Volume 5, Issue 1, January 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Aspect Mining Techniques for Legacy Code Developed in Procedural Languages

Yasmin Shaikh[*]
Internatonal Institute of Professional Studies,
Devi Ahilya University, Indore (M. P.) India

Sanjay Tanwani
School of Computer Science & IT
Devi Ahilya University, Indore (M.P.) India

***Abstract:** In the development of a software system, the design process allows the system to be broken into subsystems and programming languages allow defining the subsystems at an abstract level. Further, the programming languages provide mechanism to compose the abstract subsystems into overall system. It is desired to define single functionality in each subsystem. But some functionality always cut-across the sub-units. This kind of functionality is called crosscutting concern or aspect. Aspect Oriented Programming overcomes the problem of code scattering by having stand-alone modules called aspects for such crosscutting functionality. The main functionality is programmed as components and aspects are weaved separately as and when required with the main functionality.*

Existing legacy systems can be refactored into aspect oriented ones. Such refactoring of legacy system involves identification of aspects in the legacy code and then transforming legacy code base into equivalent aspect oriented code base. Aspect mining techniques are used to identify the aspects from the legacy code. Much of the research in Aspect mining is focused on mining aspects from Object Oriented Programming language code. Even today, a large number of software are written in procedural languages and are used by various organizations. In view of advantages provided by Aspect Oriented Programming for maintainability and extensibility of software, refactoring of such code in aspect oriented code can be proved to be very useful. But not much of the research is done to mine aspects from such systems. In this paper, a novel technique for mining candidate aspects from COBOL code is proposed. A study of legacy code written in COBOL that has been revised number of times is presented in this paper. On the basis of information available about the different versions of programs and the history of changes made into the software over the period of time, a transaction database is constructed. This database includes the time stamp, author and the description of change. The database is analyzed for the modules that have been changed frequently along with the sequential patterns of modules that have been changed over the time are mined. This analysis identified the modules and sequential patterns that have been maintained frequently. Thus, the code requiring most of the maintenance effort is identified. Also, the sequential patterns localized the code for candidate aspects thereby reducing the efforts of aspect mining.

***Keywords**— Aspect mining, Cross-cutting concern, Maintainability, Mining COBOL, Code Localization*

I. INTRODUCTION

Some programming tasks cannot be neatly encapsulated in objects, but must be scattered throughout the code such as logging, profiling, tracing, session tracking, and security management. The goal of aspect oriented programming is to identify such crosscutting concerns and program them as separate modules called Aspects. These are developed and maintained separately from core components. Aspects are weaved with components at build time or runtime. Weaving is the process of combining (or assembling) aspects and components to create the desired runtime behavior. Aspect oriented programming complements rather than replaces object oriented programming. Object oriented programming modularizes hierarchical structure and aspect oriented programming modularizes crosscutting concerns.

Aspect oriented programming offers several advantages. First, it allows modularized implementation of crosscutting concerns. In case of conventional programming languages the cross-cutting concerns are scattered through different modules. For example, if there are four modules having system level requirements like security, logging and authentication that crosscut many core module level requirements. In conventional programming language, each of the four core modules have code for required concern. But with aspect oriented programming, core modules contain code for actual core concerns and the security, logging and authentication concerns are programmed as aspects. These aspects are then weaved with core modules as required. Second, it is easier to evolve a system with aspect oriented programming. If a new feature is to be added, in the existing system, which involve crosscutting concerns; with aspect oriented programming we need to write module only for new feature that can be then weaved with existing core modules and aspects for cross-cutting concerns. Third, late binding of design decisions can be done. The modules that were not coupled during the design phase may be coupled at any later stage as the system has independent modules and aspects (without any cross-cutting functionality in core modules). Fourth, more code reuse is possible. The core modules do not have any cross-cutting functionality so they can be reused wherever the corresponding core functionality is required in a system.

Aspect mining research is concerned with the development of concepts, principles, methods and tools supporting the identification of aspects in existing software systems as well as the subsequent refactoring of such systems into aspect-oriented systems. Aspect mining is defined as a specialized reverse engineering process, which aims at investigating legacy systems (source code) in order to discover which parts of the system can be crosscutting concern. Aspect mining search for candidate aspects in existing systems and isolate them from the system into separately described aspects. The principal symptom of existence of an aspect is code duplication and other is code scattering. A number of aspect mining techniques have already been proposed. These techniques are based mainly on three approaches namely, Identifier Analysis, Fan-in Analysis and Dynamic Analysis.

II. WHY COBOL?

With the evolution of programming languages now software can be developed using new programming paradigms such as Object Oriented Programming and Aspect Oriented Programming. The question arises that why the research should be done on COBOL language. The answer to the question is that billions of lines of business COBOL code is in use worldwide. This existing code need be maintained continuously. Also, millions of lines of COBOL code are still written every year. The main advantage offered by Aspect Oriented Programming is easy code comprehension and better modularity. Both these features increase maintainability of code. So, research on mining aspects from COBOL code and then refactoring this legacy code to aspect oriented one is extremely significant in present business data processing scenario. Ralf Lämmel [15] presented the potential of COBOL as research theme. He presented his findings on COBOL that reveal following facts:

- 75% of all business data is processed in COBOL.
- There are 180 to 200 billion lines of COBOL code is in use worldwide.

Also, in mathematical terms, amount of research dedicated to a programming language is proportional to the usage of language in reality.

III. RELATED WORK

Various research attempts were made for identifying aspects in procedural language code such as C and COBOL code. W. Opdyke[20], has made pioneer contribution to refactor legacy code. Several aspects have been identified in C language code such macros, code clones, and error handling code [21].

COBOL already has features of aspect oriented programming language. The USE statement in procedure division of a COBOL program is clearly an aspect [16]. An aspect oriented approach to understand legacy COBOL code is presented in [19]. Error handling aspect is mainly identified as most important aspect in this paper.

Aspect mining techniques have also been applied to identify aspects from COBOL code. In [18], program slicing technique is used to identify the parts of program that might affect or be affected by the values computed at some point of interest.

IV. ASPECTS IN COBOL CODE

Logging and tracing, synchronization, automated regression testing, are some aspects in COBOL that have been identified earlier. In our study we found two new candidate aspects - Report generation code and code for Debugging. Most reports have same structure- report heading, footing and columns of data. Automatic report generation tools are used to generate report. The tool requires format of report as input and generates report. The report generation code is embedded in source code in different modules as and when required. The report generation code scattered in the software has duplicated code. The maintenance of such code requires a lot of effort.

V. ISSUES IN MINING ASPECTS FROM COBOL CODE

Most of the aspect mining techniques that have been developed so far focuses on mining aspects from object oriented programming source code. Not much of the work has been done on mining aspects from COBOL. Identification of aspects from code firstly requires understanding of code. To understand source code, analysis of data flow and control structures of program is required. A number of data analysis tools and techniques exist for program understanding. Secondly, candidate aspects need to be identified. Different aspect mining techniques specifically designed for mining aspects from COBOL need to be developed in order to mine aspects from COBOL code.

VI. CASE STUDY

Legacy software written in COBOL for processing the results of students is studied to identify the possible candidate aspects. The source code included 20K lines of code. In the study it was found that the 64% of the source code is dedicated to report generation. Also, since the inception of software the report generation code and marking scheme code has been changed the most. A change in marking scheme leads to change in graphical output of marks. In the present system if there is a change in scheme of marks then all source code containing report generation logic for various reports need to be changed. The code is scattered in different modules to generate different types of reports. Code clones increase maintenance efforts. If the report generation code and the code for calculation of marks based on scheme of marks are made separate concerns i.e. aspects then if there is a change in scheme of marks, one needs to make change in report generation aspect and scheme calculation aspect only. This would reduce maintenance efforts significantly.

VII. Aspect Mining Techniques for COBOL

Clone detection techniques is used to identify the report generation code and debugging code. Report generation code written in a system for generating different reports have similar type of code or similar characteristic. A hybrid technique, as suggested by Bruntink which combine AST based clone detection with clone detection tool based on tokenized representations of source code [22] is used to identify the similar report generation code in different modules of the system. This technique uses parsers to obtain a syntactical representation of the source code, typically an abstract syntax tree (AST). The clone detection algorithm then search for similar sub trees in this AST. The clone code found by the algorithm is an aspect.

Token based clone detection techniques is used for identifying the aspect involving debugging aspect. Token based clone detection techniques apply lexical analysis (tokenization) to the source code and subsequently use tokens as a basis for clone detection. Lexical analysis is usually initiated by the user specifying a seed of information (either a regular expression or a string). Lexical search simply searches for duplicates of the seed. The debugging code in COBOL source code contains two seeds READY TRACE and DISPLAY.These seeds are given for lexical analysis and duplicates are searched in the code.

VIII. Concluding Remarks

In this paper we have presented our ideas on candidate aspects in COBOL code. We have identified the code that may be candidate aspects while refactoring the legacy COBOL code to aspect oriented one. Two new aspects in COBOL code namely report generation aspect and debugging aspect have been reported in the paper. We have also presented suitable aspect mining techniques to mine identified aspects from legacy COBOL code. Although the aspect mining techniques for object oriented languages are well understood and developed; there is a lot of scope in developing aspect mining techniques for COBOL because of large COBOL code base existing even today for business applications. If serious efforts are made to identify aspects from COBOL code it would help in refactoring of legacy code base into aspect oriented ones. The advantages of aspect oriented programming such as easy code comprehension and maintenance are well known. Billions of lines of COBOL code is maintained throughout the world. Refactoring of existing code into aspect oriented one will definitely reduce the maintenance efforts by significant percent

IX. Future Enhancements

We have identified aspects from COBOL code and suggested aspect mining techniques to mine these aspects from code. A dedicated tool to mine these aspects from code base may be developed. Existing COBOL applications can be refactored into aspect-oriented ones leading to better software evolution and maintainability

References

[1] A.. Deursen, M. Marin, and L. Moonen, "Aspect mining and Refactoring", *In Proc. of the First Int. Workshop on REFactoring: Achievements, Challenges, Effects (REFACE03)*, 2003.

[2] B. Nora, G. Said, and A. Fadila, "A Comparative Classification of Aspect Mining Approaches", *Journal of Computer Science* vol.2 no. 4, pp- 322-325, 2006.

[3] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonello, T. Tourw´e, "A qualitative comparison of three aspect mining techniques" In *International Workshop on Program Comprehension (IWPC 2005), IEEE Computer Society Press*, pp. 13-22, 2005.

[4] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé, " Applying and Combining Three Different Aspect Mining Techniques. *Software Quality Journal*, vol.14, no. 3, pp. 209-231, 2006

[5] G. S. Cocojar, G. Serbian, "On some Criteria for Comparing Aspect mining Techniques", In *Proceedings of the 3rd workshop on Linking aspect technology and evolution (LATE)*, ACM, 2007.

[6] E. S. Abait, S. A. Vidal, C. A. Marcos, "Dynamic analysis and association rules for aspects identification", In *II Latin American Workshop on Aspect-Oriented Software Development (22nd Brazilian Symposium on Software Engineering). Institute of Computing - UNICAMP*, pp. 31-39, 2008.

[7] M. Marin, L. Moonen and A.V. Deursen, "A common framework for aspect mining based on crosscutting concern sorts", *Delft University of Technology Software Engineering Research Group Technical Report Series, Report TUD-SERG*-2006-09

[8] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-Oriented Programming", *Proc. 11th European Conference on Object-oriented Programming*, vol. 1241 of Lecture Notes in Computer Science, pp. 220–242. Springer, 1997.

[9] K. Mens, A. Kellens, J. Krinke, "Pitfalls in Aspect Mining", *Proc. 15th IEEE Working Conference on Reverse Engineering WCRE'08*, pp. 113-122, 2008.

[10] Y. Sakamoto, H. Sato, M. Kurihara, "A Study on Online Aspect Mining", In *proceedings of the International MultiConference of Engineers and Computer Scientists*, Vol I, IMECS 2010

[11] A. Kellens, K. Mens, P. Tonella, "A survey of automated code-level aspect mining techniques" In the *Transactions on aspect-oriented software development IV, SECTION: Aspects and software evolution*. pp: 143-162, 2007.

[12] D. Shepherd, E. Gibson, and L. Pollock, "Design and Evaluation of an Automated Aspect Mining Tool", In *International Conference on Software Engineering Research and Practice* , 2004.

[13] M. Bruntink, "Aspect mining using clone class metrics", In *1st Workshop on Aspect Reverse Engineering*, 2004.

[14] C. Zhang and H. Jacobsen, A Prism for Research in Software Modularization Through Aspect Mining, *IEEE Transactions Parallel and Distributed Systems*.
[15] Ralf Lämmel. Cobol as research Theme.
[16] R. Lämmel and K. De Schutter, “What does Aspect Oriented Programming mean to Cobol?”, In *Proceedings of Aspect-Oriented Software Development (AOSD 2005),* ACM Press, 2005.
[17] An Aspect Oriented COBOL. Report AOSD.08
[18] H. Shinomi, and Y. Ichimori, “Program Analysis Environment for Writing COBOL Aspects”.
[19] J. Pu, Z. Zhang, J. Kang, Y. Xu and H.i Yang, “Using Aspect Orientation in understanding legacy COBOL code”.
[20] William Opdyke. Refactoring Object Oriented Frameworks. 2002.
[21] S.A.M. Rizvi, Z. Khanam. Introduction to Aspect Oriented Techniques for Refactoring Legacy Software.
[22] B. Magiel, , A.V. Deursen, R.V. Engelen and T. Tourw´e, “An evaluation of clone detection techniques for identifying crosscutting concerns. In *Proc. Intl. Conf. Software Maintenance (ICSM).IEEE Computer Society*, 2004.