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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

More Refactorings: Aspect Oriented Programming with AspectJ

Geeta Bagade (Mete), Dr. Shashank Joshi

Ph.D. Scholar, YMC, Bharati Vidyapeeth, Pune, India

Professor/ Ph.D Guide, COE, 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 refactorings that can be used for Aspect Oriented Programming using AspectJ

KEYWORDS: Systems, Aspect Oriented Programming, Pointcut, Joinpoint, 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 Centre (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. REFACTORINGS 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 refactorings identified. that can be used with AspectJ.

A. Name of the refactoring: Introduce the get and set pointcut, introduce before and after advice

The Syntax for the get pointcut refactoring is

pointcut pointcutName(): get(Datatype VariableName)

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

The syntax for the set pointcut is
pointcut pointcutName (Datatype VariableName) : set (Datatype VariableName) && args(VariableName)

This experiment also introduces before and after advice for the get and set pointcuts.
Syntax for before and after advice for set pointcut is
before/after (Datatype VariableName) : pointcutName (VariableName)

Syntax for before and after advice for get pointcut is before/after() : pointcutName()

Refactoring Mechanics

1. Identify the pointcut that should be refactored
2. Introduce the get pointcut refactoring by using the following syntax
   pointcut pointcutName(): get(Datatype VariableName)
3. Introduce the get pointcut refactoring by using the following syntax
   pointcut pointcutName (Datatype VariableName) : set (Datatype VariableName)
   &&args(VariableName)
4. Introduce the "before" and "after" advice for set pointcut by using the following syntax
   before/after (Datatype VariableName) : pointcutName (VariableName)
5. Introduce the "before" and "after" advice for get
   pointcut by using the following syntax
   before/after() : pointcutName()
6. Test whether the code that is restructured now retains the behaviour

After introducing the get and set pointcuts, 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 pointcuts. So in this example, we have a class "NewClass" which is extended from class "MyClass". 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 pointcuts.

B. 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 have to be done by concrete sub-aspects.
Any pointcut 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 pointcut 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 pointcut should declare itself as being abstract.

Refactoring Mechanics

1. Identify the aspect that need to be made abstract
2. Remove the keywords abstract applied to both pointcuts and methods
3. Provide the body for the method as well as pointcut

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

4. Test whether the code that is restructured retains the behaviour

Original Code

Here we have a concrete aspect called “ConcreteAspect” 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() pointcut 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()

An abstract aspect is the one that contains
1. Abstract pointcut 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 pointcut.

Refactored code

The refactoring proposed here is that we want to remove the word abstract for the aspect. For this specific example when this is done, we get errors that we cannot declare abstract pointcuts and methods So we remove the keywords abstract applied to both pointcuts and methods. In that case we need to provide the body for the method as well as pointcut

So the aspect looks like shown below. Now since this aspect is made a concrete aspect it cannot be extended by another aspect.

Since the earlier aspect is now concrete we will have to remove the word extends " MyAbstract_Aspect " and let the functionality be as it is. If this is done the aspect "MyAbstract_Aspect" does not remain re-usable.

III. RESULTS OF THE RESEARCH

A. **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

B. **Name of the refactoring: Replace the pointcut 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

C. **Name of the refactoring: Introduce the get and set pointcut, 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

D. **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

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

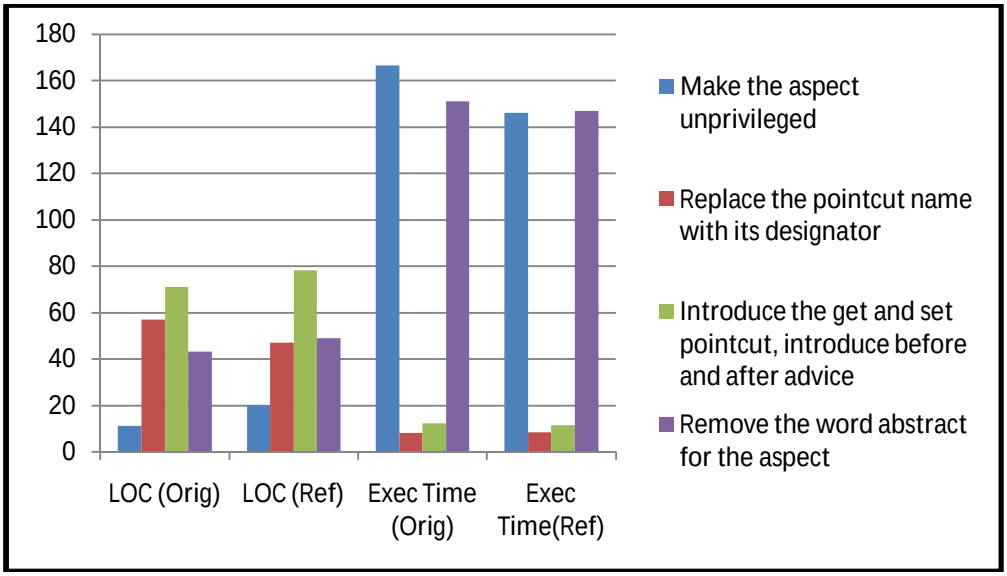
Vol. 4, Issue 4, April 2016

Lines of Code in the Original Code: 43
Lines of Code in the Refactored Code: 49

Table of Comparison before and after refactoring the code

S.No.	Name of the Refactoring	LOC (Orig)	LOC (Ref)	Exec Time (Orig)	Exec Time(Ref)
1	Make the aspect unprivileged	11	20	166.6	146
2	Replace the pointcut name with its designator	57	47	8	8.4
3	Introduce the get and set pointcut, introduce before and after advice	71	78	12.1	11.3
4	Remove the word abstract for the aspect	43	49	151	146.9

Graphs indicating the difference between original and refactored code



IV. CONCLUSION

In this paper, we have proposed a set of refactorings for Aspect Oriented Programming using AspectJ. The refactorings introduced were to make the aspect unprivileged, replace pointcut name with its designator, get and set pointcut and before and after advice and remove the word abstract from the aspect. The refactorings were applied on the code and the execution time and number of lines of code were 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 behaviour 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 and H. Kaur, "Refactoring Methods and Tools", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 12, pp. 256-260, 2012.
2. AkilaDevi R, "ASPECT ORIENTED REFACTORING FOR SOFTWARE MAINTENANCE", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 2, no. 4, pp. 79-84, 2013.

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

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

3. David W, “Analysing Java System Properties”, Software Composition Group (SCG),Institute of Computer Science and
4. Applied Mathematics,University of Bern, Switzerland, Nov 2013
5. K.Z.N.Winn,"Quantifying and Validation of Changeability and Extensibility for Aspect-Oriented Software", International Conference on Advances in Engineering and Technology (ICAET'2014), 2014.
6. M. Storzer, U. Eibauer and S. Schoeffmann, "Aspect Mining for Aspect Refactoring: An Experience Report".
7. Muhammad Sarmad Alia et al., "A systematic review of comparative evidence of aspect-oriented programming", Information and Software Technology, vol. 52, pp. 871-887, 2010.
8. S. Casas and C. F. Zamorano,"Refactoring and AOP under Scrutiny", Computer Science and Engineering, vol. 4, no. 1, pp. 7-16, 2014
9. S. Apel and D. Batory, "How AspectJ is Used: An Analysis of Eleven AspectJ Programs", Department of Informatics and Mathematics University of Passau, Germany ,2008.
10. Dr. Rizvi and Z. Khanam, "Introduction of Aspect Oriented Techniques for refactoring legacy software", International Journal of Computer Applications, vol. 1, no. 6, pp. 29-32, 2015.
11. T. Hon and M. Tkatchenko, "Refactoring JQuery with AspectJ: an experience report", 2005.

BIOGRAPHY

Geeta Bagade (Mete), a Master in Computer Science from the University of Pune and currently pursuing her Ph.D in Computer Science from Bharati Vidyapeeth, Pune has more than 12 years of experience in IT training. She possesses good technical skills with respect to programming languages as well as databases.

Dr. Shashank Joshi, is a B.E. in Electronics and Telecommunication from Govt. College of Engineering, Pune in 1988. He also completed the M.E. and Ph. D. Degree in Computer Engineering from Bharati Vidyapeeth Deemed University Pune. He is currently working as the Professor in Computer Engineering Department, Bharati Vidyapeeth Deemed University, College of Engineering, Pune. His research interests include software engineering. Presently he is engaged in SDLC and secure software development methodologies. He is a passionate professor with overall experience of more than 20 yrs.