# PG_R01

# Proceso de refactorización- Fowler

**Joaquín García Molina - Miguel Collado Sáez**
**Grado en Ingeniería Informática**
**Universidad de Burgos**

# 1. Objetivos específicos

- Aplicar un proceso de refactorización sobre un pequeño proyecto existente
- Conocer cómo corregir defectos de código usando refactorizaciones
- Relacionar una lista de tareas de desarrollo y de mantenimiento software con el proceso de refactorización

# 2. Enunciado

Esta sesión aborda la refactorización del sistema de un videoclub presentado en las transparencias de teoría y en la sección “proceso de refactorización” del curso en línea https://www.udemy.com/refactoriza-paramejorar-la-calidad-del-codigo-java/.

La documentación del sistema está contenida en el repositorio público

https://github.com/clopezno/refactoring-fowler-example.

La descripción funcional de la aplicación:

• Cálculo e impresión de los alquileres en un Videoclub.

  ◦ El coste del alquiler depende del tiempo de alquiler y tipo de película.

  ◦ Tipos de películas: regular, children, new-release.

  ◦ El coste de alquiler también depende de los puntos de frecuencia en el alquiler que a su vez dependen del tipo de película.

Resultados de la revisión de código:

• El programa supera las pruebas funcionales con una cobertura mayor del 80%.

• Se observan problemas/defectos de diseño, método excesivamente largo en la clase

Customer.statement()que puede dificultar futuros cambios.

Los nuevos requerimientos a implementar en la aplicación son:

1. Producir una versión que genere un informe de alquiler en HTML para

Customer.statement()

◦ Solución no válida para la práctica: copy & paste statement -> problemas de código duplicado

2. Diseñar e implementar la clasificación de películas junto con las reglas para calcular el precio y los puntos frecuentes de alquiler.

En el repositorio Github se aconseja definir una issue/tarea por cada requerimiento y un commit por cada refactorización realizada asociando la issue correspondiente.

# 3. Enlaces

**Repositorio:**

https://github.com/mcs1005/refactoring-fowler-example

**Histórico de refactorizaciones:**

https://github.com/mcs1005/refactoring-fowler-example/blob/master/Hist%C3%B3rico%20refactorizaciones.pdf

# 4. Preguntas de reflexión

**¿Cómo has utilizado en la práctica tus conocimientos de defectos de diseño?**

Hemos utilizado estos conocimientos para poder detectar en el código del proyecto dichos defectos así como para guiarnos para implementar refactorizaciones que los solucionen. Por ejemplo, el defecto de diseño Long Method, nos ha permitido descubrir defectos en el método *statemet* de la clase Customer. El defecto de duplicación de código, nos ha servido a la hora de crear el nuevo método *statementHTML*, nos hecho darnos cuenta de que no podíamos simplemente copiar y pegar el código del método anterior y hemos implementado una solución alternativa.
También hemos detectado el defecto Switch Statement, dentro de un método de la clase Customer, con una herencia simulada ya que realizaba cálculos dependiendo del tipo de película, que hemos terminando moviendo a la clase Movie y generando una herencia real.

**¿Cómo has utilizado en la práctica tus conocimientos de métricas de código?**

Hemos utilizado la métrica de porcentaje de cobertura de tests del proyecto para cumplir con el 80% solicitado (alcanzando un 87% de cobertura).

**¿Cómo has utilizado en la práctica tus conocimientos de prueba y cobertura de pruebas?**

Los hemos utilizado para modificar los tests existentes tras las refactorizaciones y para implementar nuevos test que nos permitieran llegar al porcentaje de cobertura solicitado.
También hemos hecho uso de las interfaces visuales del IDE Eclipse que nos ayudan a ver qué fragmentos de código están o no cubiertos por los tests.

Por ejemplo:

```
1  package ubu.gii.dass.refactoring;
2
3  public class NewRelease extends MovieType {
4
5      @Override
6      public double getCharge(Rental rental) {
7          return rental.getDaysRented() * 3;
8      }
9
10     @Override
11     public int getType() {
12         return MovieType.NEW_RELEASE;
13     }
14
15     @Override
16     public int getFrecuentRenterPoints(Rental rental) {
17         int frequentRenterPoints = 1;
18
19         // add bonus for a two day new release rental
20         if (rental.getDaysRented() > 1)
21             frequentRenterPoints++;
22
23         return frequentRenterPoints;
24     }
25
26 }
27
```

Nos indica que el else del if marcado en amarillo no está cubierto.

**¿Se puede automatizar completamente el proceso de refactorización a través de herramientas?**

No, hay algunas refactorizaciones que no se pueden hacer de forma automatizada. Por ejemplo, hemos realizado las refactorizaciones Replace Type Code with Subclasses y Replace Type Code With Polymorphism, necesarias para implementar *polimorfismo* de la clase *MovieType* con las clases *Children*, *NewRelease* y *Regular* que implementan los mismos métodos pero con distinto comportamiento y hemos tenido que implementar dichas refactorizaciones de forma manual.

**¿Qué relación encuentras entre el proceso de refactorización y la utilización de sistemas de control de tareas y versiones?**

Consideramos que deberían ir siempre de la mano para poder tener un control claro de los cambios realizados en nuestro código al implementar las refactorizaciones.

De esta forma, en futuras modificaciones o ampliaciones del código, cualquier desarrollador puede tener una idea y visión clara de lo que hace y se ha hecho durante todo el diseño del código y su evolución.