Journal of Computer Science 2 (4): 322-325, 2006
ISSN 1549-3636

A Comparative Classification of Aspect Mining Approaches

[1]Bounour Nora, [2]Ghoul Said and [3]Atil Fadila
[1,3]Laboratory of computer science (LRI), Department of computer science, University Badji Mokhtar
B.P. 12 Annaba (23000), Algeria
[2]Institute of computer science, University of Philadelphia, Jordan

Abstract: In object oriented paradigm, the implementation of a concern is typically scattered over many locations and tangled with the implementation of other concerns, resulting in a system that is hard to explore and understand. Identifying such code automatically greatly improves both the maintainability and the evolveability of the application. Aspect mining aims to identify crosscutting concerns in existing systems, thereby improving the system's comprehensibility and enabling migration of existing (object-oriented) programs to aspect-oriented ones. Aspect are mined either by use of static information or dynamic information of the code. The purpose of this article is to present a survey of the current techniques of aspect mining. We seek to understand both the strengths and limitations of this new area.

Key words: Aspect oriented programming, aspect mining, crosscutting concern, program analysis, reverse engineering.

INTRODUCTION

The tyranny of the dominant decomposition[1] states that no matter how well a system is decomposed into modular units like functions and classes, some functionality will always cut across that modularity. This kind of functionalities are called crosscutting concerns because they involve more than one decomposition unit. Examples of crosscutting concerns are persistence, synchronization, exception handling, error management and logging. Crosscutting concerns are a relevant source of problems to program comprehension and software maintenance. In fact, it is very difficult to evolve a crosscutting concern, because its code is affected by **scattering**. Each modification of crosscutting concern requires the localization of all the code portions pertaining to it. Generally, crosscutting concerns code is mixed and confused with the rest of the code in each unit. This problem is known as **tangling**. Figure 1 illustrates a crosscutting concern tangled in a class. This code mixes business logic with logging.

```
import java.lang.reflect.*;
public class ShoppingCart { private List items = new Vector();
public void addItem(Item item) {
System.out.println("Log:"+this.getClass().getName());
items.add(item);
}public void removeItem(Item item) {
System.out.println("Log:"+this.getClass().getName());
items.remove(item);
}public void empty() {
System.out.println("Log:"+this.getClass().getName());
items.clear();
}}
```
Fig. 1: Logging concern tangled in the shoppingcart class

Code scattering and code tangling are problems that affect applications in a systematic way. The identification of scattered and tangled code that implements concerns is known as aspect mining**.** Aspect mining is defined as a specialized reverse engineering process[2], which aim at investigate legacy systems (source code) in order to discover which parts of the system can be a crosscutting concern. This knowledge can be used for several goals, including refactoring the system into an aspect-oriented one[3]. In Aspect Oriented Programming (AOP), crosscutting concerns are captured via special classes called aspects[4]. Aspects are defined by aspect declarations[5], which may include pointcut declarations, advice declarations, as well as other declarations such as method declarations that are permitted in class declarations. We illustrate in Figure 2 the logging aspect extracted from the code of shoppingcart class.

```
public aspect LoggingAspect {
pointcut                                  loggedMethods(ShoppingCart
shoppingcart):this(shoppingcart)
&& (execution(void ShoppingCart.*(..)));
before(ShoppingCart shoppingcart): loggedMethods(shoppingcart) {
System.out.println("Log:"+
 shoppingcart.getClass().getName());
}}
```
Fig. 2: Logging aspect

Types of crosscutting concerns

Table1: Concern symptoms

Type	Symptoms	Homogenous Heterogeneous
Scattering	X	X
Code duplication	X	

Corresponding Author: Bounour Nora, Laboratory of Computer Science (LRI), Department of Computer Science, University Badji Mokhtar, B.P. 12 Annaba (23000), Algeria, Fax: (213) 38 87 24 36.

We can distinguish between two types of crosscutting concerns[6]. Homogeneous concerns implement the same behavior repeatedly at different locations in a system (table1), whereas heterogeneous concerns implement different behavior, related to the same functionality, at such locations.

Techniques used for aspect mining vary mainly in the kind of concern's symptoms they explore and in the kind of analysis they perform on a legacy system[7]. We distinguish approaches which use the code duplication as the principal symptom of the existence of an aspect; and others which use the scattering.

Approaches based on code duplication: This class of approaches attempt at finding duplicated code. They are based on a static analysis of the code to mine.

Aspect mining using clone detection techniques: Finding crosscutting concerns require specialized types of clone detection.

Token-based clone detection: They apply lexical analysis (tokenization) to the source code and subsequently uses 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 first lexical tool developed is Aspect Browser[8]. It is a programming environment that provides text-based mining. A developer specifies a regular expression that describes the code belonging to the aspect of interest and a color. The programming environment then identifies the code conforming to the regular expression and highlights it using the associated color in the source code editor. The Aspect Mining Tool[9] is an extension of the Aspect Browser that introduces a combination of text-based and type-based mining. Type-based mining considers the usage of types within an application to identify crosscutting code. The tool allows user defined queries based on type usage and regular expressions, displaying matching lines in specific colors. If a line matches more than one criterion, it will be separated into two or more differently colored parts.

The Prism tool[10] in its turn extends the Aspect Mining Tool and additionally provides a type ranking. The type ranking feature is based on the assumption that types that are used widely in the application are a good sign of crosscutting code. Therefore, the tool ranks the types in the system according to their use.

The main downfall of lexical searches is that requires the user to have an in-depth understanding of the base code because:

* They are dependent on the coding practices of the programmer, such as variable or method naming conventions, which are hard to guarantee, especially in a legacy system.
* The user must input a seed. The formulation of a seed that will return meaningful results on a lexical search is a non-trivial task

Other clone detection approaches: These approaches do not require some form of input (a seed) by the developer. They are able to identify aspects without human intervention.

Shepherd use a PDG clone detection technique[11].This approach uses program dependence graph (PDG) which contain information of semantical nature, such as control and data flow of the program.

Bruntink suggest an hybrid technique, which combine AST based clone detection with clone detection tool based on tokenized representations of source code[12]. 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. For further amelioration, Bruntink propose metrics-based clone detection approach[13].

Although, these approaches suffer from some limitation:

* Only homogenous concerns can be identified.
* The identification analysis can miss desirable aspects.
* The filtering of potential candidate's aspect is not fully automatic. Only simple aspects can be identified automatically.

Aspect Mining using formal concept Analysis: Formal concept analysis (FCA) is used to identify meaningful groupings of elements that have common properties. The FCA algorithm takes as input a relation, or Boolean table, T between a set of elements and a set of properties of those elements. The FCA algorithm determines maximal groups of elements and properties, called concepts, such that:

* Each element of the group shares the properties,
* Every property of the group holds for all of its elements,
* No other element outside the group has those same properties,
* No other property outside the group holds for all elements in the group.

All concepts are ordered into a concept lattice. The lattice's bottom concept contains those elements that have all properties. Similarly, the top concept contains those properties that hold for all elements. The concept lattice can be represented by a graph, in which nodes are the concepts and edges represent the sub-concept relations.

When applying FCA for mining source code, first the elements and properties to compute the concept lattice must be chosen.

Tonella and Ceccato[14] apply formal concept analysis to execution trace. The approach used an instrumented version of the system to execute a number of use cases. The output of this execution is a number of execution traces. These traces are then analysed using FCA algorithm. The use cases are the objects of the FCA algorithm, while the methods which get invoked during the execution of a use case are the attributes. The resulting concepts are candidates aspect, if the following two constraints hold:

* The attributes of the concept belong to more than one class.
* Different methods from a same class are contained by more than a use case.

Although, Tourwé *et al.*[15,16] assume that interesting concerns in the source code are reflected by the use of naming conventions in the classes and methods in the system. So, they apply FCA by using the classes and methods in the code as objects. Substring generated from the program entities are used as attributes. The resulting concepts consist out of maximal group of program entities which share a maximal number of substring.

When computing the lattice, lots of concepts are produced, many of which are irrelevant or redundant. Therefore, the discovered concepts must be filtered and classified. The most difficult task is that of deciding manually whether a concept identifies a valid aspect.

Approaches based on scattering

Fan-in analysis: Fan-in analysis mined source code to find symptoms of code scattering. In this case, concerns present themselves as a number of distributed calls to a method implementing a crosscutting functionality. So, the amount of calls (fan-in) is a good measure for the importance and scattering of the discovered concern. Typical examples of concerns include logging, tracing, pre- and post-condition checks and exception handling.
The fan-in analysis consists of three steps[17]:

a. Automatic computation of the fan-in metric for all methods in the investigated system.
b. Filtering of the results from the previous step by

* Eliminating all methods with fan-in values below a chosen threshold (in the experiment, we used a threshold of 10);
* Eliminating the accessor methods (methods whose signature matches a get*/set* pattern and whose implementation only returns or sets a reference);
* Eliminating utility methods, like toString() and collection manipulation methods, from the remaining subset.

Manuel analysis of the methods in the resulting, filtered set.

Analysis of recurring patterns of execution traces: This approach is based on dynamic analyzes of the code source to identify aspects[18]. To this extent, program traces are generated automatically. Then, the traces are analyzed in search of recurring execution patterns. The idea is to detect particular patterns occurring in the trace, such as a call to a particular method **a** that is always followed by a call to a method **b**, or a call to a particular method **c** that always occurs inside a call to a method **d**. Such patterns could point to before/after/around advice of aspects.

Exploratory techniques: Exploratory tools allow a programmer to navigate more intelligently around code. FEAT[19] and JQuery[20] are developed for aspect exploration. Both those tools incorporate **semantic** information (control flow) to navigate in the source code. They focus on providing intelligent exploratory capabilities, with the user controlling much of the function, in order to discover aspects.

This approach puts a heavy burden on the user. It suffers from the following drawbacks:
User must have a considerable amount of knowledge about the overall structure and function of the program being analyzed.

Require a lot of time to identify an aspect due to the required interaction with the user.

CONCLUSION

In this study, we have presented an overview of aspect mining techniques. As a basis of classifying aspect mining techniques, we have used the concern's symptom. Approaches are based on scattering or on code duplication. To discover crosscutting concerns implemented by code duplication a number of tools was developed, which are mainly based on static analysis (Table 2). Some tools require some form of input (seed of information) by the user[8-10]. More advanced tools, which are able to identify aspects without human intervention, are based on clone detection techniques[12,11,21]. Other tools, use formal concept analysis[14-16]. Scattering was a symptom used by other approaches, such as for localizing the recurrent pattern scattered in the code[18,22]. Also, to calculate a set of candidate crosscutting concerns characterized by distributed calls[17].

Aspect mining tools remain limited, because the step of filtering the set of candidates aspects is usually manual. Hybrid approach would be considered to optimize the set of candidate aspect. Full automation of aspect mining process remains a lofty goal.

Table 2: Aspect mining tools

	Code duplication based on Approaches	
Tool	Analysis type	Aspect mining result
Aspect browser[8] AMT[9] Prism[10]	Lexical Lexical+Type Lexical+Type	Highlighted code
Ophir[11]	PDG-clone detection	List of candidate aspects Manually inspected
Delfstof[16]	FCA-analysis	List of candidate aspects Exploratory inspected
Dynamo[14]	FCA analysis of execution traces	List of candidate aspects Manually inspected
Scattering based approaches		
Tool	Analysis type	Aspect mining result
Dynamit[18]	Dynamic analysis of execution traces	List of candidate aspects
Exploratory approaches		
Tool	Analysis type	Aspect mining result
Jquery[19] Feat[20] Sextant[13]	Semantic analysis	Intelligent Exploration

REFERENCES

1. Peri, T., H. Ossher, W. Harrison and Jr. S.M. Sutton, 1999. N degrees of separation: Multi-dimensional separation of concerns. In Proc. 21st Intl. Conf. Software Engg., IEEE Computer Society Press, pp: 107-119.
2. Neil, L. and A, Rashid, 2002. Mining aspects. Workshop on early aspects: Aspect-oriented requirements engineering and Architecture Design. Enschede, The Netherlands.
3. Arie, v.D., M. Marin and L. Moonen, 2003. Aspect mining and refactoring. In First Intl. Workshop on REFactoring: Achievements, Challenges, Effects (REFACE).
4. Kiczales, G., 1997. Aspect-oriented programming. Proc. European Conf. on object-oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241.
5. The AspectJ Team, 2001. The Aspect Programming Guide.
6. Tzilla, E., M. Aksit, G. Kiczales, K. Lieberherr and H. Ossher, 2001. Discussing Aspects of AOP. Communications of the ACM, 44: 33-38.
7. Bounour, N. and S. Ghoul, 2005. A survey of aspect mining techniques. Congrès International en informatique appliquée-CIIA05- BBA Algérie, pp: 241-246.
8. William, G.G., Y. Kato and J.J. Yuan, 1999. Aspect browser: Tool support for Managing Dispersed Aspects. Technical Report CS99-0640, Department of computer Science and Engineering, University of California, San Diego.
9. Jan, H. and G. Kiczales, 2001. Overcoming the prevalent decomposition in legacy code. In Proc. ICSE Workshop on Advanced Separation of Concerns, Toronto, Canada.
10. Zhang, C. and H.-A. Jacobsen, 2003. A Prism for Research in Software Modularization through Aspect Mining. Technical report, Middleware Systems Research Group, University of Toronto.
11. David, S., E. Gibson and L. Pollock, 2004. Automated mining of desirable aspects. Technical Report 4, Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716.
12. Magiel, B., A.v. Deursen, R.v. Engelen and T. Tourw´e, 2004. An evaluation of clone detection techniques for identifying crosscutting concerns. In Proc. Intl. Conf. Software Maintenance (ICSM). IEEE Computer Society.
13. Michael, E., M. Haupt, M. Mezini, T. Schafer, S. Djukanovic and M. Vekic, 2005. Graph-based software navigation with SEXTANT. Submitted for publication.
14. Paolo, T. and M. Ceccato, 2004. Aspect mining through the formal concept analysis of execution traces. In Proceedings of the 11th Working Conference on Reverse Engineering, IEEE Computer Society, pp: 112-121.
15. Tom, T. and K. Mens, 2004. Mining aspectual views using formal concept analysis. In 4th IEEE Intl. Workshop on Source Code Analysis and Manipulation, pp: 97-106.
16. Mens, K. and T. Tourwé, 2005. Delving source-code with formal concept analysis. Elsevier J. Computer Languages, Systems & Structures. To be published.
17. Marius, M., A.v. Deursen and L. Moonen, 2004. Identifying aspects using fan-in analysis. In Proc. 11th Working Conf. Reverse Engineering, IEEE Computer Society.
18. Silvia, B. and J. Krinke, 2003. Aspect mining using dynamic analysis. In GI-Software technik-Trends, Mitteilungen der Gesellschaft f˙ur Informatik. Bad Honnef, Germany, 23: 21-22.
19. Robillard, M.P. and G.C. Murphy, 2002. Concern graphs: Finding and describing concerns using structural program dependencies. In ICSE.
20. De Volder, K., 2002. The jquery tool: A generic query-based code browser for eclipse. Presentation at Eclipse. BoF at OOPSLA 2002.
21. Bruntink, M., 2004. Aspect mining using clone class metrics. In 1st Workshop on Aspect Reverse Engineering.
22. Jens, K. and S. Breu, 2004. Control-flow-graph-based aspect mining. In Workshop on Aspect Reverse Engineering.