David Mendoza Loor

**Diseño de Software**

**Trabajo Autónomo**

**Reporte de Refactoring**

Tabla de contenido

[1. Feature envy 2](#_Toc29650883)

[2. Primitive Obsession 3](#_Toc29650884)

[3. Data Clumps 4](#_Toc29650885)

[4. Refused Bequest 5](#_Toc29650886)

[5. Switch Statements 6](#_Toc29650887)

[Bibliografía 7](#_Toc29650888)

# 1. Feature envy

**Nombre en español:** Envidia de capacidades

**Descripción:** Este mal olor se da cuando en una clase utilizamos en forma excesiva los parámetros y métodos de otra clase en vez de crear el método en la clase que estamos utilizando mas, ya que esto esta comprometiendo la encapsulación entre clases que es lo que se busca evitar.

**Técnicas de Refactorización**

Entre las técnicas de Refactorización tenemos la siguiente:

1. En caso de existir una precondición en la que se utilice parámetros de otra clase, debemos guardar estos atributos en una nueva variable que va a reemplazar a los parámetros de la otra clase.
2. Encapsular la precondición en un método de la clase de la que se necesita los atributos en vez de tenerlos en otra clase rompiendo el encapsulamiento.
3. Encapsulamos la operación que se encuentra dentro de la condición.
4. Encapsulamos mas todavía creando un método que haga la condición y lo que este en el cuerpo de la condición en la clase de donde se esta utilizando los atributos. (Veljanovski, 2018)

**Ejemplo:** Ejemplo en el repositorio creado en GitHub en el paquete feature.envy.

# 2. Primitive Obsession

**Nombre en español:** Obsesión por los tipos primitivos

**Descripción:** Se da cuando podemos agrupar distintos atributos de una clase en una nueva clase ya que estos tienen una relación en común, en vez de tener una clase con muchos atributos primitivos. Las consecuencias que este mal olor causa son: código menos flexible, si queremos hacer modificaciones tendremos que hacer modificaciones a la misma clase en vez de a la clase del objeto violando el Open/Close principle de SOLID. Al añadir comportamientos a los datos tendrá que hacerse en la misma clase en que se los declara en vez de en la clase que se debería crear para manejar estos datos. Y por ultimo los datos primitivos no tiene los beneficios de ser Objeto.

**Técnicas de Refactorización**

Entre las técnicas de Refactorización tenemos las siguientes:

**Reemplazando datos primitivos por objetos**

1. Identificar los atributos primitivos que tenemos y como los podemos relacionar.
2. Crear clases nuevas donde los atributos de estas sean los atributos primitivos que queremos reemplazar a la clase que estamos refactorizando.
3. Ahora que tenemos clases para los datos primitivos podemos darle comportamiento adicional a los objetos que hemos creado mediante los datos primitivos.
4. Reemplazar los atributos primitivos que teníamos por objetos de las clases que creamos.

**Para resolver constantes primitivas en vez de objetos (Data Type)**

1. Cuando queremos representar un objeto por un string o numero como Data Type, por ejemplo, cuando tenemos un usuario que tiene un atributo char sexo que representa a hombres si el carácter es h y representa a mujeres si el carácter es m.
2. Hacemos subclases de ese objeto y cambiamos el comportamiento de la subclase cuando

**Ejemplo:** Ejemplo en el repositorio creado en GitHub en el paquete primitive.obsession.

# Data Clumps

**Nombre en español:** Agrupaciones de Datos.

**Descripción:** Este mal olor se da cuando frecuentemente utilizamos atributos juntos, y en vez de agruparlos en un objeto solo lo utilizamos como que si fueran atributos sin relación alguna. (Fowler, Data Clump, n.d.)

**Técnicas de Refactorización**

1. Identificar atributos que frecuentemente están siendo utilizados en métodos.
2. Separarlos del resto creando una clase que los contenga.
3. Reemplazar donde estaban puestos atributos por el objeto de la nueva clase creada. (Fowler, Data Clump, n.d.)

**Ejemplos:** Ejemplo en el repositorio creado en GitHub en el paquete data.clumps.

# 4. Refused Bequest

**Nombre en español:** Legado Rechazado

**Descripción:** Cuando se tiene una clase hija hereda a una clase padre, pero la clase hija no necesita todos los

metodos del padre por lo tanto rechaza la herencia del padre en sentido de que nunca los utiliza. Por lo que se

considera que la jerarquía es incorrecta.

**Técnicas de Refactorización**

**1.** Identificamos que métodos o que atributos no necesita la clase hija.

**2.** Creamos una nueva clase hija en la que pondremos los métodos y atributos que no necesitamos que sean heredados y la clase padre solo tendría lo que necesita la clase hija del inicio resolviendo este code smell. (Fowler & Beck, Refused Bequest, 2018)

**Ejemplos:** Ejemplo en el repositorio creado en GitHub en el paquete refused.bequest.

# 5. Switch Statements

**Nombre en Español:** Sentencias Switch

**Descripción:** Usar sentencias switch o varios else if para determinar un compartamiento distinto según sea el caso que se cumple en el swtich. Estas sentencias afectan la longitud y legibilidad del codigo ademas de que en varias ocasiones es necesaria hacerla mas de una vez en el codgio si es que no lo correjimos. (Vorobiov, 2018)

**Técnicas de Refactorización**

**Usando Polimorfismo**

**1.** Crear una interfaz con un método que es lo que se desarrollara dentro del switch.

**2.** Crear clases que implementen la interfaz y que para cada caso use lo que había dentro de cada

sentencia switch

**3.** Dependiendo del objeto que sea al utilizar el método hará lo que desea que hubiera hecho dentro del

Switch. (Vorobiov, 2018)

# Bibliografía

Veljanovski, B. (26 de Septiembre de 2018). *HASELT | Refactoring a feature envy code | Towards less coupled code, adhering to the 'Tell Don't Ask' principle*. Obtenido de HASELT: https://www.haselt.com/blog/refactoring-a-feature-envy-code

Jorge. (4 de Marzo de 2012). *Los Malos Olores del Codigo*. Obtenido de Tratando de Entenderlo: http://tratandodeentenderlo.blogspot.com/2012/03/los-malos-olores-del-codigo.html

Makowska, A. (15 de Noviembre de 2017). *Code Quality: Fighting Primitive Obsession Code Smells*. Obtenido de DZone: https://dzone.com/articles/code-quality-fighting-primitive-obsession-code-sme-1

Fowler, M. (s.f.). *Data Clump*. Obtenido de martinfowler.com: https://martinfowler.com/bliki/DataClump.html

Fowler, M., & Beck, K. (19 de Noviembre de 2018). *Refused Bequest*. Obtenido de InformIt: http://www.informit.com/articles/article.aspx?p=2952392&seqNum=23

Vorobiov, I. (20 de Febrero de 2018). *Refactoring an Ugly Switch Statement without Strategy Pattern*. Obtenido de Medium: https://medium.com/@ivorobioff/refactoring-an-ugly-switch-statement-without-strategy-pattern-9398cd651f62