Índice general

1 Tema 8

ESTE ESQUEMA ESTA BIEN Y AYUDA, PERO ESTE TEMA AL SER FULL DIBUJITOS VERLO EN LAS DIAPOSITIVAS ES MÁS DIVERTIDO

1.1 Introducción

Refactorización es una técnica de desarrollo de software que consiste en reestructurar el código fuente de un programa sin cambiar su comportamiento externo. El objetivo es mejorar la legibilidad, mantenibilidad y eficiencia del código. En general, tenemos que:

1.2 Por qué refactorizar

- **Mejora el diseño**
- **Facilita la comprensión**
- **Ayuda a encontrar errores**
- **Ayuda a programar más rápido**

1.3 Cuándo refactorizar

`Fowler` da los siguientes protips para refactorizar:

- La **tercera vez** que haces algo similar.
- Al **añadir funcionalidad** o **arreglar un bug**.
- En la **revisión de código**.

`Astels` dice otras tres intenciones relacionadas:

- **Eliminar duplicados**: Cuando se repite código en muchas partes surge la idea de abstraerlo. Evitar números mágicos. Si varias subclases tienen el mismo método, se puede subir a la superclase.
- **Código dudoso**. Elige nombres más descriptivos evitando redundancias. No me vengas con 'a' como variable :(
- **Sospechas** de un error en x zona. El código se dice que "huele" por que estas cometiendo errores anteriores, malas prácticas y quizas rompiendo ciertos patrones como GRASP.

1.4 Algunas refactorizaciones importantes

- Composición de métodos: Extraer métodos largos en otros más pequeños.
- Mover características entre clases: Si una clase tiene demasiadas responsabilidades, se pueden mover a otra clase. Esto también sirve para métodos/atributos.
- Organizar datos: Evitar números mágicos. Evitar tipos por subclases.
- Simplificar expresiones condicionales: Evitar anidaciones. Usar polimorfismo en vez de 40 else ifs con los distintos tipos.
- Simplificar llamadas a métodos: Minimizar el número de parámetros (preservar objeto completo), reemplazar constructores por métodos factoria. Hacer uso de interfaces.
- Generalización: Jugar con la herencia y la composición para evitar duplicidades.