REUSABILTY OF CODE WITH ASPECT MINING AND REFACTORING: A PROPODED APPOACH

Shivani Waman Borle, Priyanka Nana Kothawade, Vibhuti Vasant Vadje, Priyanka Uday Wani

Abstract-
Aspect-oriented software development allows the encapsulation of crosscutting concerns, achieving a superior system modularization and improving its maintenance. One important challenge is how to develop an object-oriented system into an aspect-oriented system in such a way that the system structure gets regularly improved. This paper describes a process to support developers in the refactoring of object-oriented systems to aspects. To do so, an approach that combines aspect mining techniques to apply refactoring with pattern matching & clustering techniques is used. This approach reduces time and space complexity. This paper also describes how code is replaced according to user's need.

Keywords:
Aspect mining, aspect refactoring.

1. Introduction:

Aspect-Oriented Programming (AOP) provides mechanisms for the separation of crosscutting concerns—functionalities scattered through the system and tangled with the base code. Existing systems are a natural test bed for the AOP approach since they often contain several crosscutting concerns which could not be modularized using traditional programming constructs. This paper presents an automated approach to the problem of migrating systems developed according to the Object-Oriented Programming (OOP) paradigm into Aspect-Oriented Programming (AOP).

The concerns that orthogonally crosscut the components of a system by means of aspects are encapsulated. These concerns

are called crosscutting concerns (CCCs) [7] and these CCC's are supported by Aspect-oriented software development (AOSD)[7]. CCC is difficult to modularized using traditional software engineering approaches (e.g., the object-oriented paradigm) to deal with the complexity and evolution of systems. Examples of CCCs are logging, exception handling and concurrency control. For existing object-oriented systems are unable to corporate the benefits of AOSD[7], so those systems are usually re-modularized into aspect oriented Systems. So we require the techniques and tools that can help developers and that can identify the crosscutting concerns, and this is called aspect mining, and then refactoring of those concerns is done into an aspect which is called as aspect refactoring.

2. Related Work:

In the migration of existing OO code to aspects [8], the problem that has received the most attention is the detection of candidate aspects (aspect mining). The problem of refactoring has been considered only more recently. This paper takes the results of aspect mining as its starting point and focuses on the problems associated with automating the refactoring process. Human guidance is important to ensure that inherent value judgments are taken into account. Some of the various aspect mining approaches rely upon the user's definition of likely aspects, usually at the lexical level, through regular expressions and support the user in the code browsing and navigation activities conducted to locate them. Other approaches try to improve the identification step by adding more automation. They exploit either execution traces or identifiers [33], often in conjunction with formal concept analysis [30], [33]. Clone detection [5], [29] and fan-in analysis [25] represent other alternatives in this category.

Among the programming languages and tools that have been developed to support AOSD, AspectJ [21], an extension of Java with aspects, is one of the most popular and best supported. AspectJ offers two main programming constructs to modularize crosscutting concerns: pointcuts and introductions. Pointcuts intercept the normal execution flow at specified join points. Aspect code (called advice) can be executed before, after, or instead of (around) the intercepted join points. Introductions modify properties of the classes they affect by adding methods or fields and by making them implement interfaces or specialized superclasses previously unrelated to them. Unlike pointcuts/ advices, which alter the dynamic behavior, introductions operate statically on the class members and structure (this may, in turn, have effects

on the dynamic behavior in the presence of dynamic method binding). Aspects can also modify the superclass or the implemented interfaces of a class, via the declare parents construct.

3. Proposed System for Aspect Mining Process:

The proposed approach consists of two main phases: [1]
(i) aspect mining[1]
(ii) aspect refactoring [7].
The first phase receives an object-oriented system as input, and produces a number of candidate aspects as output. These aspects are identified by making a dynamic analysis of the system and applying association rules on it. The information of candidate aspects and the initial system's source code are then passed to the aspect refactoring phase. In this phase, different refactoring are evaluated and ultimately applied to the code. As output, this second phase generates a new version of the system that contains aspect-oriented final code. Aspect mining discovers of crosscutting concerns in the source code that become aspects. (i.e. Candidate Aspect)
Aspect refactoring is the technique that accomplishes the necessary transformations in the code to convert the Candidate aspects into aspectual code.

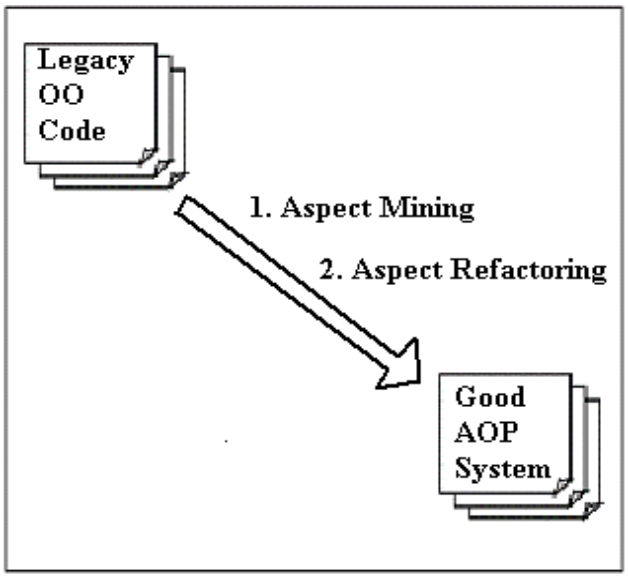

Fig 1: Aspect Mining and Refactoring Process

In this paper, we show the gradual evolution of an object-oriented system to an aspect oriented one. The aim of this approach is assisting the developer in: performing the evolution process, automating many tasks involved in this process, taking advantage of precise aspect mining techniques, and then applying different types of aspect refactoring. The migration from an OO [6] system to an AO one improves the structure and quality of the software, and thus eases software development.
Taking consideration of above line we believe that the provision of semi-automated support to help the developer to discover crosscutting concerns and to encapsulate them into aspects that are really beneficial. An originality of our approach is the use of dynamic analysis together with data mining techniques for identifying candidate aspects. Also, based on existing types of refactorings which automates the major steps of the migration, we present an aspect refactoring process.

3.1 Pattern Matching:

Data to be processed often does not decompose logically into independent Records with small identifiable pieces.[5] This type of data is characterized only by the fact that it can be written down as a string: A string can be a linear (Typically very long) sequence of characters. Strings are obviously central in "word processing" systems, which provide a variety of capabilities for the manipulation of text. Such systems process text strings, which might be loosely defined as sequences of letters, numbers, and special characters. These objects can be quite large (for example, this book contains over a million characters), and efficient algorithms play an important role in manipulating them. Another type of string is the binary string, a simple sequence of 0 and 1 value. This is merely a special type of text string, but it is worth making the distinction not only because different algorithms are appropriate but also binary strings arise naturally in many applications. Further the brute force algorithm is used for pattern matching

.3.1.1 Brute-Force Algorithm:

The method for pattern matching that immediately comes to mind is just to check, for each possible position in the text at which the pattern could match, whether it does in fact match.[5]
The following program searches in this way for the first occurrence of a pattern p [1. .M] in a text string a [1. .N]:

```
Function brutesearch: integer;
var i, j: integer;
begin
i:=l; j:=l;
repeat
if a[i]=p[j]
then begin i:=i+l; j:=j+l end
```

```
else begin i:=i-j+2; j:=1 end;
until (j>M) or (i>N);
if j>M then brutesearch:=i-M else brutesearch:=i
end;
```

The program keeps one pointer (i) into the text, and another pointer (j) into the pattern. As long as they point to matching characters, both pointers are incremented simultaneously. If the end of the pattern is reached (j>M), then a match has been found. If i and j point to mismatching characters, then j is reset to point to the beginning of the pattern and i is reset to correspond to moving the pattern to the right one position for matching against the text. If the end of the text is reached (i>N), then there is no match. If the pattern does not occur in the text, the value N+l is returned. In a text-editing application, the inner loop of this program is seldom iterated, and the running time is very nearly proportional to the number of text characters examined.

3.2 Refactoring Process:

This section introduces the iterative process for the conversion of existing OOP code to AOP code. [2] Our aspect re-factoring approach is based on different kinds of aspect refactoring. Specifically, we use the following classification: [6, 7]
The classifications are:

• Aspect-Aware OO Refactoring: [6, 7] This includes the object oriented refactoring which were extended and adapted to be used in the aspect-oriented paradigm. This type of re-factoring ensures that the OO refactoring correctly update the references to the AOP constructions.

• Refactoring for AOP constructs: In this type of refactoring, have the property of being specifically oriented to elements of the aspect-oriented programming. Its objective is mainly to improve the internal structure of aspects so that they are more legible and modifiable.

• Refactoring [7] of CCCs (Crosscutting concerns): The aim of this third group is to transform the crosscutting concerns in the aspects. Regarding the idea of the concepts of the aspect-oriented paradigm, these refactoring group the different concerns that are dispersed throughout the code when modularizing them into an aspect .

The proposed approach follows an process that starts with an object-oriented code and evidences of “aspectizable” code. In the process, each cycle produces a code refactoring by adding aspect-oriented paradigms. The main steps of the refactoring approach are as follow:

1. Get evidences of aspectizable code [2]: This step recovers the code that has been identified as aspectizable by the aspect mining phase. There is a description of OO code[2] attributes, methods, classes, etc. that should be refactorized to encapsulate the crosscutting concerns into the aspects. The connection with the aspect mining process is achieved through a XML file, which contains a list of candidate aspects with relevant data about those aspects.

2. Analyze possible refactoring [2] of CCCs: This step examines the possibility of applying one re-factoring of CCCs to the target code. That is, a set of valid refactoring is selected. The reason for using CCC refactoring in this step is because the fragments of aspectizable code identified in the previous step contain crosscutting concerns that must be encapsulated into an aspect.

3. Apply refactoring of CCCs: [2] The refactoring selected are executed, so that every crosscutting concern is extracted from the object-oriented code and converted as an aspect. The code refactoring are applied directly by the AspectRT tool. Sometimes the developer’s view is necessary for some decisions, like the choice of an aspect in which a fragment of code will be encapsulated, the name of a new point cut, etc.

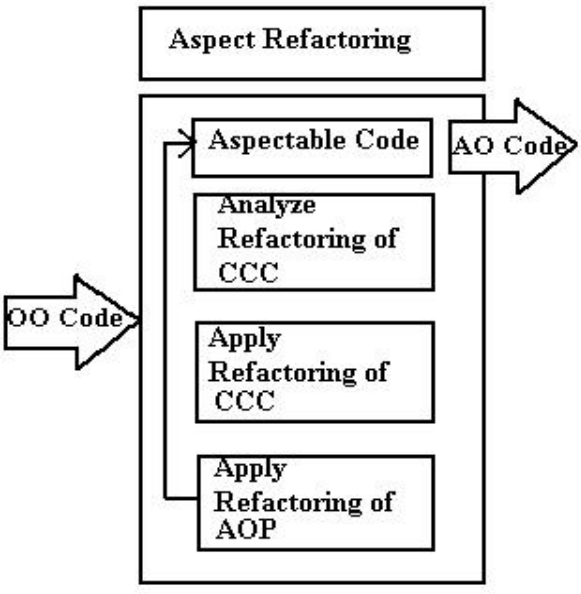

Fig 2: Proposed Refactoring Process

4. Apply OO refactoring [7] or Aspect-Aware OO: [2]If it is not possible to apply any refactoring of CCCs, this step seeks to apply object oriented refactoring and/or aspect-aware OO ones on the target code in order to restructure it and retry step 2. Sometimes, the identified code cannot be encapsulated directly into an aspect, and a previous OO refactoring is needed for the OO code to be adapted to the aspect refactoring pattern.

5. Apply refactoring for AOP constructs:[2] The last step tries to apply refactoring for AOP constructs to the aspect that has been modified in the application of refactoring of CCCs. Sometimes, when extracting a crosscutting concern, multiples refactoring are applied. So, the internal structure of the aspect that encapsulates the aspectizable code can need refactoring to improve its legibility and modularity, remove duplicate code etc.

3.2.1 K-means Algorithm:

The K-means Clustering technique[3] is easy to implement. Firstly K initial centroids are chosen, where K is number of clusters (user specified parameter).Each point is then assigned to closest centroid and each collection of points assigned to the centroid is a cluster. The centroid of each cluster is then modified based on the points assigned to the cluster. The procedure is repeated until the centroid remains same.

Conclusion:

This paper presents the approach for a refactoring process that assists the evolution of an object-oriented system into an aspect oriented system. The main advantages of the technique are: the automatic identification of scattering symptoms, and the generation of expressive rules describing the crosscutting. The main advantages of this technique are: the integration of different kinds of refactorings, and the automation of the transformations. The goal of this feature is that of showing more effective visualizations to the developer about relations between packages, classes, aspects, methods and crosscutting concerns. This way, the developer has a high-level vision of the current system architecture and possible evolution paths for it. The developer can then decide which aspects should be created, resolve encapsulation issues for these aspects, and check the effects of possible refactorings on the system.

This approach combines aspect mining techniques to apply refactorings with pattern matching & clustering techniques and reduces time and space complexity.

References:

[1] Santiago A. Vidal, Esteban S.Abait, Claudia Marcos, "Aspect Mining meets Rule-based Refactoring". *ISISTAN Research Institute, Faculty of Sciences, UNI-CEN University.*

[2] Dave Bink-ley, Mariano Ceccato, Mark Harman, Filippo Ricca, Paolo Tonella "Automated Refactoring of Object Oriented Code into Aspects".

[3] Van Deursen and T. Kuipers. "Identifying objects using cluster and concept analysis". *In Proc. Int. Conf. on Software Engineering (ICSE)*, pages 246-255. ACM, 1999.

[4] D. Janzen and K. De Volder. "Navigating and querying code without getting lost." *In Proc. 2nd Int. Conf. on Aspect-Oriented Software Development (AOSD), pages 178-187*. ACM Press, March 2003..

[5] Robert Sedgewick, Brown Unnermy "String processing algorithm"

[6] S. Hanenberg, C. Oberschulte and R. Unland. "Refactoring of aspect-oriented software." *In 4th International Conf. on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World*, pages 19-35, Erfurt, Germany, 2003

[7] J. Hannemann. "Aspect-Oriented Refactoring: Classification and Challenges". *Workshop on Linking Aspect Technology and Evolution (LATE'06). 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, Bonn, Germany, 2006.

[8] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Automated Refactoring of Object Oriented Code into Aspects," *Proc. Int'l Conf. Software Maintenance (ICSM)*, pp. 27-36, 2005.

[9] P. Tonella and M. Ceccato, "Aspect Mining through the Formal Concept Analysis of Execution Traces," *Proc. 11th Working Conf. Reverse Eng. (WCRE), pp. 112-121, Nov. 2004.*

[10] T. Tourwe and K. Mens, "Mining Aspectual Views Using Formal Concept Analysis," *Proc. Fourth IEEE Int'l Workshop Source Code Analysis and Manipulation (SCAM '04), pp. 97-106, Sept. 2004.*

[11] David Binkley, Member, IEEE, Mariano Ceccato, Student Member, IEEE, Mark Harman, Filippo Ricca, and Paolo Tonella, Member, IEEE "Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects "

[12] D. Shepherd, E. Gibson, and L. Pollock, "Design and Evaluation of an Automated Aspect Mining Tool," *Proc. Mid-Atlantic Student Workshop Programming Languages and Systems (MASPLAS), Apr-2004*

AUTHOR'S PROFILE

Shivani Waman Borle.

BECOMP, Sandip Institute of Technology and Researc
h Centre, Mahiravani, Nashik.
Presented Paper in ICRTET.

Priyanka Nana Kothawade.

BECOMP, Sandip Institute of Technology and Research Centre, Mahiravani, Nashik.
Presented Paper in ICRTET.

Vibhuti Vasant Vadje.

BECOMP, Sandip Institute of Technology and Research Centre, Mahiravani, Nashik.
Presented Paper in ICRTET.

Priyanka Uday Wani.

BECOMP, Sandip Institute of Technology and Research Centre, Mahiravani, Nashik.
Presented Paper in ICRTET.