UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Resumen Pitfalls

Integrantes:
Achig Toapanta Steven Jossue
Asmal Haro, Kevin David
Cadena Diaz, Jeremy Joel
Caiza Tacuri, Alisson Lisbeth
Cajas Recalde, Karla Lizbeth
Campoverde Encarnacion, Carlos Danny

NRC: 3682
Fecha de entrega: 06/07/21

Pitfalls

Los pitfalls son errores comunes de los programadores que producen un software de baja calidad que pueden arruinar el proyecto.

Para cada pitfall, sería útil saber

- La descripción del problema: ¿Qué es?
- Consecuencias: ¿Qué daños o riesgos introduce en un proyecto?
- Causas: ¿Que conduce al escollo?
- Cómo evitarlo: ¿Cómo puede evitarlo un proyecto?
- Reconocimiento: ¿Cómo reconocer que un proyecto ha sucumbido a este escollo?
- Cómo salir de él: ¿Cómo puede un proyecto salir de este escollo?
- Cuando se documentan con este conocimiento, los pitfalls son antipatrones
- Algunos autores llaman a los pitfalls orientados a la implementación "Code Smells"

Lista de Pitfalls interesantes

- Nombres poco comunicativos

Los identificadores deben comunicar exactamente lo que el componente representa: su función, responsabilidades, intención y alcance.

- Nombres inconsistentes

El diseño no sigue ningún conjunto lógico o estándar de terminología o la implementación no sigue el diseño.

Evitación:

• Diseño y léxico cuidadosos

• Asegurarse de que una implementación se adhiera a un diseño

• Voluntad de refacto

Reconocimiento:

• Recorridos e inspecciones de diseño / código

• Considere cada nombre o etiqueta en el contexto de la intención del componente.

rol y alcance

Extrication:

• Refactor names

- Tipos incrustados en nombres

Descripción del problema:

• Los identificadores en la implementación incluyen información de tipo, p. Ej.

firstName String en lugar de solo firstName para una propiedad de nombre

Consecuencias:

• Escasa comprensibilidad y legibilidad à menor facilidad de mantenimiento y reutilización

• Puede romper la encapsulación, porque expone el tipo de implementación

Causas:

• Esfuerzos equivocados para proporcionar más información en el código.

- Métodos largos

Descripción del problema:

• Un método largo que carece de cohesión (un único propósito definitorio)

Consecuencias:

• Menor mantenibilidad

• Menor compresibilidad

Causas:

• Falta de atención al propósito definitorio de un método o a la cohesión de su

funcionalidad

• Evolución sin refactorización

- Código duplicado

Evitación:

• Refactorización según sea necesario cuando se agregan nuevas funciones o se realizan cambios al sistema

Reconocimiento:

• Recorridos e inspecciones de diseño / código

Extricación:

• Extraer superclase o extraer clase

• Método de extracción

15-20

- Explosión de clases :

Las subclases que tienen una funcionalidad similar a su clase principal pueden ocasionar lento mantenimiento, menos reusabilidad y flexibilidad, complejidad accidental, esto se debe al uso de la herencia cuando la agregación hubiera sido más óptima.

- Cadenas de mensajes largas :

Un objeto tiene que delegar una llamada de método a un objeto contenido, el cual a su vez tiene que delegar una llamada de método a un objeto que contiene, y así sucesivamente. Esto puede ocurrir cuando el patrón de decorador o una relación de composición es recursivo y mal utilizado esto causa mal rendimiento, complejidad accidental en el flujo de control durante el tiempo de ejecución

- Clases Grandes

El nombre de una clase es utilizado en muchos lugares y empieza a abarcar mucha carga, a causa de esto dicha clase se vuelve difícil de entender, mantener y reutilizar además que su localización es deficiente de su objetivo.

(Página 21 a 26)

- Evitar Errores

Al usarlo aumentamos la dificultad de entender, probar, mantener y utilizar. Sabemos que el método tiene varias complicaciones usando un decorador, método de patrones de estado y es un método de operación.

Solución Diferente:

Podemos similar a los duplicados de código. Pero el código se vuelve difícil de probar y reutilizar tiene una mala generalización de decisiones

- ❖ Evitación: congruencia de clase de objeto
- ❖ Reconocimiento: Tiene un código similar.
- ❖ Liberación:Es una extracción de clases, métodos o incluso una adaptación. Refactoriza métodos de plantillas

Comentarios redundantes o sin sentido:

Los comentarios en el código no agregan valor solo es una aclaración del tema. No debemos integrar comentarios inútiles.

- ❖ Consecuencias: El código es difícil de leer, al actualizar el código no se renueva el comentario y existe confusión.
- ❖ Causas: No tiene una buena comprensión.

Código Muerto.

No usamos variable de declaración. Por sus riesgos de seguridad

- ❖ Causas: Cambio a los requisitos, mejora el diseño para su calidad de implementación.

pag(27 a 32)

Descripción del problema:

· Una variable, Sentencia, Parámetro, Campo, Método o clase (ya no se utiliza)

Consecuencias:

· Menor compresibilidad, Menor capacidad de mantenimiento, Posibles riesgos de seguridad

Causas:

· Refactorización del código para mejorar la calidad, Mejoras en el diseño para mejorar su calidad, Cambio en los requisitos, provocan cambios en el diseño, y en el código

Reconocimiento:

· Mismo conocimiento.

Extracción:

· Eliminar el código muerto, Hacer comentarios sobre la decisión de diseño

- Generalidad Especulativa.

Descripción del problema:

· Programación basada en especulaciones sobre lo que podría ser necesario

Consecuencias:

· Menor compresibilidad, Menor capacidad de mantenimiento

Evitar:

· Dar prioridad a las características, Permitir el cambio en los lugares adecuados

- Campo Temporal

Descripción del problema:

· Los campos temporales obtienen sus valores sólo en determinadas situaciones.

Consecuencias:

· Menor extensibilidad, Menor compresibilidad, Menor capacidad de mantenimiento

preguntas:

1. Los pitfalls son ……... comunes de los programadores que producen un software de …….. calidad que pueden arruinar el proyecto.

   R:errores, baja

2. El diseño no sigue ningún conjunto lógico o estándar de terminología o la implementación no sigue el diseño es:

- ☐ nombres poco comunicativos
- ☑ ~~nombres inconsistentes~~
- ☐ tipos incrustados en los nombres

3. Al usarlo aumentamos la ……. de entender, probar, mantener y ….. Sabemos que el método tiene varias complicaciones usando un decorador
    a. R: dificultad, utilizar
4. Que no usamos en el código muerto:
    a. Variable de declaración.
    b. Comentarios.
    c. Nombres con inconsistencias.
5. La explosión de …….. se debe al uso de la …….... cuando la ……. hubiera sido más óptima.

    R: clases, herencia, agregación

6. Clases grandes se refiere a que el ……. de una …….. es utilizado en muchos lugares y empieza a ……... mucha carga

    R: nombre, clase, abarcar.

7. Realice dos descripciones de problemas del Código Muerto.
    a. Sentencia
    b. Parámetro
8. Describa el problema del Campo Temporal.

    Los campos temporales obtienen sus valores sólo en determinadas situaciones.

9.Seleccione los parámetros de reconocimiento del código duplicado

    a. Localización de decisiones de diseño

    b.Esfuerzos equivocados para proporcionar más información en el código.

    c.Recorridos e inspecciones de diseño / código

    d. Reconocimiento de errores en el código

10.Ponga un ejemplo de tipos incrustados en nombres

    ……...