Clase 13: Code Smells

Rodrigo Arturo Saffie Kattan

Pontificia Universidad Católica de Chile

rasaffie@ing.puc.cl

20 de septiembre de 2016

Contenidos

Repaso Clase Anterior

Existen 3 clasificaciones en GoF:

- Creacionales
- Estructurales
- De comportamiento

¿Qué es un Code Smell?

"A code smell is a surface indication that usually corresponds to a deeper problem in the system." [Fowler, M., 2006]

El término fue acuñado por Kent Beck, y se hizo famoso con el libro [Refactoring: Improving the Design of Existing Code, 2012].

No son bugs: el programa funciona correctamente, pero su débil diseño dificulta el desarrollo e incrementa la posibilidad de generar *bugs*.

¿Qué es un Code Smell?

Características

- Indicador superficial
- Rápido de detectar
- No siempre indican un problema en el código

Clasificación

Existen 5 clasificaciones para los *code smells*:

- Bloaters
- Object-Orientation Abusers
- Change Preventers
- Dispensables
- Couplers

Bloaters

- *Bloaters* representan código, métodos y clases que han crecido a tal punto que es difícil trabajar con ellos.

Long Method

Síntomas

- Un método contiene excesivas líneas de código. Si tiene más de 10 puede haber un problema

Razones del problema

- La única persona que lee el método es la persona que lo escribió
- "Es más fácil" agregar líneas a un método existente que crear nuevos

Beneficios

- Código más fácil de entender y mantener
- Es más fácil detectar código duplicado

Large Class

Síntomas

- Una clase contiene excesivas líneas de código, atributos y métodos

Razones del problema

- Las clases crecen en conjunto con la aplicación
- "Es más fácil" agregar líneas a una clase existente que crear nuevas

Beneficios

- Código más fácil de entender y mantener
- Es más fácil detectar código y funcionalidad duplicada

Primitive Obsession

Síntomas

- Uso de variables primitivas en vez de objetos pequeños
- Uso de constantes para guardar información

Razones del problema

- Se agregan atributos a una clase a medida que se necesitan, sin considerar que se pueden agrupar en un objeto
- Mala elección de tipos primitivos para representar una estructura de datos

Beneficios

- Código es más extensible
- Código más fácil de entender y organizar
- Es más fácil detectar código duplicado

Long Parameter List

Síntomas

- Más de 3 o 4 parámetros por método. Es difícil entender listas extensas de parámetros

Razones del problema

- Se agrupa funcionalidad en un sólo método
- Se pasan parámetros a un método directamente, y no a través de objetos que contienen la información

Beneficios

- Menos código y más fácil de entender
- Es más fácil detectar código duplicada

Data Clumps

Síntomas

- Diferentes partes de una aplicación contienen una definición recurrente de variables

Razones del problema

- Una pobre estructura de la aplicación, o *copy-and-paste programming*
- No se agrupan valores inseparables en un objeto

Beneficios

- Código más fácil de entender y organizar
- Menos código

Referencias

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (2012)
Refactoring: Improving the Design of Existing Code
1st ed., *Addison-Wesley Professional*

martinfowler.com
http://martinfowler.com/bliki/CodeSmell.html

https://refactoring.guru
https://refactoring.guru/smells/smells

Fin