¿Cómo huele nuestro código?

Code smells

GRUPO_ESFERA

Olores del código
(*Code Smells*)

No son bugs

Indican problemas más profundos

Los “olemos” con la práctica y experiencia

GRUPO_ESFERA

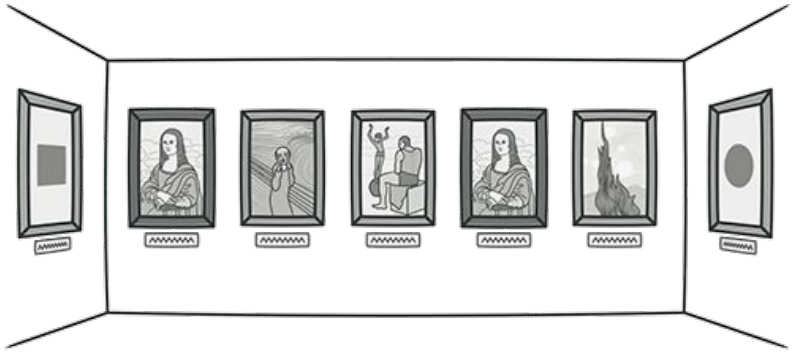
Algunos olores del código

La mugre evidente

Código duplicado
(*Duplicate Code*)

Identificación

Hay fragmentos del código que son idénticos (o casi idénticos)



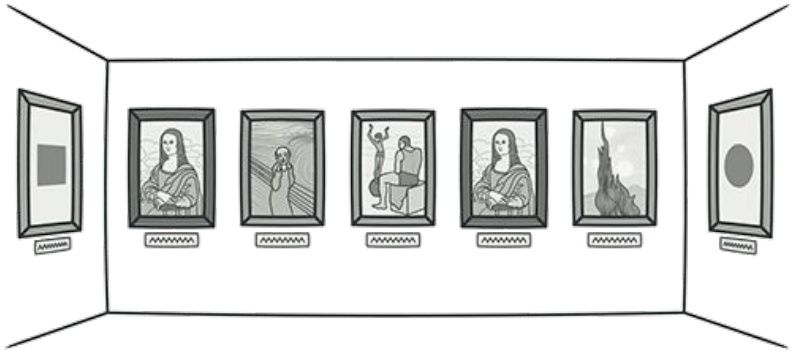
Código duplicado
(*Duplicate Code*)

Algunos tratamientos usuales

Extraer método

Extraer a una clase padre

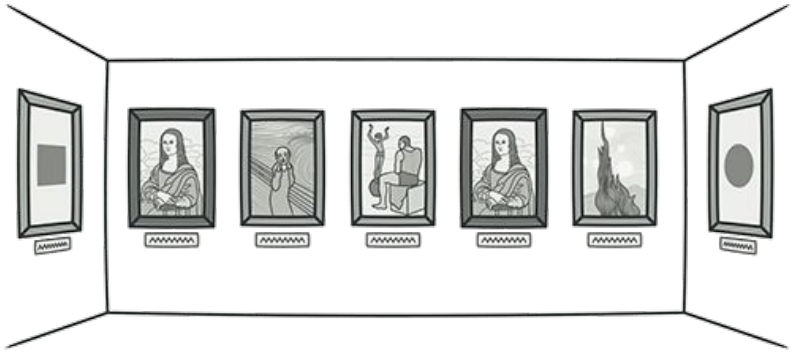
Consolidar fragmentos de código



Código duplicado
(*Duplicate Code*)

Razones para ignorarlo

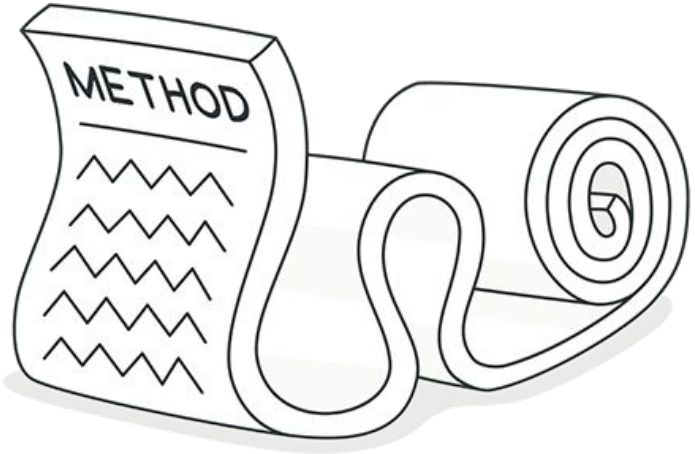
El código se vuelve menos intuitivo y obvio si intento arreglar este code smell



Método largo
(*Long Method*)

Identificación

Un método contiene muchas líneas de código (con más de 10 líneas ya deberíamos empezar a hacernos preguntas)



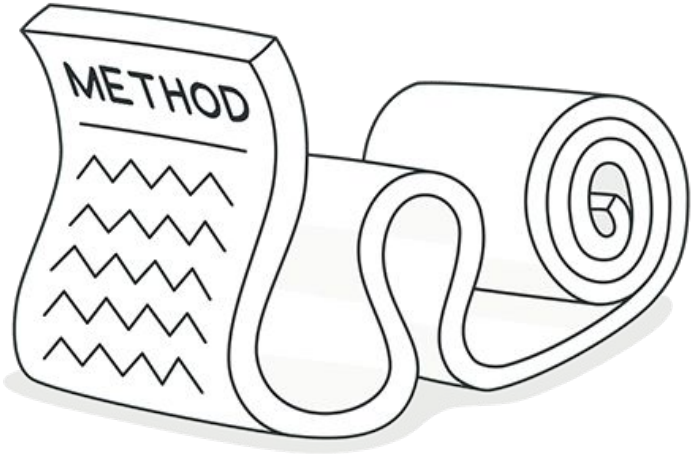
Método largo
(*Long Method*)

Algunos tratamientos usuales

Extraer método

Reemplazar método con una clase

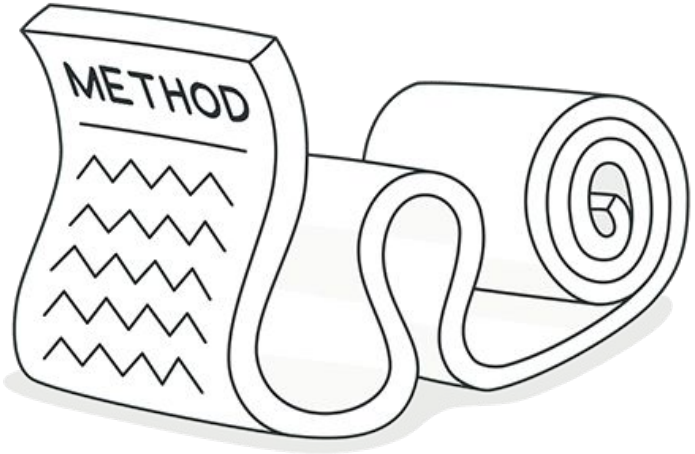
Llevar condicionales complejos a métodos



Método largo
(*Long Method*)

Razones para ignorarlo

Rendimiento de nuestra aplicación (aunque no suele suceder que ese rendimiento sea algo tan relevante…)



Código muerto
(*Dead Code*)

Identificación

Una variable, parámetro, campo, método o clase que ya no se usa (usualmente porque quedó obsoleta)

Código muerto
(*Dead Code*)

Algunos tratamientos usuales

Borrar código o archivos innecesarios

Absorber clase

Colapsar jerarquía

Quitar parámetros sin uso

Código muerto
(*Dead Code*)

Razones para ignorarlo

Armar un ejército de zombies 🧟

Ejemplo en el código

GRUPO_ESFERA

Algunos olores del código

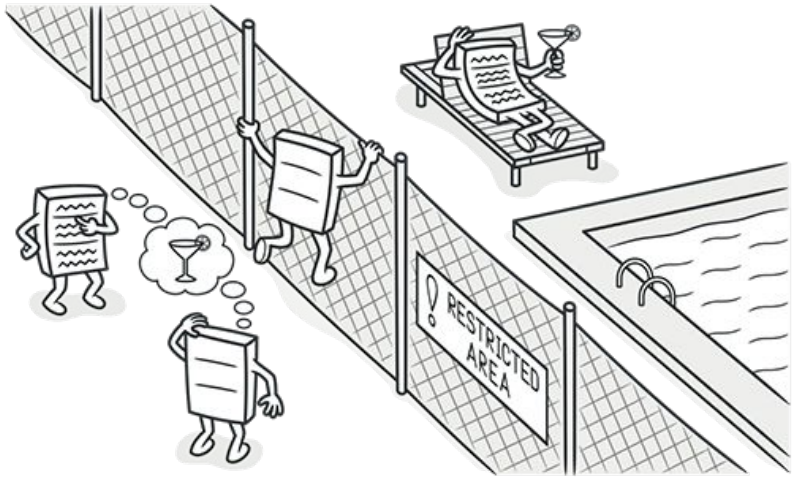
Los cimientos raros

Envidia de funcionalidad

(*Feature Envy*)

Identificación

Un método accede a los datos de otro objeto más que a sus propios datos



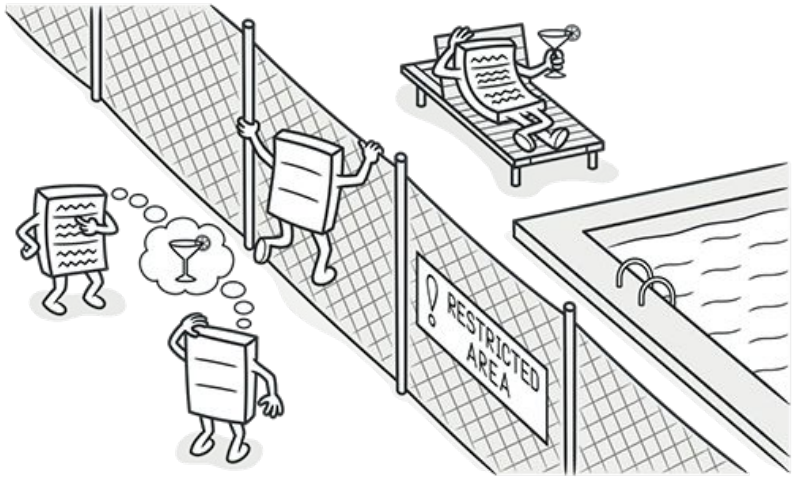
Envidia de funcionalidad

(*Feature Envy*)

Algunos tratamientos usuales

Mover método

Extraer método
(cuando solo una parte es la que accede)



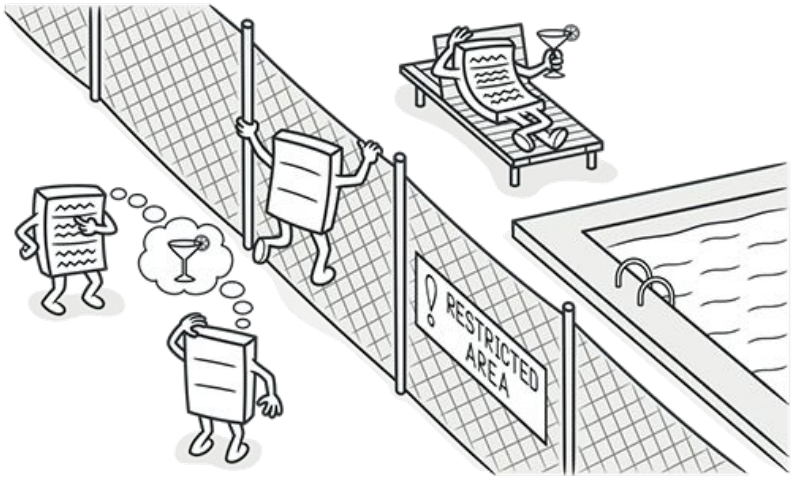
Envidia de funcionalidad

(*Feature Envy*)

Razones para ignorarlo

A veces se separa el comportamiento de la clase que contiene los datos a propósito

La ventaja principal es tener la habilidad de modificar el comportamiento dinámicamente (patrones de Strategy, Visitor)

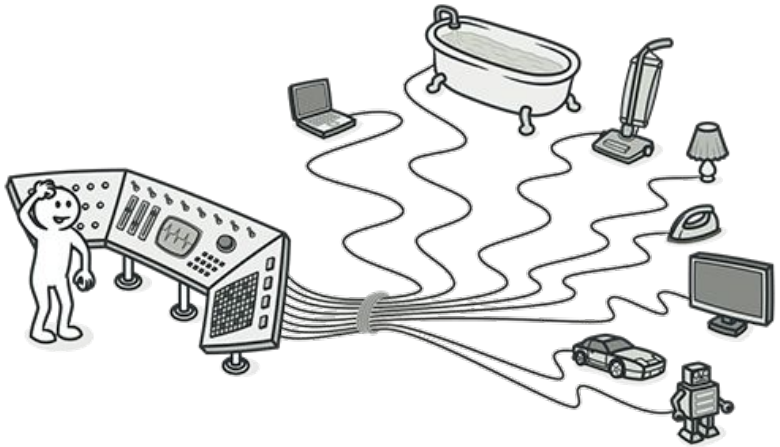


Abuso de condicionales

(*Switch Statements*)

Identificación

Hay un operador switch complejo o una secuencia de declaraciones if



Abuso de condicionales

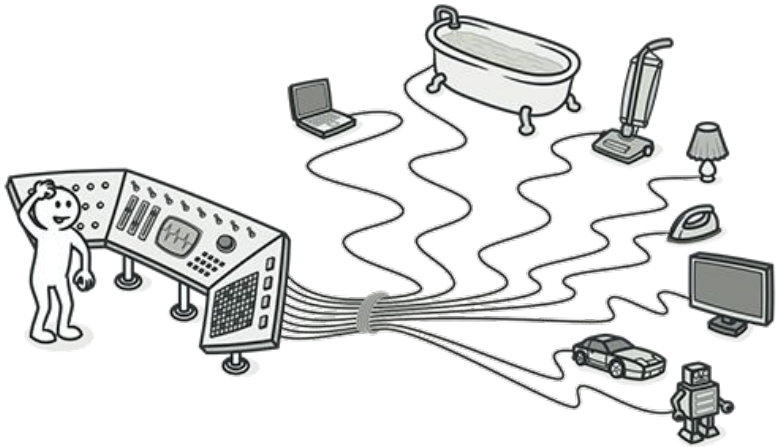
(*Switch Statements*)

Algunos tratamientos usuales

Reemplazar condicional con polimorfismo

Reemplazar código de tipo por subclases u objeto de estado

Reemplazar parámetro por métodos específicos



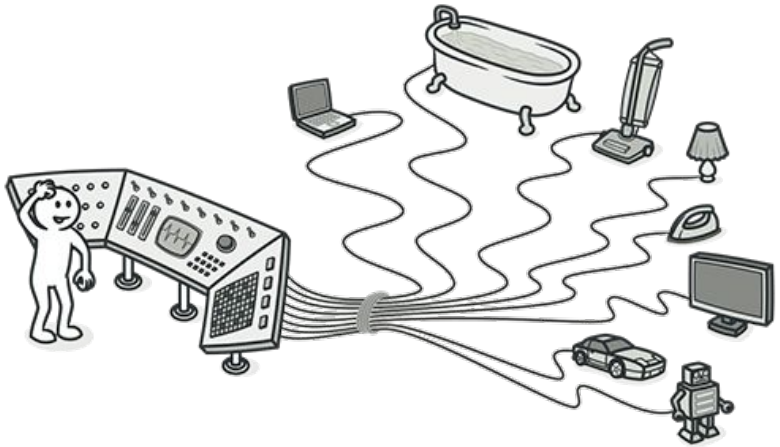
Abuso de condicionales

(*Switch Statements*)

Razones para ignorarlo

Cuando el operador switch ejecuta acciones simples

Uso de switch para el patrón de diseño Factory



Ejemplo en el código

GRUPO_ESFERA

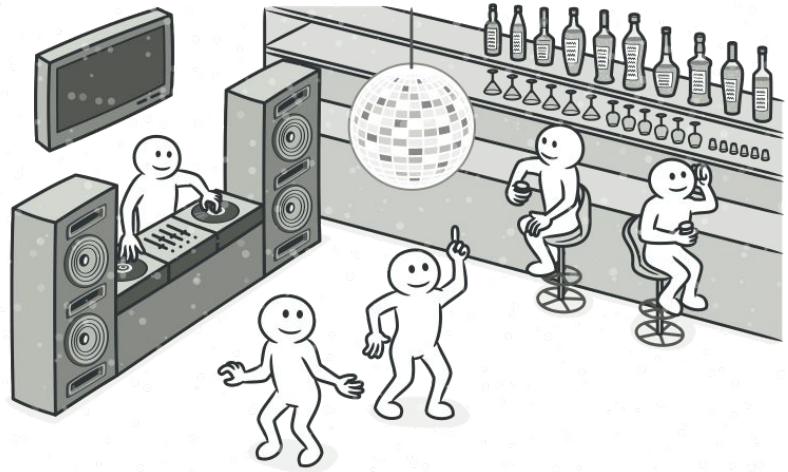
Algunos olores del código

Los que duelen en el mantenimiento

Grupos de datos
(*Data Clumps*)

Identificación

Tenemos grupos de datos que se repiten en distintas partes del código (por ejemplo parámetros de conexión a una base de datos)



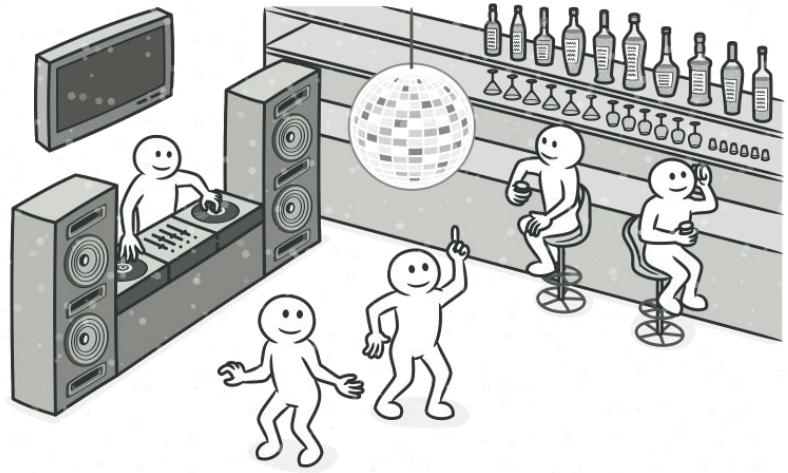
Grupos de datos
(*Data Clumps*)

Algunos tratamientos usuales

Extraer clase

Objetos como parámetros

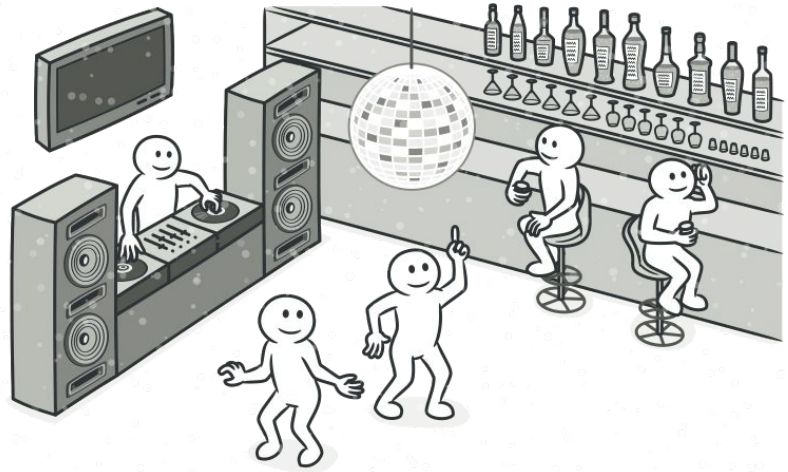
Preservar el objeto entero



Grupos de datos
(*Data Clumps*)

Razones para ignorarlo

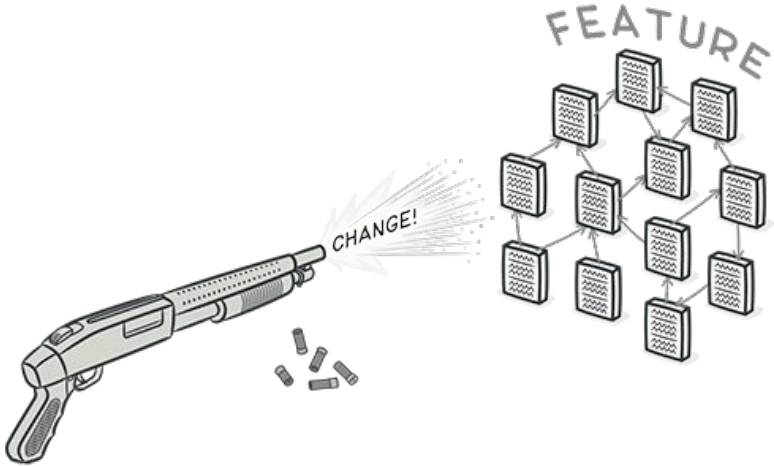
Cuando se genera una dependencia no deseada entre dos clases por pasar un objeto entero en lugar de sus valores separados como parámetros



Cirugía de escopeta
(*Shotgun Surgery*)

Identificación

Hacer un pequeño cambio te obliga a tocar muchas clases distintas

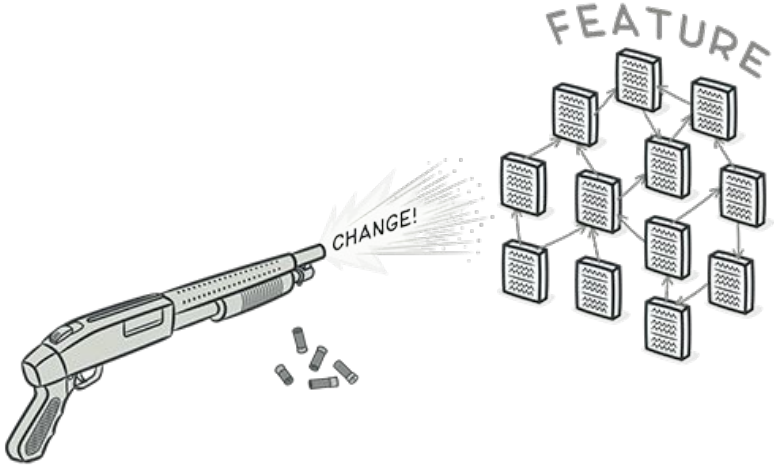


Cirugía de escopeta
(*Shotgun Surgery*)

Algunos tratamientos usuales

Mover métodos y campos a una única clase

Absorber clase



Ejemplo en el código

Categorías

Distintos tipos de code smells

Bloaters

Object-orientation abusers

Change preventers

Dispensables

Couplers

Inflados
(*Bloaters*)

Método largo
(*Long Method*)

Clase grande
(*Large Class*)

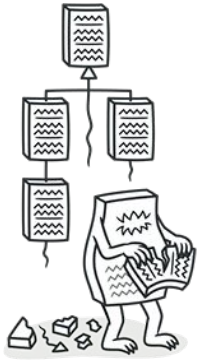
Obsesión primitiva
(*Primitive Obsession*)

Lista de parámetros larga
(*Long Parameter List*)

Grupos de datos
(*Data clumps*)

Abuso de la orientación a objetos

(*Object-Orientation Abusers*)



Clases alternativas con distintas interfaces

(*Alternative Classes with Different Interfaces*)

Herencia incómoda

(*Refused Bequest*)

Abuso de condicionales

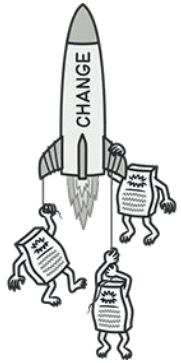
(*Switch Statements*)

Campo temporal

(*Temporary Field*)

Frenos al cambio

(*Change Preventers*)



Cambio divergente

(*Divergent Change*)

Cirugía de escopeta

(*Shotgun Surgery*)

Jerarquías paralelas de herencia

(*Parallel Inheritance Hierarchies*)

Innecesarios
(*Dispensables*)

Comentarios
(*Comments*)

Clase de datos
(*Data Class*)

Código duplicado
(*Duplicate Code*)

Clase perezosa
(*Lazy Class*)

Código muerto
(*Dead Code*)

Generalidad especulativa
(*Speculative Generality*)

Agrupadores
(*Couplers*)

Envidia de funcionalidad
(*Feature Envy*)

Dependencia incompleta
(*Incomplete Library Class*)

Intermediario
(*Middle Man*)

Intimidad inapropiada
(*Inappropriate Intimacy*)

Cadenas de mensaje
(*Message Chains*)

¡Gracias!

Más info en https://refactoring.guru/