Malos olores del código

El siguiente texto es un resumen del capítulo 3 del libro *Refactoring: Improving the Design of Existing Code* y del documento de Criterios y heurísticas de diseño de la cátedra de OO1. Los refactorings mencionados se encuentran aquí.

- **Aclaración**: El libro denomina *campo* a cualquier nombre de una declaración, variable o clase, y *módulo* a cualquier estructura de datos (función, clase)

Nombre Misterioso

▼ **Explicación**

- Ocurre cuando no nos queda claro para qué sirve una variable o que hace una función

▼ **Solución**

1. Si es una función → **Change Function Declaration**
2. Si es una variable → **Rename Variable**
3. Si es el nombre de una clase o un método → **Rename Field**

Código Repetido

▼ **Explicación**

- Ocurre cuando ves el mismo fragmento de código en varios lugares

▼ **Solución**

1. Si un fragmento de código se repite en varios métodos → **Extract Function**
2. Si un fragmento de código no es igual pero se parece a otro → **Slide Statements**
3. Si un método se repite en clases hermanas → **Pull Up Method**

Método Largo

▼ **Solución**

1. Si se pueden extraer partes de la función que tienen sentido → **Extract Function**

Lista Larga de Parámetros

▼ **Solución**

1. Si se puede obtener un parámetro de otro → **Replace Parameter with Query**
2. Si varios parámetros corresponden a un objeto → **Preserve Whole Object**
3. Si varios parámetros siempre se pasan juntos → **Introduce Parameter Object**
4. Si un parámetro se usa para controlar lo que ocurra → **Remove Flag Argument**

Datos Globales

▼ **Explicación**

- Ocurre cuando hay variables que pueden ser accedidas y modificadas desde cualquier lugar

▼ **Solución**

1. Aplicar encapsulamiento → **Encapsulate Variable**

Datos Alterables (Mutable Data)

▼ **Explicación**

- Ocurre cuando una variable se usa por razones diferentes o su acceso no está controlado

▼ **Solución**

1. Para controlar el acceso → **Encapsulate Variable**
2. Si una variable se utiliza para guardar cosas diferentes → **Split Variable**

3. Si una variable se puede calcular en otro lugar → **Replace Derived Variable with Query**

Cambio Disonante (Divergent Change)

▼ **Explicación**

- Ocurre cuando un módulo es cambiado de formas diferentes por razones diferentes

▼ **Solución**

1. Si un fragmento de código está realizando dos cosas a la vez → **Split Phase**
2. Si hay mas comunicación entre las llamadas → Crear módulos apropiados y **Move Function** para dividir el proceso
3. Si dos funciones mezclan dos tipos de procesamiento dentro de sí → **Extract Function**
4. Si son clases → *Extract Class*

Faltan aclaraciones

Arreglo Abarcativo (Shotgun Surgery)

▼ **Explicación**

- Ocurre cuando una modificación en un lugar hace que tengas que cambiar pequeñitas cosas en varios lugares

▼ **Solución**

1. Para reunir todos los cambio en un solo módulo → **Move Function**, **Move Field**
2. Si hay un monton de métodos operando sobre datos parecidos → **Combine Functions into Class**
3. Si hay métodos que transforman una estructura de datos → **Combine Functions into Transform**
4. Si los métodos en común pueden combinar sus resultados lógicamente → **Split Phase**

Envidia de Atributos

▼ **Explicación**

- Ocurre cuando un método de un objeto pasa mas tiempo comunicándose con métodos o datos de otros objetos en vez de repartir la tarea y delegarla

▼ **Solución**

1. Si un método interactúa mucho con una variable/método → **Move Function**
2. Si es solo una parte del método la que produce la envidia → **Extract Function**
3. Como heurística, poner juntas las cosas que cambian juntas

Acumulaciones de Datos (Data Clumps)

▼ **Explicación**

- Ocurre cuando varios datos/campos suelen ser usados en los mismos lugares

▼ **Solución**

1. Se podría crear un nuevo objeto que tenga los datos → **Extract Class**
2. Si varios parámetros frecuentemente o siempre van juntos → **Introduce Parameter Object**, **Preserve Whole Object**

Obsesión por los Primitivos

▼ **Explicación**

- Ocurre cuando utilizamos un tipo primitivo para un objeto que en realidad necesita un tipo de dato mas sofisticado

▼ **Solución**

1. Simplemente **Replace Primitive with Object**
2. Si el primitivo se utiliza para controlar un switch statement o algo por el estilo → **Replace Type Code with Subclasses, Replace Conditional with Polymorphism**
3. Si los primitivos aparecen de a grupo → **Extract Class**, **Introduce Parameter Object**

Switch Statements

▼ **Explicación**

- Ocurre cuando utilizamos una estructura de control por los msmos motivos en varios lugares. Quizás sea bueno aprovechar el polimorfismo

▼ **Solución**

1. **Replace Conditional with Polymorphism**

Reinventando la Rueda (Loops)

▼ **Explicación**

- Ocurre cuando programo comportamiento que sospecho que ya está programado

▼ **Solución**

1. **Replace Loop with Pipeline**

Elemento Ocioso

▼ **Explicación**

Ocurre cuando un módulo que creamos se utiliza poco

▼ **Solución**

1. Si un método se utiliza poco y en uno o dos lugares → **Inline Function**
2. Si pasa lo mismo con una clase → **Inline Class**
3. Si una clase/interfaz de la jerarquía se utiliza poco o nada → **Collapse Hierarchy**

Generalidad Especulativa

▼ **Explicación**

- Ocurre cuando controlamos casos especiales en nuestro código, o agregamos clases en la jerarquía tratando de adelantarnos a lo que pudiera ocurrir en el futuro

▼ **Solución**

1. Si hay clases abstractas que no hacen mucho → **Collapse Hierarchy**
2. Si hay delegación innecesaria → **Inline Function**, **Inline Class**
3. Si un parámetro de un método no se utiliza → **Change Function Declaration**
4. Si el único uso de un módulo son casos de prueba → **Remove Dead Code**

Variables Temporales

▼ **Explicación**

- Ocurre cuando una clase tiene variables que solo utiliza en casos especiales

▼ **Solución**

1. Si son varias variables que tiene sentido que estén juntas → **Extract Class**
2. Si es una variable usada en un código condicional → **Introduce Special Case**

Cadenas de Mensajes

▼ **Explicación**

- Ocurre cuando se forman cadenas de métodos largas

▼ **Solución**

1. Si algún miembro de la cadena se puede encargar de una parte → **Hide Delegate**
2. Si se puede extraer una parte de la cadena → **Extract Function**, **Move Function**

Intermediario (Middle Man)

▼ **Explicación**

- Ocurre cuando la delegación en un caso no es tan necesaria

▼ **Solución**

1. Si la utilidad de una clase es solo delegar → **Remove Middle Man**

2. Si solo son un par de métodos que no hacen mucho → **Inline Function**
3. Si hay comportamiento adicional → **Replace Superclass with Delegate**, **Replace Subclass with Delegate**

Acoplamiento Fuerte (Insider Trading)

▼ **Explicación**

- Ocurre cuando dos módulos se referencian mucho hasta el punto de depender del otro

▼ **Solución**

1. Si podemos acercarlos → **Move Function**, **Move Field**
2. Si los módulos tienen intereses en común → Crear un tercer módulo o **Hide Delegate**
3. Si no se quiere que una clase hija conozca tanto a su clase padre → **Replace Subclass with Delegate**, **Replace Superclass with Delegate**

Clase Larga

▼ **Explicación**

- Ocurre cuando tengo una clase muy grande en comparación al resto

▼ **Solución**

1. Si puedo agrupar variables que están relacionadas entre sí → **Extract Class**
2. Si algunas variables están relacionadas con la clase, pero estarían mejor en una subclase o superclase → **Extract Superclass**, **Replace Type Code with Subclasses**
3. También podes ver como se utiliza la clase desde el sistema, si las otras clases suelen usar solo un subconjunto de sus campos y módulos → **Extract Class**, **Extract Superclass**, **Replace Type Code with Subclasses**

Clases Substitutas con Interfaces Diferentes

▼ **Explicación**

- Ocurre cuando la implementación no cumple con los nombres que pide la interfaz, o una clase tiene varias interfaces que no son compatibles

▼ **Solución**

1. Para hacer que los nombres de los métodos coincidan → **Change Function Declaration**
2. Para mover los métodos a donde sean necesarios o tengan sentido → **Move Function**
3. Si se produce duplicación de código → **Extract Superclass**

Clase Anémica

▼ **Explicación**

- Ocurre cuando una clase no hace otra cosa que almacenar datos

▼ **Solución**

1. Si la clase tiene campos públicos → **Encapsulate Record**
2. Si una variable tiene un setter al pedo o no debe ser cambiada → **Remove Setting Method**
3. Si se puede traer algún metodo a la clase → **Move Function**

No quiero mi herencia (Refused Bequest)

▼ **Explicación**

- Ocurre cuando encontramos un método que redefine a uno heredado pero hace algo totalmente diferente

▼ **Solución**

1. Si la clase no está posicionada correctamente en la jerarquía → Crear una clase hermana y **Push Down Method**, **Push Down Field**
2. Si la clase utiliza el comportamiento de su padre pero no implementa la interfaz → **Replace Subclass with Delegate**, **Replace Superclass with Delegate**

Comentarios

▼ **Explicación**

- Ocurre cuando no entendemos un fragmento de código o método si no es con un comentario

▼ **Solución**

1. Si necesitas un comentario para explicar un fragmento de código → **Extract Function**
2. Si ya lo extrajiste pero seguis necesitando un comentario para explicarlo → **Change Function Declaration**
3. Si necesitas declarar algunas reglas sobre el estado requerido del sistema → **Introduce Assertion**