A Tool for Clustering-based Aspect Mining through Fan-in Calculation

Guru Prasad Bhandari
Banaras Hindu University
Varanasi, India

Abstract

Since the size and overall complexity of the software system have been increasing, the existing program has to be modularized on the basis of separation of concerns, so aspect oriented programming tries to address the existing program by modularizing the crosscutting functionality into more precise and manageable modules known as aspects. We have proposed a tool Aspect-CAM (Clustering-based Aspect Mining) through fan-in calculation dynamically, which produces the crosscutting concerns or clusters of the existing object-oriented program by fan-in calculation. We have calculated the fan-in matrix of every function with every class and then Euclidean distance matrix among functions to create different crosscutting concerns. Different views: list view, matrix view and including graph view have been presented which shows the new clusters of the existing Object-Oriented Program.

Keywords: Clustering; Code refactoring, Code scattering, Code tangling, Cross-cutting Concerns and Fan-in calculation

1. Introduction

Since the introduction of Aspect Mining in 1990s, it got the high consideration because Aspect Oriented Programming (AOP) has enhanced the maintenance and evolution of software by separating concerns into modules (Danfeng Zhang, 2008). Aspect mining is a reverse engineering process that aims at finding crosscutting concerns in existing systems (Marius Marin, 2007). The goal of aspect mining technique is to discover crosscutting concerns that potentially could be turned into aspects (Edison Klaflke Fillus, 2012). The issues of code scattering and tangling thus of achieving a better modularity for a system’s concerns, are addressed by the paradigm of aspect orientation (Marius Marin, 2007). Aspect Oriented Programming (AOP) is a new programming paradigm that offers a novel modularization unit for the crosscutting concerns (Ceccato, 2006). Functionalities originally spread across several modules and tangled with each other can be factored out into a single, separate unit, called an aspect. Code refactoring (David Binkley, 2006) has to be considered properly while migrating Object-Oriented Program to Aspect-Oriented Program. If we find the aspect of the system then code refactoring is also possible. Code refactoring makes readjustment with the existing system code by removing code scattering and code tangling problems. There are some examples of crosscutting (Kozsik, 2005), which are as follows: Tracing and profiling, Logging, Configuration management, Exception handling, Security, Synchronization, Verifying correctness, Visualization etc.

Different researchers have applied different approaches for aspect mining to find crosscutting concerns. In our work, clustering-based approach has been applied for finding crosscutting concerns and developed a tool to generate different views; text view, matrix view, graph view in different steps while aspect mining. Clustering is a process of partitioning the legacy system into a set of meaningful sub-parts, called clusters. Crosscutting concerns are associated to different parts of the source code. Due to this, they negatively affect maintainability and understandability of the system (Andy Kellens, 2007). Clustering based techniques group methods that are associated to the same crosscutting concerns guided by a distance measure, without requiring any prior knowledge about the system (Edison Klaflke Fillus, 2012).

Clustering-based aspect mining approach can be helpful for enhancing the maintainability and the understandability by removing the unnecessary tangled code or scattered code on the existing system. Thus Aspect-CAM will help to find the crosscutting concerns of the existing program. Traditional program dependency graph just addresses the data and control dependency in aggregate program dependency but when aspect mining came under consideration, it is able to handle tangled or scattered code in the legacy system, which is unique contribution in comparison with traditional dependency graph and existing graph based methods.

www.ijsret.org
The rest of this paper is organized as follows: Section 2 concerns about the related work in the field of aspect mining done by the researchers previously. Basic concepts implemented in our tool Aspect-CAM is given in section 3, which deals with the basic concepts implemented in our tool which includes algorithm used in our tool, steps that are fulfilled while developing Aspect-CAM tool and the work-flow mechanism of the tool. Section 4 provides the description of our tool Aspect-CAM with the help of some important interfaces. Finally, conclusion and future work is given in the section 5.

2. Related Work

Aspect mining is very emerging and challenging area of Software Engineering. Several researchers have proposed different models, techniques, algorithms and tools for identification of crosscutting concerns from the existing procedural or object-oriented program. Some of approaches considered by researchers are mentioned here.

Zhang D. et. al., In their paper (Danfeng Zhang, 2008), have proposed clustering based fan-in analysis(CBFA) to recommend aspect candidates in the form of method clusters. Their approach CBFA uses a new lexical based clustering approach to identify method clusters and rank the clusters using a new ranking metric called cluster fan-in. They have done some experiments on Linux and JHotDraw to provide the recommendations for aspect mining coverage. Marin M. et. al. have also mentioned fan-in analysis approach in their paper (Marius Marin, 2007). They have considered a technique based on determining methods that are called from many different places using fan-in analysis to identify candidate aspects in a number of open-source Java System. Fillus et. al. have also introduced an approach called CAAMPI (Clustering Based Approach for Aspect Mining and Pointcut Identification) (Edison Klafke Fillus, 2012) based on clustering analysis that includes an integrated process to discover aspect candidates and to identify point cuts. They have also presented the experimental results to show the usefulness of the identified pointcuts by using three Java systems as JHotDraw, Apache Tomcat, HSQLDB (HyperSQLDataBase)(HSQLDB, 2014).

Ceccato M. and Binkley D. proposed Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects. In their paper (David Binkley, 2006), an automated approach to the problem of migrating systems developed according to the Object-Oriented Programming (OOP) paradigm into Aspect-Oriented Programming (AOP) has been presented. A simple set of six refactorings has been defined to transform OOP to AOP and has been implemented in the AOP-Migrator tool, an Eclipse plug-in. The have also presented the four case studies to the approach to migrate selected crosscutting concerns and the results show the feasibility of the migration and indicate the importance of the enabling transformations as a pre-processing step.

In the paper (Magiel Bruntink, 2004), Bruntink M. et. al. have evaluated the suitability of clone detection as a technique for identification of crosscutting concerns. They have manually identified for specific concerns in an industrial C application, and analysed to what extent clone detection is capable of finding these concerns. Krinke J. and Breu S. have described an automatic static aspect mining approach in their paper (Jens Krinke, 2004) and revised (Jens Krinke, 2005), where the control flow graph of a program are investigated for recurring execution patterns based on different constraints as the patterns have to exist in different calling contexts. They have also done the case study with the implemented tool that extracts uniform and crosscutting execution relations from both prospective inside and outside execution relations.

Majumdar D. et. al. have proposed a graphical-model-SAG(Symbolic Associatively Graph) (Dipankar Majumdar, 2011), which represents a procedural program and defines a mathematical-model to identify and remove the tangled and interleaving code-fragments. These code fragments are traced back to the requirements model and that identified user requirements are put forward as aspects to be handled or re-engineered. Serban G. and Moldovan G.S. have also presented a new graph-based approach in aspect mining in their paper(Gabriela Serban, 2006). They have defined the problem of identifying the crosscutting concerns as a search problem in a graph and introduced GAAM algorithm (Graph Algorithm in Aspect Mining) for solving this problem. They have also evaluated the result obtained by GAAM algorithm from the aspect mining point of view.

This present work extends the above contributions further by developing a tool Aspect-CAM and adding some new features such as list view of fan-in calculation of every function, matrix of functional dependency, matrix and list view of class to function dependency, Euclidian distance matrix to find the weight of functional dependency. Graph view of crosscutting concern is also presented in our tool. The main propose of this tool is to help maintainer to visualize the aspects on the basis of separation of concerns that could help easy debugging of the program and code refactoring.
3. Basic Concepts implemented in Aspect-CAM

In this paper, we have considered program code slicing, fan-in calculation and Euclidian distance for clustering-based approach to find crosscutting concerns for aspect mining. To get this, we have implemented it in C#.Net programming language and GraphViz tool has been used to display the graph view of crosscutting concerns into the browser.

3.1 Work-flow of Aspect-OOP tool

Source code has been taken as input and finally it produces the clustered methods. During that process, there are some steps which are as follows, shown in figure 1.

- **Source Code**: Before performing any other activity our tool takes source code as input. Source code may be any C#.Net complete project code. It also maintains the repository for the already analysed project.
- **Code Slicing**: The technique to find the fan-in value is code slicing on the basis of class and function. Pattern matching of the regular expression has been used to extract the functions and classes of the program.
- **Fan-in Calculation**: The number of calls of every function is calculated here. Fan-in implies the number of incoming call to that function from any other functions. For example: A, B, C and D are the functions where B→A (B calls A), C→A and B→D then in that case, fan-in value of function A is two because function B and function C are calling function A. In the same way, fan-in value of D is one.
- **Fan-in Matrix**: After calculating the fan-in value of every function. The matrix representation of function call has been presented into GridView control in which row represents the function and column represents the class. Thus if depicts the relationships between class to function means which class is having the function call.
- **Euclidian Distance matrix**: To find the distance how far one function is from another we used Euclidian distance formula. Relationship between one to every other functions are presented in Euclidian distance matrix.
- **Clustered methods**: After filtering and analysing the code, call site will be declared to show the crosscutting concern of the existing program. The method which has the highest value selected as call site and other some functions which has less value than threshold value, comes with that call site function. Now one cluster is created from the function having the highest value. And the same step is applied for rest of the functions which are not included to the call site. The clusters of methods are shown in the graph as well by our tool.

There are mainly two steps from existing procedural software system or object oriented software system to the aspect oriented software system without any code tangle or code scatter.

- Aspect mining to identify potential aspects in legacy software
- Refactoring to encapsulate these aspect candidates into aspect-oriented (AO) code.

3.2 Aspect-mining Algorithm:

Czibula et. al. have proposed a new hierarchical agglomerative clustering algorithm in aspect mining (HAC)(Gabriela Czibula, 2010). We have implemented this algorithm partially to find the crosscutting concerns in our tool Aspect-CAM. This algorithm is used in the Grouping step of the crosscutting concerns identification process. In their approach, the objects to be clustered are the methods from the software system, X = \{m1, m2, . . . , mn\}. The methods belong to the application classes or are called from the application classes.

Based on the vector space model, They consider each method as an l-dimensional vector: \( m_i = \{m_{i1}, m_{i2}, \ldots, m_{il}\} \). They have considered two vector-space models.
1. A method \( m \) is characterized by a 2-dimensional vector \( \{ \text{FIV}, \text{CC} \} \), where \( \text{FIV} \) is the value of the fan-in metric and \( \text{CC} \) is the number of calling classes. They denote this model by \( \mu_1 \).

2. A method \( m \) is characterized by an \( l \)-dimensional vector \( \{ \text{FIV}, B_1, B_2, \ldots, B_{l-1} \} \), where \( l-1 \) is the number of classes from the software system \( S \) (called application classes), \( \text{FIV} \) is the value of the fan-in metric and \( B_i \) is the value of the attribute corresponding to the application class \( A_{Ci} \) \((1 \leq i \leq l - 1)\), as follows:

\[
\text{Bi} = \begin{cases} 
1 & \text{if } m \text{ is called from at least one method belonging to application class } A_{Ci} \\
0 & \text{otherwise}
\end{cases}
\]

This model is denoted by \( \mu_2 \).

In their approach they consider that the distance between two methods \( m_i \) and \( m_j \) is expressed using the Euclidian distance, as:

\[
d_Z(m_i, m_j) = \sqrt{\sum_{k=1}^{l}(m_{ik} - m_{jk})^2}
\]

They have chosen Euclidian distance as distance metrics for expressing the dissimilarity between two methods. According their experiment, they concluded that Euclidian distance is the most appropriate for clustering based aspect mining.

### 3.3 Steps for extraction of Crosscutting-concerns

The main idea of hierarchical agglomerative clustering algorithm in aspect mining (HAC)(Gabriela Czibula, 2010) heuristic for choosing the number \( k \) of clusters is to determine \( k \) representative methods (called medoids) from \( M \), performing the following steps:

(i) The initial number \( k \) of clusters is \( n \) (the number of methods from the system).

(ii) The method chosen as the first medoid is the most “distant” method from the set of all methods (the method that maximizes the sum of distances from all other methods).

(iii) For each remaining methods (that were not chosen as medoids), we compute the minimum distance (dmin) from the method and the already chosen medoids. The next medoid is chosen as the method \( m \) that maximizes dmin and this distance is greater than a given positive threshold (distMin). If such a method does not exist it means that \( m \) is very close to its nearest medoid and should not be chosen as a new medoid (from the aspect mining point of view \( m \) and \( nc \) should belong to the same (crosscutting) concern). In this case, the number \( k \) of clusters will be decreased.

(iv) The step (iii) will be repeatedly performed, until \( k \) medoids will be reached.

### 4. Description of Aspect-CAM Tool

In our tool Aspect-CAM, user is required to import any C#.Net project before any other operation. Figure 2 depicts the project import interface where user can have two options- either can import project code from the already existing project from already analysed project collection repository or import the fresh project which has not been analysed before. The absolute path will be shown in path text box after importing the project by clicking ‘Select Project’ button. Project name will be shown in the left centre of this form in large size green text. List of files that are included in the imported project are shown in left side list box. After clicking the ‘load’ button, the complete concatenated project code is shown in the right side rich box control.
Figure 2 Project Importing Interface

Figure 3 shows the list of fan-ins of functions. Left side list box control presents the list of occurrences of function calls in the format of Called Class. Function ---->Calling Class.Function as shown in the label control, after clicking the button ‘Show List’. For example: get_max function of main Class is called by term_Load function defined in term class. ‘Show Fanin’ button click event has been used to show the fan-in values of all the invoked functions as presented in the centerlistbox in figure 3. For example: insert_record(fanin=7) means insert_record function has been called seven times by different functions. Right side tree view control has been used to show all the classes with their corresponding functions of the complete C#.Net project. ‘Show Matrix’ button, shown in the form switches the form into matrix view form.
Figure 4 depicts the two matrixes, first one is corresponding to fan-in of every function with every class, denotes whether the function presented in column has been called by the class presented in row or not. Here ‘0’ means there is no occurrence of call of the corresponding function by the corresponding class whereas ‘1’ means there is function call- row class calls to the corresponding function at least once. Matrix value is calculated by applying the abovementioned aspect mining algorithm and here first matrix shown in gridview control in figure 4 shows the value of Bi of the algorithm.

\[
B_i = \begin{cases} 
1 & \text{if } m \text{ is called from at least one method belonging to application class AC} \\
0 & \text{otherwise}
\end{cases}
\]

Second matrix concerns about the generation of Euclidean distance among the function on the basis of value of Bi (‘0’ or ‘1’). The formula for generating Euclidean distance has been presented in the abovementioned aspect mining algorithm.

\[
d_e(m_i, m_j) = \sqrt{\sum_{k=1}^{n} (m_{ik} - m_{jk})^2}
\]

This tool is able to calculate Euclidean distance among functions in matrix form automatically as shown in the bottom right of this form as after clicking the button ‘Show EucDistance’. Generation of clusters are possible after generating Euclidean distance of every function. User can generate the clusters by just clicking on the button ‘Generate Clusters’, shown in figure 4. After generation clusters, user can have the graph view of that by just clicking the button ‘Show Graph’.

![Figure 4 Object-Oriented Program Dependency](image)

We have used GraphViz tool to display the clusters or the crosscutting concerns in the browser. Sample clusters are represented the figure 5 of our example C#.Net project code. The whole program has been partitioned on the basis of Euclidean distance among function by checking with threshold value. Here we have declared threshold value ‘1’. The functions with the highest Euclidean distance create a call site first and it combines all other functions with it if any function has Euclidean distance less that threshold value. Then after, the Euclidean distance matrix has to be regenerated dynamically for the rest of the functions. Next, same process is applied for the rest of the functions as abovementioned algorithm.
5. Conclusion and Future Work

Aspect mining is based on some essential feature of the any existing program code. There are many approaches to identify the potential aspects. In our approach, clustering-based approach has been used to find crosscutting concerns of object-oriented program. C#.Net programming language has been used to develop our tool Aspect-CAM. As our experiment tested on 5K lines of code of C#.Net project, our tool is able to find the crosscutting concerns. For that, fan-in calculation technique, program slicing method and expression pattern extraction technique have been applied. Through this tool, we tried to enhance the already contributed tools which have been developed by other researchers in the field of aspect mining by making automatic clusters from the source code. GraphViz tool has been applied for demonstrating purpose of different clusters or aspects retained in the object oriented program code after extraction into the graph view. Designing and developing tool to help reverse engineering will always be under our consideration. As future work, we are trying to consider component-based program and service-oriented software system to extract the crosscutting concerns by enhancing already existing feature in our current tool or by adding some extra new features.

References


