ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 4, Issue 6, June 2016

More Refactoring's: Aspect Oriented Programming with AspectJ

[1]Geeta Bagade, [2]Dr. Shashank Joshi
[1]Ph.D. Scholar, [2]Professor/Ph.D Guide
Bharati Vidyapeeth, Pune, India

ABSTRACT: Even though Object Oriented Programming has been established firmly in the software industry, it has some disadvantages like code scattering, code tangling etc. Due to such limitations changes to the software become difficult. The size of the software goes on increasing whenever the code is changed. So it becomes weak and difficult to change. These are some of the limitations that should be resolved. Aspect Oriented Programming Languages provide us a way to solve these limitations. There are many AOP languages. One of them is AspectJ. The process of changing software is called as refactoring. By using refactoring we can change the existing software without affecting the behaviour of the software. In the previous paper, we had proposed a set of refactoring for Aspect Oriented Programs. In this paper, we propose some more refactoring's that can be used for Aspect Oriented Programming using AspectJ

KEYWORDS: Systems, Aspect Oriented Programming, Point cut, Join point, Refactoring Advice Aspect Oriented Programming, Aspect Oriented Concerns, AspectJ, Concerns, Aspect, Aspect Mining.

I. INTRODUCTION

Aspect Oriented Programming is used to solve the question of cross cutting concerns. These concerns are present everywhere in the software. Some of the examples of cross cutting concerns are exception handling, logging, security, synchronization etc. We use the concept of classes in OOP to handle such concerns. But then the code becomes more scattered and entangled. Even making a slight change to the software becomes difficult. To solve this problem of cross cutting concerns, we have Aspect Oriented Programming. Aspect Oriented Programming uses the notion of aspect. An Aspect is nothing but a class. This class manages the cross cutting concern. Therefore the code becomes more understandable, adaptable and good Here the main focus of the programmer are the cross cutting concerns. In the 20th century, the Xerox Palo Alto Research Center (Xerox PARC) invented Aspect Oriented Programming Various tools like AspectC, AspectC++, and AspectJ have this functionality. Aspect Oriented Programming solves the problem of code tangling, code scattering etc. Aspect Oriented Programming helps us in reusing the code and making the software more modular. It helps in reducing code scattering and code tangling. Since the arrival of Java and AspectJ, Aspect Oriented Programming is on its way to be a great success in the field of computer science after OOP.

II. REFACTORING'S IDENTIFIED

In the 1990's, refactoring surfaced. The main purpose of refactoring is to change the code in an organized way so that the probability of introducing errors/bugs is reduced. Refactoring can help in reducing the cost involved during development and maintenance. It will also help the systems to evolve. Refactoring helps us in changing the software in a way that the existing functionality/behavior is retained. Refactoring can be done by using manual method or by using a set of tools like Eclipse. A number of refactoring techniques like Assertions, Graph Transformations, Program Slicing, Software Metrics, Formal Concept Analysis, and Program Refinement are used. Here we present a new set of refactoring's identified. That can be used with AspectJ.

2.1. Name of the refactoring:

Introduce the get and set point cut, introduce before and after advice
The Syntax for the get point cut refactoring is point cut point cut Name (): get (Data type Variable Name)

The syntax for the set point cut is Point cut point cut Name (Data type Variable Name) : set (Data type Variable Name) && args (Variable Name)

This experiment also introduces before and after advice for the get and set point cuts.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 4, Issue 6, June 2016

Syntax for before and after advice for set point cut is
Before/after (Data type Variable Name): point cut Name (Variable Name)
Syntax for before and after advice for get point cut is before/after (): point cut Name ()

2.2. Refactoring Mechanics:

1. Identify the point cut that should be refactored
2. Introduce the get point cut refactoring by using the following syntax
3. point cut point cut Name(): get(Data type Variable Name)
4. Introduce the get point cut refactoring by using the following syntax
5. point cut point cut Name (Data type Variable Name) : set (Data type Variable Name)
6. &&args Variable Name)
7. Introduce the "before" and "after" advice for set point cut by using the following syntax
8. before/after (Data type Variable Name) :
9. point cut Name (Variable Name)
10. Introduce the "before" and "after" advice for get
11. point cut by using the following syntax
12. before/after() : point cut Name()
13. Test whether the code that is restructured now retains the behavior

```
Class My Class {private int p2 = 0; void increment (int z2) {p2 = p2+z2;}
 int get Value() { return p2; }

 Recharged public static void main (String [] args) {My Class c = new My Class (); c increment (10);
             System.out.println("Code Executing"); System.out.println(c.get_Value()); }}

Aspect Aspect1 {static final int MAX = 1000; before(int p2): set(int p2) && args(p2)
 {System.out.println("Calling Set Method"); System.out.println(p2); System.out.println(thisJoinPoint.getSignature());}

 After (int p2): set (int p2) && args(p2) {System.out.println("Called Set Method"); System.out.println(p2);
              System.out.println(thisJoinPoint.getSignature());}
```

Before (): get (int p2){System.out.println("Calling Get Value Method");
stem.out.println(thisJoinPoint.getSignature());}

After (): get (int p2){System.out.println("Called Get Value Method");
System.out.println(thisJoinPoint.getSignature());}

Before (int z2, MyClass c2): call (void MyClass.increment (int)) && target(c2) && args(z2)
{System.out.println("Calling Increment Method"); System.out.println(thisJoinPoint.getSignature());
If (c2.get_Value ()+z2 > MAX) throw new RuntimeException(); }

Before (int z2, MyClass c2): call (void MyClass.increment(int)) && target(c2) && args(z2)
{System.out.println("Calling Increment Method"); System.out.println(thisJoinPoint.getSignature());
If (c2.get_Value () +z2 > MAX) throw new RuntimeException();}

Before (): call (int get Value ()) {System.out.println("calling Get_Value Method");
System.out.println(thisJoinPoint.getSignature());}

After (): call (int get_Value()) { System.out.println("Calling Get_Value Method");
System.out.println(thisJoinPoint.getSignature());}
Before (): execution (int get_Value()) { System.out.println("Executing Get_Value Method");

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 4, Issue 6, June 2016

```
             System.out.println(thisJoinPoint.getSignature());}

 After (): execution (int get_Value()) { System.out.println("Executing Get_Value Method");
             System.out.println(thisJoinPoint.getSignature()); }

After (int z2, MyClass c2): call (void increment (int)) && target (c2) && args(z2) {System.out.println("Called
Increment Method"); System.out.println(thisJoinPoint.getSignature());}

Before (int z2, MyClass c2): execution (void increment (int)) && target (c2) && args (z2)
{System.out.println("I am Executing Increment"); System.out.println(thisJoinPoint.getSignature()); }

After (int z2, MyClass c2): execution (void increment (int )) && target(c2) && args(z2)
{System.out.println("After Executing Increment"); System.out.println(thisJoinPoint.getSignature()); }}
```

2.3. The energy Refactored code:

```
Aspect Aspect1 {static final int MAX = 1000; point cut getVal() : get (int p2); point cut setI(int p2) :
               set(int p2) && args(p2); before(int p2): setI(p2) {System.out.println("Calling SetI Method");
               System.out.println(p2); System.out.println(thisJoinPoint.getSignature());}

After (int p2): setI (p2) { System.out.println("Calling SetI Method");
                   System.out.println(p2); System.out.println(thisJoinPoint.getSignature()); }

Before (): getVal() {System.out.println("Calling GetValueOfI Method");
               System.out.println(thisJoinPoint.getSignature()); }

After (): getVal() {System.out.println("After Calling GetValueOfI");
             System.out.println(thisJoinPoint.getSignature());}

Before (int z2, MyClass c2): call (void MyClass.increment(int)) && target(c2) && args(z2)
 {System.out.println ("I am Calling IncI"); System.out.println(thisJoinPoint.getSignature());
                         If (c.get_Value() + z2> MAX) throw new RuntimeException();}

Before (int z2, MyClass c2): call (void MyClass.increment(int)) && target(c2) && args(z2)
           {System.out.println("I am Calling IncI"); System.out.println(thisJoinPoint.getSignature());
           If (c2.get_Value () + z2 > MAX) throw new RuntimeException(); }

Before (): call (int get_Value()) {System.out.println
               ("I am calling get_Value"); System.out.println(thisJoinPoint.getSignature()); }

After (): call (int get_Value()) { System.out.println("After calling get_Value");
            System.out.println(thisJoinPoint.getSignature()); }

Before (): execution (int get_Value()) { System.out.println("I am executing get_Value");
            System.out.println(thisJoinPoint.getSignature()); }

After (): execution (int get_Value()) { System.out.println("After executing get_Value");
            System.out.println(thisJoinPoint.getSignature()); }

After (int z2, MyClass c2): call (void increment (int)) && target(c2) && args(z2)
 {System.out.println("After Calling IncI"); System.out.println(thisJoinPoint.getSignature()); }
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 4, Issue 6, June 2016

```
Before (int z2, MyClass c2): execution (void increment (int)) && target (c2) && args(z2){
                    System.out.println("I am Executing IncI"); System.out.println(thisJoinPoint.getSignature()); }

After (int z2, MyClass c2): execution (void increment (int )) && target(c2) && args(z2) {
                    System.out.println("After Executing IncI");  System.out.println(thisJoinPoint.getSignature()); }}
```

After introducing the get and set point cuts, we have checked if two classes (super and sub classes) have a variable with the same name, then does it affect the introduced get and set point cuts . So in this example, we have a class “New Class” which is extended from class “My Class”. Both have a variable with the name “p2”. While writing advice we have specified the name of the class, so it executes specifically for that class only. We observed that it does not affect the get and set point cuts. The code for the above is

```
Class NewClass extends MyClass {private int p2 = 0; void increment (int x) { p2 = p2+z2; }
 int get_Value1() { return p2; }
Public static void main (String [] args) { New Class d = new New Class (); d.increment(100);
          System.out.println("Working Prototype");
                    System.out.println(d.get_Value1()); }}
```

2.4. Name of the refactoring: Remove the word abstract for the aspect

An aspect which is abstract or concrete can extend a class. An aspect which is abstract or concrete can implement interfaces. By making use of abstract aspects we are in a position to create units of code that are re-usable. Some parts of code related to crosscutting implementation has to be done by concrete sub-aspects.

Any point-cut or any method can be marked as abstract by an abstract aspect. By doing so the base aspect can provide the implementation logic for the crosscutting logic without using the details of an aspect that is specific to a particular system.
Weaving does not occur in case of an abstract aspect. For weaving to happen, we need concrete aspects. An aspect should be declared as an abstract aspect if it contains

1. Abstract point cut Or
2. Abstract method

We can therefore say that abstract aspects are similar to abstract classes. Also if the sub-aspect that is creating using abstract aspect does not provide the definition for each and every abstract method or abstract point cut should declare itself as being abstract.

2.4.1. Refactoring Mechanics;
1. Identify the aspect that needs to be made abstract
2. Remove the keywords abstract applied to both point cuts and methods
3. Provide the body for the method as well as point cut
4. Test whether the code that is restructured retains the behaviour

2.4.2. Original Code;
Here we have a concrete aspect called “Concrete Aspect” that extends the “MyAbstract_Aspect” aspect. The “Concrete Aspect” has definitions for its abstract method and abstract. In this aspect, we have defined the getVal() point cut which will read the value of “i”. Many such sub-aspects can be created. The outcome of this is that the code that is present in the base aspect is common and shared by the sub-aspects. Therefore the sub-aspects can now have the code that is specific for a particular application. We have also implemented the abstract method getI()

```
Public aspect ConcreteAspect extends MyAbstract_Aspect {Private int i = 0;
          Public point cut getVal (): get (int i); Public int get_ Value () {return i; }
```

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Vol. 4, Issue 6, June 2016

Before (): getVal () {System.out.println("Before Calling GetValueOfI"); }

After (): getVal (){System.out.println("After Calling GetValueOfI"); }

Before (): call (int get_Value ()) {System.out.println("Before calling getI"); }

After (): call (int get_Value ()) {System.out.println("After calling getI"); }

Before (): execution (int get_Value ()) {System.out.println("Before executing getI"); }

After (): execution (int get_Value ()) {System.out.println("After executing getI"); }}

An abstract aspect is the one that contains

1. Abstract point cut and
2. Abstract method

Aspects can extend classes and abstract aspects. So we can reuse the aspects that are already written. The aspects which are abstract implement most of the logic. Here we have defined an abstract aspect that contains abstract method and abstract point cut. The code is shown below
Public abstract aspect My Abstract_Aspect {public abstract int get_Value (); public abstract point cut getVal ();
Before (): getVal (){ System.out.println("Getting I"); }}

Class My Class {private int x2 = 0; void incrment(int x) { x2 = x2+x;} int get_Value() {return x2;}
public static void main(String args[]) { MyClass2 c2 = new MyClass2(); c2.increment(10);
System.out.println(c2.get_Value()); System.out.println("Example Working");}}

III. RESULTS OF THE RESEARCH

3.1. Name of the refactoring: Make the aspect unprivileged

Time taken to execute the Original Code: 166.6 ms
Time taken to execute the Refactored Code: 146ms
Lines of Code in the Original Code: 11
Lines of Code in the Refactored Code: 20

3.2. Name of the refactoring: Replace the point cut name with its designator

Time taken to execute the Original Code: 8 ms
Time taken to execute the Refactored Code: 8.4ms
Lines of Code in the Original Code: 57
Lines of Code in the Refactored Code: 47

3.3. Name of the refactoring: Introduce the get and set point cut , introduce before and after advice

Time taken to execute the Original Code: 12.1 ms
Time taken to execute the Refactored Code: 11.3ms
Lines of Code in the Original Code: 71
Lines of Code in the Refactored Code: 78

4.4. Name of the refactoring: Remove the word abstract for the aspect

Time taken to execute the Original Code: 151 ms
Time taken to execute the Refactored Code: 146.9 ms
Lines of Code in the Original Code: 43
Lines of Code in the Refactored Code: 49

IV. CONCLUSIONS

In this paper, we have proposed a set of refactoring for Aspect Oriented Programming using AspectJ. The refactoring's introduced were to make the aspect unprivileged, replace point cut name with its designator, get and set point cut and before and after advice and remove the word abstract from the aspect. The refactoring was applied on the code and the execution time and number of lines of code was measured. In some cases even if the execution time decreased, the number of lines of code increased. In the other cases the number of lines of code has decreased but the execution time has increased. From these experiments, we can note that the behavior of the system was preserved and the system executed faster even if the number of lines of code increased. In future we intend to compare the original code and the refactored code in terms of vocabulary size, number of attributes, number of operations, number of statements, weighted operations per component etc and present the analysis for the same.

REFERENCES

1. A Rani, H Kaur, Refactoring Methods and Tools, International Journal of Advanced Research in Computer Science and Software Engineering, 2012; 2: 256- 260.
2. R AkilaDevi, ASPECT ORIENTED REFACTORING FOR SOFTWARE MAINTENANCE, International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 2013; 2: 79-84.
3. W David, Analysing Java System Properties, Software Composition Group (SCG),Institute of Computer Science and Applied Mathematics,University of Bern, Switzerland, Nov. 2013
4. KZN Winn, Quantifying and Validation of Changeability and Extensibility for Aspect-Oriented Software, International Conference on Advances in Engineering and Technology (ICAET'2014), 2014.
5. M Storzer, U Eibauer, S Schoeffmann, Aspect Mining for Aspect Refactoring: An Experience Report.
6. A Muhammad Sarmad et al., A systematic review of comparative evidence of aspect-oriented is programming, Information and Software Technology, 2010; 52: 871-887.
7. S Casas, CF Zamorano, Refactoring and AOP under Scrutiny, Computer Science and Engineering, 2014; 4: 7-16, 2014.
8. S Apel, D Batory, How AspectJ is Used: An Analysis of Eleven AspectJ Programs, Department of Informatics and Mathematics University of Passau, Germany, 2008.
9. Dr Rizvi and Z Khanam, Introduction of Aspect Oriented Techniques for refactoring legacy software, International Journal of Computer Applications, 2015; 1: 29-32.
10. T Hon, M Tkatchenko, Refactoring JQuery with AspectJ: an experience report, 2005.