Refactorización Aspectual

Aspectual Refactoring

Alexis Messino Soza*, Dairo Arias Morales** y Sergio Obredor Castro***

*amessino@unisimonbolivar.edu.co
Universidad Simón Bolívar
Barranquilla – Atlántico

**dariuscley@hotmail.com
***soyac@live.com

Palabras clave:
Refactorización, Aspectos, POA, Ingeniería del Software

Resumen
En este artículo se aborda el proceso de refactorización, enfatizando como se lleva a cabo a nivel de aspectos, mostrando las diferentes técnicas y métodos que se utilizan en esta, presentando con ejemplos simples, el uso de cada uno de estos métodos.

Key-words:
Refactoring, Aspect, AOP, Software engineering

Abstract
This article addresses the process of refactoring, emphasizing how it is carried out at the level of issues, showing the different techniques and methods used here, with simple examples showing the use of each of these methods.

I. INTRODUCCION

El mantenimiento de productos software legados trae consigo una gran inversión de tiempo y de recursos, debido a la poca organización del código fuente, muchas veces la forma arcaica en la que está distribuida y organizada el código fuente de un sistema legado, truncan mucho características fundamentales tales como la escalabilidad, y como ya se ha mencionado el mantenimiento.

La necesidad de solventar estos problemas ha suscitado el nacimiento y posterior evolución de técnicas tales como la refactorización, cuyo objetivo es llevar a cabo el proceso mediante el cual se busca mejorar la estructura interna del código, sin alterar el comportamiento externo del mismo [1], en cuanto a esta técnica de reingeniería, la cual se potencializa cuando es integrada con la programación orientada a aspectos, mezclando sinérgicamente estas dos técnicas para da paso a lo que ha sido llamado la refactorización orientada a aspectos [2].

La refactorización es una técnica utilizada en la ingeniería de software, y cuando es utilizada específicamente en la programación orientada a aspecto, ayuda a mejorar y evolucionar el código debido a la manera de encapsular los crosscutting concerns [1]. Tales crosscutting concerns cortan trasversalmente los componentes funcionales de una aplicación, mejorando la modularización y el mantenimiento [4] de un sistema a través del uso de nuevas unidades de modularización llamadas aspectos.

II. PROGRAMACION ORIENTADA A ASPECTOS

Nace como paradigma de programación el cual utiliza los aspectos como unidad básica, pero que también soporta la manipulación de objetos, funciones, procedimientos, etc..., tal como sus antecesores, es decir, trata de solucionar el problema de la diseminación de código, propio de las anteriores técnicas [7].

Es por eso que algunos autores la han clasificado como la 5ta generación [6], ya que vincula estructuras soportadas por anteriores y las entrelaza con sus nuevos conceptos, tales como aspecto, tejedor, etc... que serán tratados más adelante. La programación adaptativa es un precedente de la POA, ya que presento ciertas bases, pero no fue hasta 1995 cuando Gregor Kiczales introdujo el término

III. REFACTORINGS

La refactorización agrupa una serie de métodos o técnicas denominadas refactorings, los cuales son utilizados para lograr la optimización de la estructura interna del código, he aquí una lista de los más utilizados:

Método extraer (Extract Method): Si hay fragmentos de código que pueden ser agrupados en un nuevo método que contenga este código, debe utilizarse el método extraer. [1]

Ejemplo:

```
void printOwing() {
        printBanner();
        //print details
      System.out.println   ("name:   "   +
      _name);
      System.out.println    ("amount"    +
      getOutstanding());
}
```

Codigo diseminado.

```
void printOwing() {
        printBanner();
        printDetails(getOutstanding());
}

void printDetails (double outstanding) {
      System.out.println   ("name:   "   +
      _name);
      System.out.println    ("amount"    +
      outstanding);
}
```

Codigo refactorizado.

Método mover (Move method): Crea un método con un código similar en la clase en la que el método es más utilizada. [1]

Ejemplo:

```
class Project {
        Person[] participants;
}
class Person {
        int id;
  boolean participate(Project p) {
  for(int                      i=0;
  i<p.participants.length; i++) {
        if  (p.participants[i].id
  == id) return(true);
  }
  return(false);
  }
}

... if (x.participate(p)) ...
```

Codigo diseminado.

```
class Project {
  Person[] participants;
  boolean participate(Person x) {
  for(int                       i=0;
  i<participants.length; i++) {
  if  (participants[i].id  ==  x.id)
  return(true);
    }
    return(false);
  }
}

class Person {
  int id;
}

... if (p.participate(x)) ...
```

Codigo refactorizado.

La tabla 1 presenta una lista resumida de refactorings con una breve descripción en inglés

Tabla 1. Lista de refactorings con breve descripción en ingles

REFACTORING	DESCRIPCIÓN
Add Parameter	A method needs more information from its caller.
Change Bidirectional Association to Unidirectional	You have a two-way association but one class no longer needs features from the other.
Change Reference to Value	You have a reference object that is small, immutable, and awkward to manage.
Change Unidirectional Association to Bidirectional	You have two classes that need to use each other's features, but there is only a one-way link.
Change Value to Reference	You have a class with many equal instances that you want to replace with a single object.
Collapse Hierarchy	A superclass and subclass are not very different.
Consolidate Conditional Expression	You have a sequence of conditional tests with the same result.
Consolidate Duplicate Conditional Fragments	The same fragment of code is in all branches of a conditional expression.
Convert Dynamic to Static Construction by Gerard M. Davison	You have code that loads other classes dynamically. This can introduce a un-warranted overhead and can produce code that is more fragile.
Convert Static to Dynamic Construction by Gerard M. Davison	You have classes that have static compile time dependencies on classes that can only be built on a specific platform.
Decompose Conditional	You have a complicated conditional (if-then-else) statement.
Duplicate Observed Data	You have domain data available only in a GUI control, and domain methods need access.
Eliminate Inter-Entity Bean Communication (Link Only)	Inter-entity bean relationships introduce overhead in the model
Encapsulate Collection	A method returns a collection.
Encapsulate Downcast	A method returns an object that needs to be downcasted by its callers.
Encapsulate Field	There is a public field.
Extract Class	You have one class doing work that should be done by two.
Extract Interface	Several clients use the same subset of a class's interface, or two classes have part of their interfaces in common.
Extract Method	You have a code fragment that can be grouped together.
Extract Package by Gerard M. Davison	A package either has too many classes to be easily understandable or it suffers from the 'Promiscuous packages' smell.
Extract Subclass	A class has features that are used only in some instances.
Extract Superclass	You have two classes with similar features.
Form Template Method	You have two methods in subclasses that perform similar steps in the same order, yet the steps are different.
Hide Delegate	A client is calling a delegate class of an object.
Hide Method	A method is not used by any other class.
Hide presentation tier-specific details from the business tier (Link Only)	Request handling and/or protocol-related data structures are exposed from the presentation tier to the business tier
Inline Class	A class isn't doing very much.
Inline Method	A method's body is just as clear as its name.
Inline Temp	You have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactorings.

IV. TIPOS DE ASPECT REFACTORINGS

Existe una larga lista de métodos que son utilizado en un proceso de refactorización, los cuales son también utilizados en la refactorización orientada a aspectos, para la identificación de los Crosscutting concerns, "Aunque la mayoría de los principios conocidos de AOP se continúan aplicando a la Refactorización AO, unos pocos de esos principios adquieren una importancia especial o una perspectiva diferente en Refactorización AO. " [3].

Al mismo tiempo para llevar a cabo la refactorización de aspectos, hay que tener en cuenta mecanismos o estrategias de reestructuración de código [4]

Por ejemplo para llevar a cabo una migración de sistemas orientados a objetos a orientado a aspectos es necesario tener en cuenta 3 tipos de aspect refactoring: [5]

Refactorings Aspect-Aware OO (orientada a objetos): Agrupa los Refactoring orientados a objetos, que se extendieron y adaptaron a la refactorización orientada aspectos [5]. Cuando se somete a un proceso de refactorización a una aplicación existen muchas estructuras orientadas a objetos, ya que la mayoría de la lógica del negocio se encuentra encapsulada en estructuras orientadas a objetos, sin embargo al momento de refactorizar de una aplicación orientada a objetos a una orientada a aspectos, muchos de los refactorings Aspect-Aware sufren cambios, para poder soportar el nuevo nivel de abstracción al que son sometidos.

Refactorings de construcciones AOP (programación orientada a aspectos): Estos refactorings poseen la propiedad de que están orientada a la POA [5]. Gracias a la característica principal de la POA, de encapsular a un nivel más alto los distintos aspectos de una solución, es posible encontrar este tipo de construcciones codificadas de forma implícita en muchas aplicaciones con código no tratado. La principal diferencia entre Aspect oriented refactoring y object oriented refactoring es que los primeros se centran en objetos y aspectos, mientras que los segundos solo se centran solo en objetos [8].

Refactorings de CCC (crosscutting concerns) Este tipo de refactorings son pequeñas trasformaciones de código orientado a aspectos. [5]

V. CONCLUSIONES

En este artículo mostramos una técnica de refactorización orientada a aspectos, refiriéndonos a los refactorings, que son métodos con los cuales se busca mejorar pequeños fragmentos de código, en un sistema legado, dividiendo estos refactorings en 3 tipos: Refactorings Aspect-Aware OO, Refactorings de construcciones AOP, Refactorings de CCC, de esta división consideramos que la practicas más apropiada seria utilizar los refactorings orientados a aspectos, ya que no solo están centrados en los aspectos, sino que es posible utilizarlos también en estructuras orientadas a objetos [8], además de que cuando se refactoriza en un contexto de aspectos, se logra una mayor abstracción de los distintos componentes de un programa.

Examinaremos el proceso de la Refactorización AO a través de este ejemplo con una extensión de Java conocida como AspectJ el cual se explica de una manera sencilla el proceso de refactorización antes y después.se conserva las características ventajosas de Java.

Como, por ejemplo la independencia de la plataforma la funcionalidad de los Aspectos se expresa utilizando Java estándar utilizamos la implementación trasversal dinámica, que permite definir un comportamiento adicional(afectando, modificando o añadiendo código),el cual se ejecutara en puntos específicos dentro del programa esta se especifica a través de los puntos de enlace, que son puntos bien definidos en la ejecución de un programa por ejemplo, la llamada a un método, el acceso a un atributo, ctc donde puntos de corte: Agrupan los puntos de enlace al igual que en una clase, en el aspecto se puede crear instancias.

Examinaremos el proceso de la Refactorización AO a través de este ejemplo con una extensión de Java conocida como Aspect J [9] el cual se explica de una manera sencilla el proceso de refactorización antes y después. Del proceso de refactoring for extracting point-cut; donde se conserva las características de Java.

En la figura 1 se presenta un fragmento de programa que contiene un aspecto AspectSample se declara una clase class1 el cual tiene un método principal que es declarado.

Tanto antes y después del refactoring se unen a un punto destino donde recibe una llamada del método (call *Sample.pm())

Con el fin de volver a utilizar este punto de unión podemos realizar una refactorización para extraer el PointCut para formar un nuevo PointCut del methodCall. De este modo, el pointCut extraidos del MethodCall puede ser reutilizado.

Antes de la refactorización

```
class Class1 {
 public static void main(String args[])
{
  Class1 c1 = new Class1();
  c1.method();
 }
 void void method() {
  System.out.println("method");
 }
}
aspect AspectSample {
 before(): call(void Class1.method())
 {
  System.out.println("before method");
 }
 after(): call(void Class1.method())
 {
  System.out.println("after method");
 }
}
```

Después de la refactorización

```
Aspect AspectSample{
 Before( ): call(* Sample.pm( )){
  System.out.println("pm ok");
 }
}
Class Sample{
 Public static void main (String rgs[]){
        new Sample ( ).print_method( );
 }
 Void print method( )
 {
  System.out.println("print Method");
 }
```

Figura 1. Un Refactoring orientado a aspectos que extrae un punto de corte

REFERENCIAS

[1] S. Casas, C. Marcos Exploración de Reglas de Inferencia para Automatizar la Refactorización Aspectual. Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina y el proyecto PICT 32079 Julio 2008.

[2] C. Majluf Refactorización Orientada al Aspecto. Julio 2007.

[3] S. Vidal, E. Abait, C. Marcos Un proceso iterativo para la refactorización de aspectos. Consejo Nacional de investigaciones Científicas y Técnicas Julio 2009.

[4] J. Hannemann, Aspect-Oriented Refactoring: Clasification and challenges. 5th International conference on Aspect-Oriented Software Development Bonn. Germany 2006.

[5] M. Fowler, Refactoring: Improving the Design of Existing Code. Junio 2000.

[6] E. Santiago Programación Orientada a Aspectos. Mayo 2009.

[7] R.Walker, E. Baniassad, G. Murphy An Initial Assessment of Aspect-oriented Programming. In Proceedings of the 21st International Conference on Software Engineering Mayo 1999.

[8] M. Iwamoto, J. Zhao Refactoring Aspect-Oriented Programs. Enero 2002.

[9] The AspectJ Team. The AspectJ Programming Guide.
2002.