Clase 15: Code Smells

Rodrigo Arturo Saffie Kattan

Pontificia Universidad Católica de Chile

rasaffie@ing.puc.cl

27 de septiembre de 2016

Contenidos

¿Qué es un Code Smell?

Características

- Indicador superficial
- Rápido de detectar
- No siempre indican un problema en el código
- No es un *bug per se*

Clasificación

Existen 5 clasificaciones para los *code smells*:

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

Object-Orientation Abusers

- Estos *code smells* se generan por una incompleta o incorrecta utilización de los principios OOP.

- **Switch Statements**
- **Temporary Field**
- **Refused Bequest**
- **Alternative Classes with Different Interfaces**

Change Preventers

- Estos *code smells* reflejan un posible problema si un cambio en una parte del código genera varios cambios en otras partes. A la larga, esto aumenta el costo y complejidad de desarrollar

Divergent Change

Síntomas

- Cambiar una clase resulta en cambiar varios métodos sin relación directa

Razones del problema

- Una pobre estructura de la aplicación, o *copy-and-paste programming*

Beneficios

- Mejora la organización del código
- Disminuye el código duplicado
- Simplifica la mantención del *software*

Shotgun Surgery

Síntomas

- Hacer cualquier modificación resulta en muchos cambios pequeños en distintas clases

Razones del problema

- Una responsabilidad especifica fue dividida entre varias clases

Beneficios

- Mejora la organización del código
- Disminuye el código duplicado
- Simplifica la mantención del *software*

Parallel Inheritance Hierarchies

Síntomas

- Agregar una subclase a una clase necesita que se agregue otra subclase a otra clase

Razones del problema

- Cuando las jerarquías son pequeñas las modificaciones necesarias se tienen bajo control. Sin embargo, a medida que la jerarquía crece se hace cada vez más difícil realizar cambios

Beneficios

- Mejora la organización del código
- Disminuye el código duplicado

Dispensables

- Representan fragmentos de código que son innecesarios e inútiles, cuya ausencia haría que el código fuese más claro, eficiente y fácil de entender

Comments

Síntomas

- Un método contiene excesivos comentarios

Razones del problema

- La intención de los comentarios es buena: mejorar la legibilidad de código complejo. Sin embargo, puede ser que la verdadera solución sea mejorar la estructura del código, más que explicarla

Beneficios

- El código es más intuitivo y claro

Duplicate Code

Síntomas

- Fragmentos de código son (casi) idénticos

Razones del problema

- Diferentes programadores trabajan por separado en funcionalidades relacionadas
- Puede existir duplicación implícita, donde códigos que no son similares tienen el mismo propósito
- Para aumentar la velocidad de desarrollo al corto plazo

Beneficios

- Simplifica la estructura del código
- Código más fácil de entender y mantener

Lazy Class

Síntomas

- Una clase no cumple un rol que realmente justifique su existencia

Razones del problema

- Cambios en la estructura del código pueden generar que una clase que era importante ya no lo sea
- Pueden ser clases de posibles funcionalidades que nunca se concretaron

Beneficios

- Disminuye la cantidad de código
- Simplifica la mantención de la aplicación

Data Class

Síntomas

- Una clase contiene solamente atributos, sin aplicar comportamiento sobre ellos

Razones del problema

- Cambios en la estructura del código pueden quitar los métodos de una clase para agruparlos bajo otra
- Clases que en un principio se diseñaron pensando que tendrían más responsabilidades

Beneficios

- Simplifica la estructura del código
- Ayuda a detectar código duplicado

Dead Code

Síntomas

- Una variable, parámetro, campo, método o clase que ya no se utiliza

Razones del problema

- Los requerimientos del *software* cambiaron sin limpiar el código antiguo
- Lógica de condicionales que es inaccesible

Beneficios

- Disminuye la cantidad de código
- Simplifica la mantención de la aplicación

Speculative Generality

Síntomas

- Existe una clase, método, campo o parámetro que no se utiliza

Razones del problema

- Se crea código para posibles funcionalidades que nunca se implementan
- Se generaliza lógica en el código por si alguna vez es necesario

Beneficios

- Disminuye la cantidad de código
- Simplifica la mantención de la aplicación

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