Improve Quality of Code using Aspect Mining

Prof. Amruta A. Pundlik¹ Bharat A. Kadam² Tejashree S. Jagtap³ Priyanka Nerkar⁴
¹Lecturer ²,³,⁴Department of Computer Engineering

Abstract—Specification Oriented Programming (AOP) is a new programming paradigm that offers a novel modularization unit for the crosscutting concerns. AOP was originally proposed for the development of new software, systems written using traditional modularization techniques may also benefit from the adoption of the more versatile decomposition offered by AOP, in terms of code understandability and Evolvability. Abstract Specification mining is a process that tries to identify crosscutting concerns in existing software systems. The goal is to refactor refine the internal structure of the software to make it easier to understand and cheaper to modify without changing its observable behaviour. Refactoring is systematically organized into catalogues, in a similar way as design patterns. Hints that certain refactoring’s are applicable are provided by so-called code smells: suspicious code parts that require improvement.

Key words: Specification Oriented Programming (AOP), object oriented programming, Detecting Unique Methods

I. INTRODUCTION

In all engineering disciplines, humans scope with complex systems by using a “divide and conquer” approach: they divide a complex problem into many simpler sub-problems. Each sub-problem object should have a low level of coupling with the others[1][3][4], so the team assigned to this sub-problem can reason about it as a whole and can devise a solution for it. The composition of the sub-problem solutions produces the solution for the whole complex problem. In software engineering, many modularization mechanisms[4] have been introduced. For example object oriented programming (OOP) provides the object model. This model usually is a natural decomposition of the domain, because it tends to parse systems down into units of data and behaviour that correspond to real world entities[6]. Whatever decomposition type is used, it becomes the major architecture of the system under development, and all future changes will refer to it. Software systems are so complex that they cannot be developed without dividing them in this sub-modules. The main drawback in this approach is that there are some system functionalities that cannot be assigned to a single module in the system decomposition[1].

The main aim of project is to find the Cross Cutting in the program as well as optimize the project space and time complexity, also provide suggestion for the new programmer (i.e. Refactoring) Specification mining is a relatively new research domain whose aim is to study and to develop techniques and tools for automatic identification of cross-cutting concerns in legacy systems. The goal is to refactor them to aspect. A manual approach for crosscutting see discovery is difficult and error prone due to the size of the software system[1], its complexity and lack of documentation[5][6]. A refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior[4]. Refactoring’s are systematically organized into catalogues, in a similar way as design patterns. Hints that certain refactoring’s are applicable are provided by so-called code smells: suspicious code parts that require improvement. In this project we aim to evaluate the use of clustering for crosscutting concerns identification as clustering tries to identify groups of similar objects[7][8], and crosscutting in the concerns in legacy systems can be viewed as such groups of methods and using refactor abstract change the structure of system to make it easy to understand. An existing OOP application containing bad modularization can be improved by adopting AOP[1][2][7].

II. APPROACH

Software System Mining Software mining techniques aim at ending valuable information in the source code of a software system[5][6][7][8], in order to make this information explicitly available to software engineers involved in the evolution of that system[4]. A software mining example of aspect mining is business rule extraction. Software system mining is supported by software exploration[8]. This typically involves three steps: data collection from main code, knowledge inference based on abstraction from the collected data, and information presentation using, for example, hypertext and visualizations. In this paper[7], we provide an initial assessment of how software exploration techniques can help in specification mining. Refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour[1][3][4]. Refactoring’s are systematically organized into catalogues, in a similar way as design patterns. Hints that certain refactoring are applicable are provided by so-called code smells: suspicious code parts that require improvement. Position Statement In this paper, we argue for the need for researching in the area of specification mining, covering concepts [3-4], principles, methods and tools supporting the identification of specifications in object-oriented software systems as well as the subsequent refactoring of such systems into specification-oriented systems.

A. Features of Proposed System:
Software System Mining Software mining techniques aim at ending valuable information in the source code of a software system[5][6][7][8], in order to make this information explicitly available to software engineers involved in the evolution of that system[4]. A software mining example of aspect mining is business rule extraction. Software system...
mining is supported by software exploration[8]. This typically involves three steps: data collection from main code, knowledge inferencing based on abstraction from the collected data, and information presentation using, for example, hypertext and visualizations. In this paper[7], we provide an initial assessment of how software exploration techniques can help in specification mining. Refactoring A refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour [1][3][4]. Refactoring’s are systematically organized into catalogues, in a similar way as design patterns. Hints that certain refactoring are applicable are provided by so-called code smells: suspicious code parts that require improvement. Position Statement In this paper, we argument for the need for researching in the area of specification mining, covering concepts [3-4], principles, methods and tools supporting the identification of specifications in object-oriented software systems as well as the subsequent refactoring of such systems into specification-oriented systems.

B. Natural Language Processing on Source Code;

Similar to the previous approach, Shepherd et al. propose a technique that is based on the assumption that cross-cutting concerns are often implemented by the rigorous use of naming and coding conventions. Their approach uses natural language processing (NLP) information as an indicator for possible specification candidates[3]. They report on an experiment in which they use an NLP technique called lexical chaining in order to find groups of related source-code entities which represent a cross-cutting concern [2-3]. Lexical chaining will output, given a collection of words as input, chains of words which are semantically strongly related. In order to create the chains, the algorithm requires a semantically distance measure between each combination of words. To this end, Shepherd et al. used the WordNet database, in combination with information about the parts of speech of each word, to calculate the semantically path between two words. In order to mine for cross-cutting concerns [1], they apply the chaining algorithm to the comments, member function names, variable names and class names of the system they are analysing. A user of their approach needs to manually inspect the resulting chains in order to select likely specification candidates.

C. Detecting Unique Methods;

In pre-AOP days, cross-cutting concerns were often implemented in an idiomatic ways. Certain of these idioms can be regarded as “symptoms” of aspect candidates. An example of such an idiom is the implementation of a cross-cutting concern by means of a single entity in the system which is called from numerous places in the code (for instance, a ‘logging’ entity which is called from throughout the code). To detect instances of this pattern[3], Gybel and Kellen’s propose the “Methods” heuristic which is defined as: “A method without a return value which implements a message implemented by no other method.” After summation all main methods in a system, sorting them according to the number of times a method is called, and filtering out irrelevant methods (like for instance accessor and mutator methods)[2], the user has to manually inspect the resulting methods in order to find suitable specification candidates. Regardless of the simplicity of this approach, the authors demonstrated the applicability of their technique by detecting typical specifications like tracing, uploaded notification and main memory management in the context of a Smalltalk image.

D. Clustering Of Related Methods;

Hierarchical clustering of similar method names Shepherd and Pollock [7] report on an experiment in which they used agglomerative hierarchical clustering [8] to group related methods. This technique starts by putting each method in a separate cluster and then recursively merges clusters for which the distance between the methods is smaller than each threshold. They developed this technique as part of a specification-oriented IDE named AMAV (Specification Miner and Viewer), which allows for easy adaptation of the distance measure used by the algorithm. For an initial experiment they used a simple distance measure opposite proportional to the common substring length of the names of the methods. This mining algorithm is used in combination with the viewing tool of the IDE which not only lists all the clusters which were found, but also consists out of a crosscutting pane which displays the methods related to a cluster as well as an editor window, in which the class code of a fix method is displayed[3].

E. Clustering Based On Method Invocation:

Specification mining technique based on cluster analysis. They start from the assumption that if the same methods are called frequently from within different modules, this may be a good indication with each other that a hidden cross-cutting concern is present. As input for the aspect clustering algorithm, a set of methods is given along with a distance measure based on the Static Direct Invocation Relationship (SDIR) between the methods. This distance measure varies between 0 and 1 and either in natural language or in computer languages, according to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech). The term has slightly different meanings in different branches of linguistics and computer science. Traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence, sometimes with the aid of devices such as sentence diagrams. It usually emphasizes the importance of grammatical divisions such as subject and predicate. Parsing was formerly central to the teaching of grammar throughout the English-speaking world, and widely regarded as basic to the use and understanding of written language. However the teaching of such techniques is no longer current, will have distance approximating 0, while the distance between methods which are never or seldom called together will be close to 1. represents the dissimilarity of the methods.

Methods which are closely related (i.e. which get called frequently together).

III. Module

A. Error Finding (Parser):

Parsing or syntactic analysis is the process of analyzing a string of symbols,

B. Documentation:

Documentation module in this system provides all the required documents related code been refactor. E.g. if the
system identifies that the code written is bubble sort system will provide the documentation help related that code.

C. Code Optimization:
System will optimize the code size, code time complexity and space complexity. This is done by removing unused code, repeated code and replacing code with better code.

D. Refactoring:
Refactoring means process of changing an internal structure of software system without changing its function or behaviour.

E. Visual Representation:
Graphical representation can be represented using Pie-Charts. The Project will consider to Pie-charts, one for the original code before reduction and second one for the code which is reduced both the pie-charts are compared for the time complexity and space complexity which will show the difference in both.

F. Auto Repair:
Our System will highlight the errors and will automatically repair the minor errors in the program.

IV. SYSTEM ARCHITECTURE

![Diagram of System Architecture]

![Java Code Flowchart]

V. ALGORITHMS

A. KAM Algorithm for Specification Mining (KAM):
In order to avoid the two main disadvantages of the traditional k-means approach, we propose a new k-means based clustering algorithm[1], KAM (k-means in Specification Mining), that uses a heuristic for choosing the number of clusters and the initial centroids [3]. This heuristic is particular to specification mining and it will provide a good enough choice of the initial centroids [1].

After selecting the initial centroids, KAM behaves like the classical k-means algorithm. The main idea of KAM’s heuristic for choosing the initial centroids and the number k of clusters is the following:

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

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

3) For each remaining methods (that were not chosen as centroids), we compute the minimum distance (dmin) from the method and the already chosen centroids. The next centroid is chosen as the method m that maximizes dmin and this distance is greater than a positive given threshold (distMin). If such a method does not exist it means that m is very close to its nearest centroid nc and should not be chosen as a new centroid (from the specification 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.

4) The step (iii) will be repeatedly performed, until k centroids will be reached. We have to notice that step[3] (iii) described above assures, from the specification mining point of view, that near methods (with respect to the given threshold distMin) will be merged in a single (crosscutting) concern, instead of being distributed in different (crosscutting) concern. We mention that at steps (ii) and (iii) the choice could be a non-deterministic one. In the current version of kAM algorithm[4], if such a non-deterministic case exists, the first selection is chosen. Improvements of kAM algorithm can tackle these kind of situations. KAM algorithm is presented in Algorithm 1. We mention that the algorithm stops when the clusters from two consecutive iterations remain unchanged, or the number of steps performed exceeds the maximum number of iterations allowed.

B. KAM Algorithm Mathematical Model:

1) Input:

\[ \text{Input:} \]

- \( \text{set} M = \{m_1, \ldots, m_n\} \) of methods to be clustered
- the metric \( d_E \) between methods in a multidimensional space
- distMin \( > 0 \) the threshold for merging the clusters
- noMaxIter the maximum number of iterations allowed

Output:

- \( K = \{K_1, \ldots, K_p\} \) the partition of methods in \( M \)
- KAM

Algorithm Mathematical Model:

\[ \text{Algorithm Mathematical Model:} \]

1) **Input:**

\[ \text{Input:} \]

- \( \text{set} M = \{m_1, \ldots, m_n\} \) of methods to be clustered
- the metric \( d_E \) between methods in a multidimensional space
- distMin \( > 0 \) the threshold for merging the clusters
- noMaxIter the maximum number of iterations allowed

Output:

- \( K = \{K_1, \ldots, K_p\} \) the partition of methods in \( M \)
- KAM

Algorithm Mathematical Model:

\[ \text{Algorithm KAM is:} \]

1. \( k \gets n \) // the initial number of clusters
2. \( i \arg \max_{i=1,n} \{ P \} \)
3. \( j \arg \max_{j=1,J} \{ d_E(m_i, m_j) \} \) // the index i1 of the first centroid is chosen
4. \( n_r \gets 1 \) // the number of centroids already chosen
5. \( D \gets \{ j \mid 1 \leq j \leq n, j \neq \text{index of centroid} \} \)
6. \( \forall \) centroids in \( D \) do
7. \( d \gets \min \{ d_E(m_i, m_j) \} \)
8. \( \text{if} \ d < \text{distMin} \) then
9. \( k \gets k - 1 \) // the number of clusters is decreased
10. \( n_r \gets n_r + 1 \) // another centroid is chosen
11. \( \text{end if} \)
12. \( \text{end for} \)
13. \( f_i \gets m_i \) // the centroids are initialized
14. \( \text{end for} \)
while (K changes between two consecutive steps) and (there were not performed noMaxIter iterations)
do
  for j ← 1, k do
    Kj← {mi | 8 1_ r_k, r 6= j, dE(mi, fj) _dE(mi, fr)}
fj← the mean of objects in Kj
  // the j-th centroid is recalculated
Endfor endwhile.

VI. CONCLUSION

Specification mining research is concerned with the development of concepts, principles, methods and tools supporting the identification of specifications in object-oriented software systems as well as the subsequent refactoring of such systems into specification oriented systems. In this paper, we explored the state of the art in specification mining research, and we identified a series of promising research directions. Research results can be used to support software development, continuously analyzing a system while it is built in order to identify (1) code smells requiring specifications; and (2) specification smells requiring (specification specific) refactorings. Moreover, specification mining can be used on completed systems, in order to offer better support for future evolution of the system.

REFERENCES