UTN F r a
Avellaneda

Programación II

Clase 07

Programación Orientada a Objetos

Temario

Refactoring

- Definición
- Alcance
- Ventajas
- ¿Cuándo aplicarlo?

Code Smells

- Definición
- Catálogo

Refactorings

- Catálogo

01.
Refactoring

Refactoring

Mantenimiento de Software

- **Evolución**

  Con cada agregado o modificación de funcionalidad el código aumenta su complejidad.

- **Diseño**

  Lo que al principio era un buen diseño, luego puede no serlo.

- **Herramientas**

  Refactoring.
  Testing.

Refactoring

Definición

- **Sustantivo**

  Cambio en el código de software que modifica la estructura del mismo que agrega legibilidad.

- **Verbo**

  Acción de reestructurar un código aplicando refactorings.

Refactoring

Definición

Podemos pasar horas haciendo refactoring, durante las cuales estaremos aplicando varios refatorings individuales.

REFACTORS

Refactoring

¿Qué implica?

- **Mejora Legibilidad**

  El código fuente resultante es más fácil de entender y, por consiguiente, será más fácil de modificar en el futuro.

- **Pequeños refactorings**

  Al estar manipulando código fuente, el refactoring debe ser lo más pequeño posible de manera de no alterar el funcionamiento.

Refactoring

¿Qué NO implica?

- **Eficiencia**

  Al focalizarse en la legibilidad, el código fuente resultante puede ser más o menos eficiente.

- **Comportamiento**

  No modifica la funcionalidad, pero si puede modificar el mecanismo utilizado para implementarla.

- **Fixes**

  No arregla bugs. Teoría de los dos sombreros del desarrollador: features y refactoring.

INCREMENTAR FUNCIONALIDADES

REESTRUCTURAR

Refactoring

Ventajas

Diseño

- **Tiempo de desarrollo**

  El desarrollador tarda más trabajando con código pobre de diseño.

- **Mantenimiento**

  Un código en el cuál contribuyen muchos desarrolladores tiende a ver su diseño disminuido.

Refactoring

Ventajas

Legibilidad

- **Contribución**

  Un nuevo desarrollador puede acoplarse de manera más rápida al equipo de desarrollo.

- **Detección de bugs**

  Identificación del bloque de código que está generando comportamientos anómalos.

UTN F r a Avellaneda

¿Cuándo hacer Refactoring?

Preparatorio

- **Al agregar una nueva funcionalidad**

  Revisar cómo está estructurado el código actual.
  Si necesita mejoras, refactoring.

Existe una función similar a la que necesito desarrollar, pero tiene valores por defecto. Una modificación me permitiría reutilizar código.

¿Cuándo hacer Refactoring?

Comprensión

- **Al modificar una funcionalidad o arreglar un bug**

Primero, leer el código para ver qué hace.

Si hay deficiencia en captar la idea rápidamente, entonces requiere un refactoring.

Renombrar nombres de variables. Partir una función que realiza varias acciones en varias funciones relacionadas.

Mejorar la lógica.

UTN F r a Avellaneda

¿Cuándo hacer Refactoring?

Planificado

- **Dedicar tiempo de desarrollo**

La mayoría de los refactoring se dan cuando se programa ya que usualmente no es una actividad que pueda desacoplarse de la necesidad de contribuir código.

Planificar un refactoring es una **mala práctica**.

¿Cuándo hacer Refactoring?

Code Review

- **Revisión antes de contribuir**

Antes de realizar un merge el código es revisado por desarrolladores más experimentados.

Se lleva la legibilidad individual al nivel del equipo.

UTNFra Avellaneda

¿Cuándo hacer Refactoring?

¿Refactoring o rehacer?

- **Depende...**

  No existe una fórmula, es subjetiva de cada desarrollador.
  Se basa en experiencia: es difícil determinar la dificultad de un refactoring sin dedicarle tiempo.

- **Tiene que terminar**

  Un refactoring

02.
Code Smells

Code Smells

¿Qué son?

- Se trata del **sentido intuitivo** que genera la lectura del código en el desarrollador.
- “If it stinks, change it”. Es decir, **refactoring**.
- Este sentido se perfecciona con la **experiencia**.
- Para ayudar, existe un **catálogo** de códigos que generan “mal olor”.

Code Smells

Catálogo

- **Mysterious Name**

  Variables y métodos con nombres no representativos.

  Es uno de los “code smells” más comunes.

  Si no se nos ocurre un buen nombre para la variable probablemente exista un problema de diseño más que de creatividad.

- **Duplicate Code**

  Código con el mismo patrón repetido en varios lugares.

  Código más extenso.

Code Smells

Catálogo

- **Long Function/Method**

  Cuanto más larga es el método, más difícil es su interpretación.

  Indefectiblemente tienen más comentarios de lo que deberían y su nombre no representa su propósito.

- **Long Parameter List**

  Una gran cantidad de parámetros tiende a agregar confusión a la verdadera responsabilidad del método.

Code Smells

UTNFra Avellaneda

Catálogo

- **Large Classes**

  Cuando una clase hace muchas cosas (responsabilidad) termina con, también, con una gran cantidad de atributos.

  Lo que usualmente ocurre es que da lugar a código duplicado (por la no utilización, por ejemplo, de herencia) y descontrol.

- **Data Class**

  Clases con estructura, pero sin comportamiento.

  Generalmente da indicios de que el comportamiento que debería tener ha sido declarado en otra parte.

Code Smells

Catálogo

- **Shotgun Surgery**

  Cuanto cada vez que hay que hacer un cambio debe también realizarse pequeños cambios en varias clases.

  Esos lugares so, además, difíciles de encontrar, por lo que dan nacimiento a bugs.

- **Feature Envy**

  Modularizar, es una forma de organizar el código para agrupar interacciones.

  Este “mal olor” presenta cuando una función o módulo interactúa más frecuentemente con funciones en otros módulos.

03.

Refactorings

Refactorings

¿Qué son?

- Técnicas que sirven para lidiar con “Code Smells”.

- Dan una explicación acerca de:
  - **Motivo**.
    Identificación del “Code Smell”. Beneficios de aplicar el refactoring
  - **Mecánica**.
    Paso a paso de su implementación.
  - **Ejemplos**.
    Facilita el concepto.

Refactorings

Rename Variable

- **Motivación**

  El nombre de la variable no explica cuál es su propósito.
  Aplicarle un nombre acorde, autodescritptivo.

- **Mecánica**

  1. Encontrar las referencias a la variable mal nombrada.
  2. Cambiar el nombre en cada lugar necesario.
  Sólo es posible si la variable es privada. En ese caso de ser pública deberá encapsularse antes.

UTN F r a Avellaneda

Extract Method

- **Motivación**

  Un síntoma es que cuando método realiza varias tareas nos toma un mayor esfuerzo entender qué hace.

  La solución es extraer las tareas en métodos pequeños para mejorar la legibilidad, acota responsabilidades y permitir reutilización.

- **Mecánica**

  1. Crear un nuevo método.
  2. Incorporar el bloque de código a extraer (revisar variables).
  3. Borrar bloque de código e invocar nuevo método.

UTN F r a Avellaneda

Extract Variable

- **Motivación**

  Algunas expresiones (matemáticas, lógicas) puede ser muy complejas dificultando identificar el cálculo que hacen.

  La solución dividir la expresión en variables más significativas.

- **Mecánica**

  1. Crear una nueva variable.

  2. Incorporar la parte de la expresión a extraer.

  3. Reemplazar parte de la expresión por la nueva variable.

UTN F r a Avellaneda

Move Method

- **Motivación**

Se puede visualizar que los objetos no están bien agrupados, ya que tienen muchas interacciones, pero no están en el mismo módulo.

Si esto ocurre, el método no pertenece a la clase donde está declarado y hay que moverlo.

- **Mecánica**

1. Encontrar interacciones múltiples y repetidas con un objeto.
2. Definir a dónde pertenece la funcionalidad y crear método.
3. Reemplazar las múltiples interacciones por una sola.

Refactorings

Pull Up

- **Motivación**

  En las subclases hay métodos idénticos (mismo *cuerpo*), lo cual implica que hay código duplicado.

  La duplicidad se elimina llevando el método a la superclase.

- **Mecánica**

  1. Verificar que los métodos son idénticos.
  2. Asegurarse que las variables son accesibles desde la superclase.
  3. Crear nuevo método en la superclase.
  4. Borrar métodos de las subclases para evitar *override*.

UTN F r a Avellaneda

Encapsulate Collection

- **Motivación**

  Exponer la estructura de datos de un atributo de una clase genera acople, lo que genera que un cambio rompa la lógica en otro lugar.

  Se soluciona encapsulando a la colección, ofreciendo la información necesaria y la forma requerida.

- **Mecánica**

  1. Encontrar donde se accede una colección (property).
  2. Incorporar métodos para proveer la colección (read-only).
  3. Incorporar métodos para modificar la colección.

Replace Type with Subclass

- **Motivación**

  A veces se incorpora un atributo que define un “tipo”, una clasificación para la clase que luego se utiliza para tomar decisiones.

  Esto indica que se requiere la utilización de subclases.

- **Mecánica**

  1. Elegir un tipo y crear una subclase.
  2. Incorporar un switch para retornar la clase específica.
  3. Eliminar el atributo.
  4. Aplicar **Replace Conditional with Polymorfism** de ser necesario.

Refactorings

UTNFra Avellaneda

Split Loop

- **Motivación**

  Por motivos que creemos de eficiencia utilizamos una iteración para realizar más de una tarea. Se pierde noción de la responsabilidad.

  Iterar una vez por tarea le da mayor legibilidad.

  La eficiencia no es parte del refactoring.

- **Mecánica**

  1. Identificar las tareas en el loop.

  2. Duplicar loop con una única tarea.

Refactorings

CODE SMELL	REFACTORING
Long Method	Extract method. Decompose Conditional. Replace Temp with Query.
Long Parameter List	Replace Parameter with Method. Introduce Parameter Object. Preserve Whole Object. Remove Flag Argumen.
Shotgun Surgery	Move Function. Move Field. Inline Function.
Feature Envy	Move Function. Extract Method.
Data Class	Encapsulate Record. Move Method.

Refactorings

Referencias

- **Refactoring Guru**

  https://refactoring.guru/refactoring/techniques

- **Refactoring: Improving the Design of Exisisting Code.**

  Martin Fowler. 2nd Edition.