Resumen Clase 2

PDF 1:

Refactoring-Ejemplo:

Extract Method: se usa cuando se tiene un metodo largo, habria que dividirlo.

Mecánica:

1. Crear un nuevo método cuyo nombre explique su propósito

2. Copiar el código a extraer al nuevo método

3. Revisar las variables locales del original

4. Si alguna se usa sólo en el código extraído, mover su declaración

5. Revisar si alguna variable local es modificada por el código extraido. Si es solo una, tratar como query y asignar. Si hay más de una no se puede extraer.

6. Pasar como parámetro las variables que el método nuevo lee.

7. Compilar

8. Reemplazar código en método original por llamada

9. Compilar

En Eclipse se puede hacer haciendo REFACTOR -> Extract Method

Importante testear luego de hacer cambios y usar nombres claros.

Move Method: Un metodo usa muchos servicios que estan definidos en una clase diferente a la suya. Se soluciona moviendo este metodo a la clase.

A mano:

1. Revisar las v.i. usadas por el método a mover. Tiene sentido moverlas también?

2. Revisar super y subclases por otras declaraciones del método. Si hay otras tal vez no se pueda mover.

3. Crear un nuevo método en la clase target cuyo nombre explique su propósito

4. Copiar el código a mover al nuevo método. Ajustar lo que haga falta

5. Compilar la clase target

6. Determinar como referenciar al target desde el source

7. Reemplazar el método original por llamada a método en target

8. Compilar y testear

9. Decidir si remover el método original o mantenerlo como delegación

Replace temp with query: usarlo para evitar metodos largos ya que las var temporales al ser locales, fomentan metodos largos. Como solucion extraer la expresion en un metodo y reemplazar todas las referencias a la var temporal por la expresion. Osea tener un metodo que realice el calculo y devuelva la variable.

Mecanica:

1. Encontrar las vars. temporales con una sola asignación (si no, Split Temporary Variable)

2. Extraer el lado derecho de la asignación (tener cuidado con los efectos colaterales; si no, Separate Query From Modifier)

3. Remplazar todas las referencias de la var. temporal por el nuevo método

4. Eliminar la declaración de la var. temporal y las asignaciones

5. Compilar y testear

Rename method: Renombrar metodos para que sean claros.

Para eliminar un IF con 3 condiciones se pueden extender subclases y tener una solucion por cada if. En el ejemplo de partido pasa a jugador y este jugador tiene 3 subclases que realizan el metodo a su manera.

Replace Conditional with Polymorphism:

Mecánica:

1. Crear la jerarquía de clases necesaria

2. Si el condicional es parte de un método largo: Extract Method

3. Por cada subclase:

1. Crear un método que sobreescribe al método que contiene el condicional

2. Copiar el código de la condición correspondiente en el método de la subclase y ajustar

3. Compilar y testear

4. Borrar la condición y código del branch del método en la superclase

5. Compilar y testear

4. Hacer que el método en la superclase sea abstracto

.La mejor manera de optimizar un programa, primero es escribir un programa bien factorizado y luego optimizarlo. Mientras que estamos refactorizando no es necesario pensar en la perfomance ya que aunque perdamos un poco por el lado de la performance vamos a ganar en flexibilidad y reusabilidad.

PDF 2:

Refactoring:

BAD SMEELS:

-Duplicate Code: El mismo codigo, o codigo muy similar, aparace en muchos lugares.

-Large Class: Una clase intenta hacer demasiado trabajo, tiene muchas variables de instancia o tiene muchos metodos.

-Long Method: Un metodo tiene muchas lineas de codigo. Cuanto mas largo es mas dificil de entenderlo, cambiarlo y reusarlo.

-Envidia de atributo: Un metodo en una clase usa principalmente los datos y metodos de otra clase para realizar su trabajo (se muestra “envidiosa” de las capacidades de otra clase)

-Data Class: Una clase que solo tiene variables y getters/setters, actua unicamente como contenedor de datos. En general sucede que otras clases tienen metodos con “envidia de atributo”, esto indica que esos metodos deberian estar en la “data class”, suele indicar que el diseño es procedural.

-Condicionales: Cuando sentencias condicionales contienen logica para distintos tipos de objetos, cuando todos los objetos son instancias de la misma clase indica que se necesitan crear subclases. El problema es que la misma estructura condicional aparece en muchos lugares.

-Long Parameter List: Un metodo con una larga lista de parametros es mas dificil de entender. Tambien es dificil obtener todos los parametros para pasarlos en la llamada entonces el metodo es mas dificil de reusar y la excepcion es cuando no quiero crear una dependencia entre objetos llamador y el llamado.

Malos olores y sus posibles soluciones:

.Código duplicado

-Extract Method -Pull Up Method -Form Template Method

.Métodos largos

-Extract Method -Decompose Conditional -Replace Temp with Query

.Clases grandes

-Extract Class -Extract Subclass

.Muchos parámetros

-Replace Parameter with Method -Preserve Whole Object -Introduce Parameter Object

.Cambios divergentes (Divergent Change)

-Extract Class

.“Shotgun surgery” (cirugia de escopeta)

-Move Method/Field

.Envidia de atributo (Feature Envy)

-Move Method

.Data Class

-Move Method

.Sentencias Switch

-Replace Conditional with Polymorphism

.Generalidad especulativa

-Collapse Hierarchy -Inline Class -Remove Parameter

.Cadena de mensajes (banco cuentaNro: unNro) movimientos first fecha

-Hide Delegate -Extract Method & Move Method

.Middle man

-Remove Middle man

.Inappropriate Intimacy

-Move Method/Field

.Legado rechazado (Refused bequest)

-Push Down Method/Field

.Comentarios

-Extract Method -Rename Method

Catalogo de refactorings:

En la organización de catalogo Fowler tenemos varios items: Composición de métodos, mover aspectos entre objetos, organización de datos, simplificación de expresiones condicionales, simplificación en la invocación de métodos, manipulación de la generalización, big refactorings.

Composicion de metodos: Permiten “distribuir” el codigo adecuadamente, el uso de metodos largos son problematicos ya que contiene mucha informacion y/o logica compleja.

Extract method, inline method, replace temp with query, split temporary variable, replace method with method object, substitute algorithm.

Mover aspectos entre objetos: Ayudan a mejorar la asignacion de responsabilidades.

Move method, move field, extract class, inline class, remove middle man, hide delegate.

Manipulacion de la generalizacion: Ayudan a mejorar las jerarquiaws de clases.

Push up / down field, push op / down method, pull up constructor body, extract subclass / superclass, collapse hierarchy, replace inheritance with delegation, replace delegation with inheritance.

Organización de datos: Facilitan la organización de atributos.

Self encapsulate field, encapsulate field / collection, replace data value with object, replace array with object, replace magic number with symbolic constant.

Simplificaciones de expresiones condicionales: Ayudan a simplificar los condicionales.

Decompose conditional, consolidate conditional expression, consolidate duplicate conditional fragments, replace conditional with polimorfism.

Simplificacion de invocacion de metodos: Sirven para mejorar la interfaz de una clase.

Rename method, preserve whole object, introduce parameter object, parameterize method.

PDF 3:

Refactoring tools:

Cuando aplicar refactoring:

-En el contexto de TDD

-Cuando se descubre codigo con mal olor, aprovechando la oportunidad. Dejarlo al menos un poco mejor, dependiendo del tiempo que lleve y de lo que este haciendo.

-Cuando no puedo entender el codigo. Aprovechar el momento en que lo logro entender.

-Cuando encuentro una mejor manera de codificar algo.

Los dos sombreros: Adding Function (Se exploran ideas, se corrigen bugs) y refactoring (solo puedo refactorizar con tests en verde).

Puedo cambiar de sombrero frecuentemente, pero solo puedo usar 1 sombrero por vez.

Automatizacion del refactoring:

.Las herramientas solo chequean lo que sea posible desde el arbol de sintaxis y la tabla de simbolos. Pueden ser demasiado conservativas (ya que no realizan un refactoring si no pueden asegurar preservacion de comportamiento) o asumir buenas tecnicas de programacion.

Ejemplos: SmallTalk RB, IntelliJ IDEA, Sonar qube.