



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# **Clase 7**

# **Code Smells + Refactoring**

## **IIC2113 – Diseño Detallado de Software**

Rodrigo Saffie

rasaffie@uc.cl

19 de octubre de 2018

# *Code Smell*: Características

- **No son bugs:**

- el programa funciona correctamente, pero su débil diseño dificulta el desarrollo e incrementa la posibilidad de generar *bugs*.

- Es un indicador superficial rápido de detectar
- No siempre indican un problema en el código
- Se pueden generar por un mal diseño del programa, pero también por presiones en cumplir plazos al desarrollar

# *Code Smell*: Clasificaciones

Existen 5 clasificaciones:

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

# ¿Qué es *refactoring*?

- Es un proceso sistemático para mejorar código sin modificar su funcionalidad.
- El objetivo es reducir la deuda técnica, para así mejorar la mantenibilidad y facilitar la extensibilidad.
- Normalmente la presencia de un *code smell* motiva su uso para generar *clean code*.

# Actividad 6

- *Code Smells: Bloaters*
  - *Long Method*
  - *Large Class*
  - *Primitive Obsession*
  - *Long Parameter List*
  - *Data Clumps*
- *Refactor*

# Code Smells

## ***Object-Orientation Abusers***

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



# *Object-Orientation Abusers*

## *Switch Statements*

### **Síntomas**

- Se tiene un operador *switch* complejo, o una gran secuencia de *if*

### **Razones del problema**

- Mal uso de polimorfismo

### **Beneficios de solucionarlo**

- Código más organizado

# *Object-Orientation Abusers*

## *Temporary Field*

### **Síntomas**

- Se tienen atributos de una clase que se les asigna un valor bajo ciertas circunstancias. El resto del tiempo se encuentran vacíos.

### **Razones del problema**

- En vez de pasar datos como parámetros, se crean atributos para las clases

### **Beneficios de solucionarlo**

- Código más claro y organizado



# *Object-Orientation Abusers*

## *Refused Bequest*

### **Síntomas**

- Una subclase utiliza solamente algunas de las propiedades y métodos heredados de sus padres

### **Razones del problema**

- Se quiso reutilizar código entre una super-clase y una clase, pero son completamente distintas

### **Beneficios de solucionarlo**

- Código más claro y organizado

# *Object-Orientation Abusers*

## *Alternative Classes with Different Interfaces*

### **Síntomas**

- Dos clases realizan funcionalidades idénticas, pero con nombres distintos (clases y métodos)

### **Razones del problema**

- El programador que creo una clase probablemente no sabía que otra clase que ya existía tenía funcionalidad equivalente

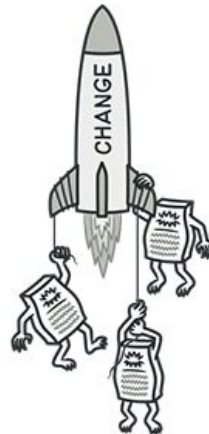
### **Beneficios de solucionarlo**

- Elimina código duplicado
- Código más claro y organizado

# Code Smells

## *Change Preventers*

- Estos *code smells* reflejan un posible problema si al realizar un cambio en una parte del código se generan varios cambios en otras partes. A la larga este comportamiento aumenta el costo y complejidad de desarrollar.



# *Change Preventers*

## ***Divergent Change***

### **Síntomas**

- Cambiar algo particular de una clase resulta en varios cambios a métodos sin relación directa

### **Razones del problema**

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

### **Beneficios de solucionarlo**

- Código más fácil de entender y organizar
- Menos código

# *Change Preventers*

## *Shotgun Surgery*

### **Síntomas**

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

### **Razones del problema**

- Una responsabilidad específica fue dividida entre varias clases

### **Beneficios de solucionarlo**

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

# *Change Preventers*

## *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 de solucionarlo**

- Mejora la organización del código
- Disminuye el código duplicado

### [Ejemplo](#)

# Code Smells

## *Dispensables*

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



# *Dispensables*

## ***Comments***

### **Síntomas**

- El código 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 de código más que explicarla.

### **Beneficios de solucionarlo**

- El código es más intuitivo y claro



# *Dispensables*

## ***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
- Aumentar la velocidad de desarrollo al corto plazo

### **Beneficios de solucionarlo**

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

# *Dispensables*

## *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 implementaron

### **Beneficios de solucionarlo**

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

# *Dispensables*

## ***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 de solucionarlo**

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

# *Dispensables*

## ***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 de solucionarlo**

- Disminuye la cantidad de código
- Simplifica la comprensión y mantenibilidad del código

# *Dispensables*

## *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 de solucionarlo**

- Disminuye la cantidad de código
- Simplifica la comprensión y mantenibilidad del código

# Code Smells

## Couplers

- Estos *code smells* aumentan el acoplamiento en el código.



# *Couplers*

## *Feature Envy*

### **Síntomas**

- Un método utiliza más información de otro objeto que en el que está definido

### **Razones del problema**

- Mala modelación
- Se crean clases para almacenar datos, pero no se mueven las operaciones que se basan en estos

### **Beneficios de solucionarlo**

- Menos código duplicado
- Código más organizado

# *Couplers*

## *Inappropriate Intimacy*

### **Síntomas**

- Una clase utiliza los métodos y atributos internos de otra clase

### **Razones del problema**

- Clases están muy ligadas entre sí

### **Beneficios de solucionarlo**

- Código más organizado
- Simplifica el mantenimiento del código



# *Couplers*

## ***Message Chains***

### **Síntomas**

- Excesivas llamadas encadenadas a métodos

### **Razones del problema**

- Un cliente depende de la estructura de navegación de las clases. Cualquier cambio de esta estructura requiere cambiar al cliente

### **Beneficios de solucionarlo**

- Reduce la dependencia entre clases

# *Couplers*

## *Middle Man*

### **Síntomas**

- Una clase solamente delega trabajo a otras, sin agregar funcionalidad

### **Razones del problema**

- Al reorganizar código, una clase puede quedar sin una responsabilidad propia

### **Beneficios de solucionarlo**

- Reduce acoplamiento
- Menos código

# *Couplers*

## *Incomplete Library Class*

### **Síntomas**

- Una librería/dependencia no satisface todas las necesidades del cliente

### **Razones del problema**

- La librería no provee las funcionalidades o no se tiene intención de desarrollarlas

### **Beneficios de solucionarlo**

- Disminuye el código duplicado

# Actividad 7



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# **Clase 7**

# **Code Smells + Refactoring**

## **IIC2113 – Diseño Detallado de Software**

Rodrigo Saffie

rasaffie@uc.cl

19 de octubre de 2018