Resumen Code Smells

IIC2113 2022-1

Profesor Antonio Ossa
Por Daniela Poblete

En programación, un code smell es cualquier característica en el código fuente de un programa que posiblemente indique un problema más profundo.

Estos se dividen en 5 categorías:

- **Bloaters:** Código, métodos y clases que han aumentado a proporciones tan gigantescas que son difíciles de trabajar.

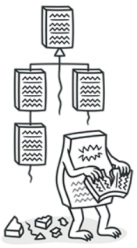
- **Object-Orientation Abusers:** Aplicación incompleta o incorrecta de los principios de la programación orientada a objetos.

- **Change Preventers:** Si necesitas cambiar algo en un lugar de tu código, tienes que hacer muchos cambios en otros lugares también.

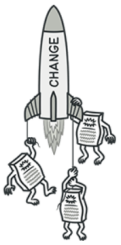
- **Dispensables:** Es algo inútil e innecesario cuya ausencia haría el código más limpio, eficiente y fácil de entender.

- **Couplers:** Acoplamiento excesivo entre clases o muestran lo que ocurre si el acoplamiento se sustituye por una delegación excesiva.

Bloaters



Object-Orientation Abusers



Change Preventers

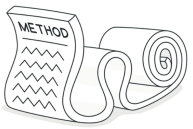
Dispensables

Couplers

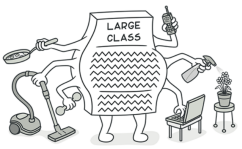
A continuación se explica los code smells dentro de cada categoría:

Bloaters

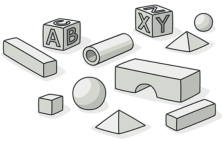
- **Long Method:** Un método contiene demasiadas líneas de código. Por lo general, cualquier método de más de diez líneas debería hacer que empieces a hacerte preguntas.



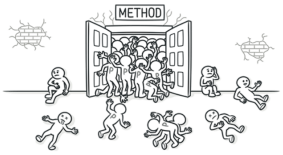
- **Large Class:** Una clase contiene muchos campos/métodos/líneas de código.



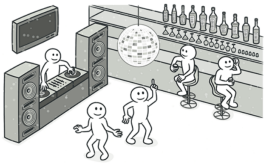
- **Primitive Obsession:** Uso de primitivas en lugar de pequeños objetos para tareas sencillas. Uso de constantes para codificar información. Utilización de constantes de cadena como nombres de campo para su uso en matrices de datos.



- **Long Parameter List:** Más de tres o cuatro parámetros para un método.

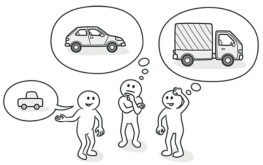


- **Data Clumps:** A veces, diferentes partes del código contienen grupos idénticos de variables. Estos grupos deben convertirse en sus propias clases.

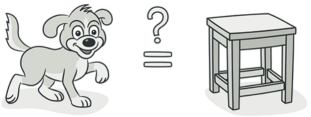


Object-Orientation Abusers

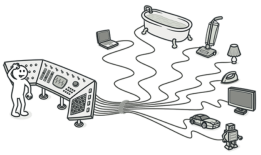
- **Alternative Classes with Different Interfaces:** Dos clases realizan funciones idénticas pero tienen diferentes nombres de métodos.



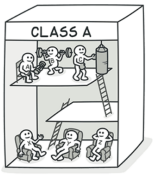
- **Refused Bequest:** Si una subclase utiliza sólo algunos de los métodos y propiedades heredados de sus padres, la jerarquía se desvía. Los métodos innecesarios pueden simplemente no utilizarse o redefinirse y dar lugar a excepciones.



- **Switch Statements:** Tiene un operador de conmutación complejo o una secuencia de sentencias if.

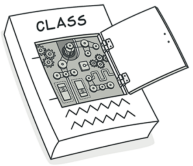


- **Temporary Field:** Los campos temporales obtienen sus valores (y, por tanto, son necesarios para los objetos) sólo en determinadas circunstancias. Fuera de estas circunstancias, están vacíos.

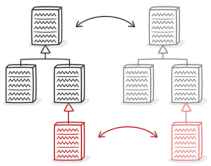


Change Preventers

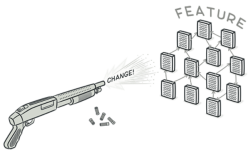
- **Divergent Change:** Te encuentras con que tienes que cambiar muchos métodos no relacionados cuando haces cambios en una clase.



- **Parallel Inheritance Hierarchies:** Siempre que creas una subclase para una clase, te encuentras con la necesidad de crear una subclase para otra clase.

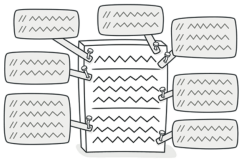


- **Shotgun Surgery:** Hacer cualquier modificación requiere que se hagan muchos pequeños cambios en muchas clases diferentes.

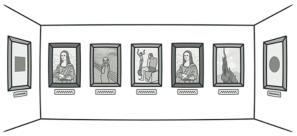


Dispensables

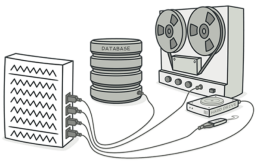
- **Comments:** Un método está lleno de comentarios explicativos.



- **Duplicate Code:** Dos fragmentos de código parecen ser casi idénticos.



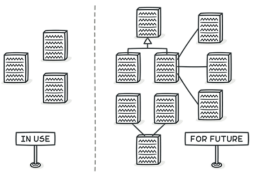
- **Data Class:** Se refiere a una clase que sólo contiene campos y métodos burdos para acceder a ellos (getters y setters). Son simplemente contenedores de datos utilizados por otras clases. Estas clases no contienen ninguna funcionalidad adicional y no pueden operar de forma independiente sobre los datos que poseen.



- **Dead Code:** Una variable, un parámetro, un campo, un método o una clase ya no se utilizan (normalmente porque están obsoletos).

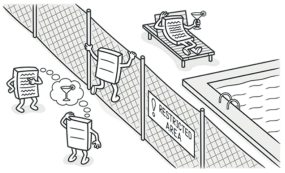
- **Lazy Class:** Entender y mantener las clases siempre cuesta tiempo y dinero. Así que si una clase no hace lo suficiente para ganarse su atención, debería ser eliminada.

- **Speculative Generality:** Hay una clase, un método, un campo o un parámetro no utilizados.



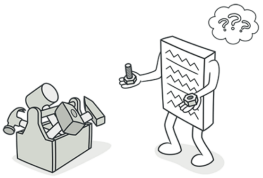
Couplers

- **Feature Envy:** Un método accede a los datos de otro objeto más que a sus propios datos.

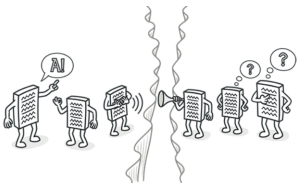


- **Inappropriate Intimacy:** Una clase utiliza los campos y métodos internos de otra clase.

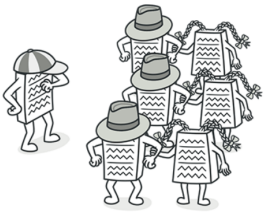
- **Incomplete Library Class:** Tarde o temprano, las bibliotecas dejan de satisfacer las necesidades de los usuarios. La única solución al problema -cambiar la biblioteca- suele ser imposible, ya que la biblioteca es de sólo lectura.



- **Message Chains:** En el código se ve una serie de llamadas parecidas a $a->b()->c()->d().



- **Middle Man:** Si una clase sólo realiza una acción, delegar el trabajo a otra clase, ¿Por qué existe?



Bibliografía

- Code smells. (s/f). Refactoring.Guru. Recuperado de https://refactoring.guru/es/refactoring/smells