

**Trabajo Autónomo: Refactoring**

**Diseño de Software**

**Nombre:** Kevin Briones Fuentes

**Objetivo:** Relacionar malos olores de programación con las técnicas de refactorización.

**Reporte:**

**Feature Envy (Característica de la envidia):** Un método de una clase accede a atributos de otra clase, esto es un problema porque los métodos deben requerir de atributos y parámetros que estén en el lugar donde fue creado, para que de esta forma este organizado el código y no haya duplicación.

* Move method, se mueve el método para la clase donde estén los parámetros que pide.
* Move field, se arregla cambiando el atributo a otra clase.
* Parameterize Method, de esta forma se parametriza el parámetro del método y puede recibir cualquier tipo de atributo.

**Primitive Obsession (Obsesión primitiva):** En ocasiones se crean atributos primitivos que exceden la cantidad adecuada en una clase, de modo que la clase se hace extensa, en cambio se podría unir los atributos según su comportamiento para organizar una clase que los agrupe.

* Extract class
* Introduce Local Extension
* Replace data value with object
* Replace array with object
* Replace Type Code with Subclasses
* Replace Type Code with State/Strategy

Con estos métodos de refactotring, se crearía otra clase la cual se dejaría la obsesión de atributos primitivos-

**Refused Request (Legado rechazado):** Hay una clase hija que no contiene todos los métodos implementados de la clase padre, lo cual genera una excepción ya que en este caso no podría existir la herencia.

* Replace method with method object
* Move method

Hay que considerer que si hay métodos que no están en la clase, se deben mover a otras clases para evitar los errores.

**Long Class (Clase grande):** Es un problema ya que, si hay una clase con mucho código, puede ser que la clase tenga muchas responsabilidades, y sea difícil de manejar.

* Replace method with method object
* Substitute algorithm
* Introduce foreign method
* Introduce local extension
* Duplicate observed data
* Replace Type Code with Class
* Replace Type Code with State/Strategy
* Parameterize Method

Para solucionar el smell de long class, hay que aplicar todos los técnicos posibles de refactoring, ya que estas ayudan a reducir el código tales como evitar variables duplicadas.

**Divergent Change (Cambio divergente):** Cuando hay un cambio pequeño en una clase, es un problema tener que cambiar todos los métodos en sí.

* Extract method
* Move field
* Hide Delegate
* Introduce foreign method
* Replace Conditional with Polymorphism
* Extract Subclass

Cuando hay clases que tienen varias responsabilidades, es mejor crear otras clases y mover las responsabilidades, cambiar los métodos donde correspondan y a la vez sus atributos, así se evita el smell de divergent change.