Identification of Crosscutting Concerns: A Survey

Arvinder Kaur
University School of IT
GGSIP, University, Delhi
arvinder70@gmail.com

Kalpana Johari
CDAC, Noida
kalpanajohari@cdacnoida.in

Abstract

Modularization of concerns is important for software development. Object oriented programming paradigm provides an ease of modularization of basic concerns. There are some concerns whose implementation cannot be modularized using object oriented paradigm like profiling, logging etc. The implementation of such concerns remains scattered throughout the source code. Such concerns are called crosscutting concerns. Identification of crosscutting concerns plays an important role in aspect mining, defect detection and software maintenance. This paper gives a survey on the techniques used for identifying crosscutting concerns. We have compared eight different techniques. The five of the eight techniques were applied on a tutorial application. The comparison has been done on five parameters like input required, output generated, availability of tools, stage of software development where the technique is applicable, the software on which the technique has already been applied.

Keywords- crosscutting concerns; aspect mining; scattered; modularization; object oriented

INTRODUCTION

Understanding of concerns plays an important role in successful software development. The meaning and the concept of concerns have been understood in different manner by number of researchers. M.P. Robillard et al describe concern as “any consideration about the implementation of a program”. S.M. Sutton et al defines it as “any matter of interest in a software system”. According to J. Lamping concern is “any coherent issue in the problem domain” In all the definition about concern, one thing emerges as a common factor, is that, it requires consideration that is showing concern about concern. From software development perspective concern may represent any feature, functional or non functional which is important for stakeholders. A requirement in requirement specification document represents a concern[15]. The separation of concerns has been emphasized by software engineering principles for better understanding of concerns. Object oriented paradigm supports the modularization of concern. Improper implementation of the feature and the limitations of programming language construct leads to scattered and tangled implementation of the concern[16]. Every concern might not get modularized into a separate module. Such concerns whose implementation is scattered over more than one module are called crosscutting concerns. Such concerns lead to tangling of code. Empirical studies have revealed that scattered and tangled code degrades the code quality[8]. The negative impact of scattered and tangled code is reflected not only by internal quality metrics but also by the external quality metrics [8]. Poor modularization of crosscutting concerns results in source code which has more defects and is difficult to maintain[9]. For development of good quality software, it is essential to identify crosscutting concerns[6]. Identification of crosscutting concerns at different stages of software development is essential for 3 reasons. First, for refactoring of legacy system to aspect oriented system. Secondly, for modularized implementation of concern, its crosscutting nature needs to be identified at analysis and design level. Thirdly, for appropriate distribution of testing effort error prone crosscutting code need to be identified.

Several research studies have resulted in number of techniques and tools for identification of crosscutting concerns[17]. Each of these techniques and tools has its own pros and cons. In this paper, we present a comparative study of techniques for identification of crosscutting concerns. We are making the comparison of eight different techniques applicable either on source code or on requirement document: ConcernMapper, Fan-in analysis, Theme/Doc, Prune dependency rule, Formal concept analysis, event traces, line co-change and clone detection. The techniques are compared on five different parameters: Stage of Software development when the technique is applicable, whether the technique is supported by a tool, number of input parameters or the quantum of information required by the technique, format of output generated and
software on which the technique have been applied or tested. Five (Prune dependency rule, concern mapper, theme, clone detection, Fan-in Analysis) of the eight discussed techniques have been applied on a small tutorial application on data structure. The tutorial application shows the result of various operations on different types of data structures like linked list stack, queue, circular queue, graph and trees. The tutorial application consists of 43 requirements and is written in C++. The requirements of the tutorial application are as follows:

R1: Creates a node
R2: Creates a node for single linked list
R3: Creates a node for double linked list
R4: Creates a node for Binary tree
R5: Creates a node for Binary Search tree
R6: Creates a node for Red Black Tree
R7: Creates a node for AVL tree
R8: Creates a vertex node for graph
R9: Creates an edge node for graph
R10: Creates a single linked list
R11: Creates a double linked List
R12: Creates a Circular linked list
R13: Creates a binary tree
R14: Creates a BST
R15: Creates a RBT
R16: Creates a AVL tree
R17: Creates a directed graph
R18: Creates an Undirected Graph
R19: Searches for a key in singly linked list
R20: Searches for a key in double linked list
R21: Searches for a key in circular linked list
R22: searches a node in a graph
R23: adds a node in single linked list at any position
R24: adds a node in double linked list at any position
R25: Adds a node in circular linked list at any position
R26: Add a node in a Binary tree
R27: adds a node in BST
R28: adds a node in RBT
R29: adds a node in AVL tree
R30: delete a node in single linked list at any position
R31: delete a node in double linked list at any position
R32: delete a node in circular linked list at any position
R33: delete a node in a Binary tree
R34: delete a node in BST
R35: delete a node in RBT
R36: delete a node in AVL tree
R37: Search a key in Binary tree
R38: search a key in BST
R39: search a key in RBT
R40: search a key in AVL tree
R41: Find the shortest path between 2 nodes in a graph
R42: Find Minimum spanning tree of a graph
R43: find shortest path for all pairs in a graph
II. A BRIEF OVERVIEW OF IDENTIFICATION TECHNIQUES

In this section, we give a brief overview of eight techniques for identifying crosscutting concerns that we are comparing. The coverage of each technique provided here is elaborate but not exhaustive.

A. Prune dependency rule[8]

It is the manual method of assigning code to the specific concern. Prune dependency rule states that if there is a program element related to a concern then its removal or alteration is guided by removal or change of concern. There are number of subrules related to scenarios where the prune dependency can be established. Let's consider the scenario where the program element P is related to a concern C.

- A reference sub rule states that if a program element P’ references P then P’ is also related to C. the reference
- can be of any type related to method, field, class, interface or enum.
- Dominates sub rule states that if all elements that references P’ are related to C then P’ is also related to C.
- Element Containment sub rule states that if all elements of P’ are related to concern C then P’ is also related to C.
- Concern Containment sub rule states that if a program element P is related to a concern then is sub element should be related to the descendents of the said concern.
- Inheritance sub rule states that if the inherited elements of the program element P’ are related to the concern then P ’ is related to the concern. If on the application of prune dependency rule, a removal or change in concern results in the removal or change of more than one program element then the concern is considered to be crosscutting.

B. ConcernMapper[11]

It is eclipse based plugin for mapping the program elements to the related concern. The tool requires the seed element on the basis of which the code is searched. The seed element can be a word or a phrase related to requirement for which the code need to be mapped. For an example if there is an implementation of word editor and undo feature is one of the requirement then seed element can be undo. The concern mapper maps the program element implementing the functionality related to undo requirement.

C. Fan-in Analysis[14]

This is a semi automatic approach for finding crosscutting concern in the existing system. This approach takes into consideration the fan-in analysis of a method, since high fan-in of a method is an indicator of crosscutting functionality. Fan-in analysis of a system involves three steps: Computing the fan-in for each method, filtering the methods to obtain the smaller set of methods which have high probability of implementing the crosscutting requirement and manual analysis of the filtered method for finding out specific patterns in the call site. The last step involving manual analysis requires the domain knowledge and is more important from aspect mining perspective.

D. Theme[1]

This approach of identifying crosscutting concerns work at requirement and design level. The theme is an element of design that represents the structure and behavior of a feature or requirement. This approach is divided into two parts: theme/Doc and Theme/UML. Theme/Doc works at analysis phase and theme/UML works at design phase. The approach requires the action element to be identified from the requirement documents and the action view is created using the requirements and the actions. In the action view, an action is related to requirements in which it appears. Suppose an action A1 is a verb that is covered in requirement R1, R2 and R3 then the action view will contain action A1 connected to R1, R2 and R3. In the action view, actions are represented as diamonds and requirements are depicted using rounded rectangle if a requirement is coupled to more than one action then it needs to be determined to which action the requirement is more closely related. The aim is to relate each requirement to one action. The actions which are related to more then one requirement are the probable candidates of crosscutting nature. That is the requirements that contain the action which is common to other requirements are the one responsible for bringing the scattered code in the implementation. Such requirements are crosscutting requirements.
E. Line co-change[2]
This is the manual technique of identifying the crosscutting concerns in the source code of the existing system. The method involves identification of lines of source code that are changed together using a versioning system. If the lines of code, belonging to more than one program element, is changed in the process of bug removal or requirement change implementation then such lines of code represents the scattered implementation of a requirement and therefore is an indicator of crosscutting concern. This approach requires that the commit transaction should be performed by the same author and the time span between the commit transactions should be very less. The approach has been studied to be effective in identifying the crosscutting concern in JhotDraw, a 2D graphical framework.

F. Clone detection[5]
This is one of the automated approaches for finding the crosscutting concern. The basic assumption behind this approach is that the presence of code clones in the existing system represents the requirements that are not properly modularized during the implementation phase. There are various code clone detection techniques described in the literature. Text based techniques, token based techniques, abstract Syntax tree (AST) based approach, Program dependency graph (PDG) based approach are some of the examples. The tool CCFinder, a token based technique for clone detection, was developed by T. Kamiya et al. The use of AST as a clone detection technique has been done in “Project Bauhaus” . The PDG based clone detector was developed by Komondoor. The clone detection as a techniques for finding crosscutting concern has been studied by Maginel Bruntink [5].

G. Formal Concept Analysis and execution trace [9]
This approach is applied on the execution trace of an application for which the crosscutting concern needs to be identified. The requirements of an application are represented as use case. For each identified use case the execution trace is generated. Concept analysis is performed on the elements of implementation units and the execution trace for each use case. The generated lattice is used for detecting the crosscutting concerns. If the implementation unit for a use case belongs to different classes then the functionality reflected by the use case is considered to be a crosscutting concern.

H. Event trace[13]
The approach uses the program execution trace to find the patterns in the method calls. The patterns of method calls that are recurring are the indicators for crosscutting concerns. The approach has been studied on Graffiti, editor for graphs, and a specific recurring pattern of method call was observed in a logging concerns in Graffiti . The approach is supported with a tool for analyzing the execution trace of the program.

III. COMPARISON OF TECHNIQUES FOR IDENTIFYING THE CROSSCUTTING CONCERN
We have performed the comparison of eight different techniques for identification of crosscutting concerns.

Table 1 represents the comparison. The comparison is based on five parameters:

- Stage of software development when the technique is applicable: The identification of crosscutting concerns can be performed at various stages of software development. Crosscutting concerns can be identified as early aspects and the detection of early aspects can help in formalization of a good design or they can be identified with in the source code. The identification of crosscutting concerns in source code is required for software maintenance and refactoring.

- Number of input parameter required or the quantum of information required: The usability of a technique or a tool depends upon the amount of information it requires. In case of usability of techniques and tools for identification of crosscutting concern, the quantum and complexity of information required as an input plays an important role. More complex the required input information is the less usable the tool becomes.

- Technique supported with tool: The techniques which are supported with tools can be applied in an easier manner.

- Format of output generated: The usability of a technique or a tool is also guided by the format of output information generated by the technique or a tool. For example, suppose tool for identification of crosscutting concerns in the source code requires the requirement document of an application as an
input and generates the output that shows the mapping of each requirement with its implementation units. Such output information can easily be used for maintenance purpose.

- The applications on which the techniques has been applied: if the technique has previously shown better results on large application then there are more chances that the technique can be used on larger applications. Moreover the widespread use of technique is also guided by the applicability of technique on multiple language applications. This means that if a technique has been tested on application written in C++, java, C and any other language then the acceptability of such techniques is much easier. This information also builds the faith in the techniques. If the technique or a tool has been used or tested on small applications then the technique may not prove to be useful for large software system.

### IV. LIMITATIONS OF TECHNIQUES

The techniques discussed in this paper are effective in detecting the crosscutting concerns at various stages of software development but still there are certain limitations of each approach that need to be discussed.

- Prune dependency rule is a manual method of detecting crosscutting concerns. This approach suffers from two limitations. First, being a manual method the approach cannot be applied on large applications and secondly, the person who intends to find the crosscutting concern should have the knowledge of the code. The knowledge related to concerns or the features of an application is also required.

- Concern mapper: this technique is supported with the tool and therefore can be used with large applications. In this approach an important term from the concern or feature is selected as a seed. This seed acts as the initial basis for searching the code related to a concern from which the seed is chosen. The effectiveness of the concern mapper depends on the type of seed selected. The seed is considered to be good if it is leading to the mapping of the concern with the respective implementation. The selection of good seed requires the domain knowledge.
Fan-in analysis: This technique has only limited tool support. The approach involves three steps and the tool is applicable for only one step. More over the application all the three steps involved in this approach requires a domain knowledge related to the application for which the crosscutting concerns need to be identified.

Theme: The approach is very effective in detecting crosscutting concerns at an early stage of software development. This approach requires the selection of actions from the requirements. This step needs to be done manually. If the number of requirements is very large then finding actions in the requirement becomes tedious job. Again this approach requires domain knowledge for finding out whether an action can be attributed to a particular requirement or not.

Line co-change: this approach requires the CVS report. The technique is not supported by a tool and therefore applicability on large applications is difficult. Formal Concept analysis of execution trace: the approach is only semi automatic and is a lot dependent on the way use cases are designed. The effectiveness of the approach is dependent on the granularity of use cases.

Clone Detection: The technique is very easy to use as it requires the source code as input and is supported by a tool. The tool is able to detect the clones existing in the code. Clone detection as a technique for identification of crosscutting concern is dependent on the approach of the definition of clone. If the size of clone to be searched for, in the source code, is very large then the probability of finding the code clone is very low and if the size of clone is taken to be very small the large number of clone will be detected and this will not find the actual crosscutting concerns.

Event traces: The approach is quite effective as it is easy to use and requires the execution trace of the programs. The basic problem with this approach is the size of data pool of execution trace required to identify the patterns in method call relationship. It is not clear how many execution traces are required to find all the patterns in the method calls. The approach is also supported by an automated tool.

ACKNOWLEDGMENT

I would like to express my sincere thanks and gratitude to the administration of Centre for Development of advanced Computing, Noida(CDAC) and Guru Gobind Singh Indraprastha University, Delhi for extending all the help and support in pursuing the research activity leading to the preparation of this paper.

REFERENCES


