos los problemas.

Cómo se forma

- Falta de conocimiento de otras herramientas y tecnologías
- Comodidad y familiaridad a una herramienta específica

- Resistencia al cambio o a aprender nuevas habilidades

Consecuencias

- Soluciones ineficientes o subóptimas
- Mayor costo y tiempo de desarrollo
- Limitación en innovación y adaptabilidad del proyecto

Evitar/Solucionar

- Educar al equipo sobre diferentes herramientas y tecnologías
- Realizar evaluaciones objetivas antes de seleccionar herramientas o tecnologías
- Fomentar un entorno abierto a experimentación y aprendizaje contínuo

Spaghetti Code

Definición

El código fuente de un programa tiene una estructra compleja y enredada, parecida a un plato de spaghetti, lo que lo hace dificil de leer, operar y mantener.

Características

- Falta de estructura y organización
- Uso exclusivo de saltos incondicionales (GOTOs)
- Difícil de seguir el flujo de control
- Ausencia de modularidad y encapsulación

Cómo se forma

- Desarrollo apresurado sin una arquitectura adecuada
- Falta de conocimiento de buenas prácticas de programación
- Modificaciones y parches sucesivos sin reconsiderar la estructura general

Consecuencias

- Dificultad en mantenimiento y depuración
- Incremento en el riesgo de introducir errores

- Mayor costo y tiempo de desarrollo

Evitar

- Planificar y diseñar la arquitectura antes de escribir en código
- Seguir buenas prácticas de programación
- Realizar revisiones de código

Resolver

- Dividir el código en funciones y clases más pequeñas y coherentes
- Reemplazar saltos incondicionales con estructuras de control más legibles
- Mejorar la documentación y comentarios.

Cut-and-Paste

Definición

Se refiere a la práctica de copiar y pegar bloques de código dentro de una aplicación en lugar de crear funciones o módulos reutilizables.

Características

- Duplicación de código
- Falta de modularidad y abstracción
- Dificultad en la implementación de cambios en lógicas duplicadas.

Como se forma

- Desarrollo apresurado o bajo presión de tiempo.
- Falta de conocimiento de buenas prácticas de programación.
- Tratar de evitar "reinventar la rueda" sin tener una adecuada reutilización de código.

Consecuencias

- Dificultad en el mantenimiento y la corrección de errores
- Incremento en el riesgo de consecuencias
- Código menos legible y comprensible

Evitar

- Pensar en modularidad y reutilización de código desde el inicio.
- Crear funciones y clases para lógicas comunes.

Solucionar

- Identificar y eliminar la duplicación..
- Refactor

Tester Driven Development

Se refiere a un anti-patrón en el que las pruebas de SW o informes de errores dirígen el desarrollo de SW; en lugar de las necesidades del usuario o los requerimiento de las funcionalidades. Este enfoque puede dar lugar a una baja calidad del código y retrasos en la entrega del SW.

Como se forma

- Las pruebas comienzan demasiado pronto
- Los requerimientos no están completos
- Los testers o desarrolladores son inexpertos
- La gestión del proyecto es deficiente