Aspect Mining and Refactoring

Arie van Deursen[*]
Software Evolution Research Lab
CWI & Delft Univ. of Technology
The Netherlands
Arie.van.Deursen@cwi.nl

Marius Marin
Software Evolution Research Lab
Delft University of Technology
The Netherlands
A.M.Marin@ewi.tudelft.nl

Leon Moonen
Software Evolution Research Lab
Delft Univ. of Technology & CWI
The Netherlands
Leon.Moonen@acm.org

Abstract

In this paper we argue for the need for research in the area of aspect mining, covering concepts, principles, methods and tools supporting the identification of aspects in object-oriented software systems as well as the subsequent refactoring of such systems in aspect-oriented systems.

We give an overview of the state of the art in this area which shows some of the research directions that have been considered up to now and describes ongoing efforts.

We provide an initial assessment of how reverse engineering and software exploration techniques can help in aspect mining, identify promising research directions and pose a number of research questions that could help to advance the state of the art in aspect mining and refactoring.

1. Introduction

Aspect-Oriented Programming In software engineering, decomposing a large software system into smaller parts is an essential way of managing the complexity and evolution of today's software systems. Such a decomposition results in a "separation of concerns", and facilitates parallel work, team specialization, localized change, systematic testing and quality assurance, and work planning.

Unfortunately, certain behavior of software systems, such as error handling or logging, are inherently difficult to decompose and isolate, leading to reduced reliability and maintainability of these systems. *Aspect-Oriented Programming* (AOP) came up in the 90s as a paradigm aimed at making such *cross cutting concerns* (or, more briefly *aspects*) explicit, and uses code generation techniques to weave aspects back into the application logic [7]. This separation solves the inherent problems of code scattering and tangling in large object-oriented systems.

Currently, as AOP is entering the innovation phase, new challenges arise while the technology becomes more largely adopted and its practice extends. The release of aspect-oriented development tools such as AspectJ[1], contributions by several research groups and the recent integration in application servers such as JBoss[2] and BEA's WebLogic[3] illustrate the increasing popularity of AOP.

Aspect Mining The topic of this paper is *aspect mining and refactoring*: the search for candidate aspects in existing object-oriented systems and isolating them from the system into separately described aspects. The goal of migrating an object-oriented system into an aspect-oriented one includes, amongst others, improving the comprehensibility of the system, and thereby improving it's maintainability and extensibility (evolvability).

Moreover, in order to be convincing when arguing that AOP is an important step forward, a number of important questions need to be answered. One of them is investigating new cross-cutting concerns, other than the classical example of logging, and ways for this to be done.

Additionally, a number of concerns regarding the use of AOP have been expressed, such as the risk of getting "spaghetti code" and ad hoc design due to the improper use of free-defined and unrestricted pointcuts. These concerns raise the question of *when* AOP is really needed and when OOP is enough for the right solution. We believe that aspect mining can help in finding answers to these questions and others alike.

Software System Mining Software mining techniques aim at finding valuable information in the source code of a software system, in order to make this information explicitly available to software engineers involved in the evolution of that system. A typical software mining example is *business rule extraction*.

Software system mining is supported by *software exploration* [10]. This typically involves three steps: (1) data collection from source code, (2) knowledge inference based on abstraction from the collected data, and (3) information presentation using, for example, hypertext and visualizations. In this paper, we provide an initial assessment of how software exploration techniques can help in aspect mining.

[*] A. van Deursen received partial support from SENTER, project Ideals, hosted by the Embedded Systems Institute, Eindhoven, The Netherlands.

[1] `http://eclipse.org/aspectj/`
[2] `http://www.jboss.org/`
[3] `http://www.bea.com/`

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 behavior. Refactorings are systematically organized into catalogs, in a similar way as design patterns. Hints that certain refactorings 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 research in the area of aspect mining, covering concepts, principles, methods and tools supporting the identification of aspects in object-oriented software systems as well as the subsequent refactoring of such systems into aspect-oriented systems.

2. State of the Art

To date, a number of research groups are working in the area of aspect mining. Here we summarize the main results and ongoing efforts.

The identification of aspects first of all requires a clear idea of where aspects can be found. As such, the study of general or domain-specific aspects already in use is a prerequisite to aspect mining. This can start from known crosscutting concerns as, for example, the ones from BEA's WebLogic Aspect framework. The framework, based on AspectJ, defines several J2EE pointcuts against which users can write advices.

One of the first research activities in identifying crosscutting concerns in source code was performed by Hannemann and Kiczales [5]. Their Aspect Mining Tool (AMT) supports both text and type-based analysis. Moreover, it can be extended with other types of analysis, such as signature-based searches. Since each of these analyzes has benefits and drawbacks, AMT has been set up as a “multi-modal analysis tool”, permitting the combined use of different techniques. Each analysis works as a query, and results in a series of matched lines, which are visualized in the system's source code.

The “Aspect browser” tool [4] is based on the assumption that aspects, defined as secondary design decisions, have a signature (textual-pattern, lexical token), which is a textual regular expression that will help to their identification. Similar to the text based analyzes of AMT, the successfulness of the tool strongly depends on naming conventions fallowed by the analyzed code. The location of the code that implements the concerns is represented graphically by views that are based on the Seesoft concept [3].

Aspects can be also identified by using dynamic program analysis techniques. This approach is proposed by [1], which defines certain classes of aspects together with their dynamic trace patterns. The mining process searches for these patterns in program traces that were captured during execution. The main classes considered are *outside-aspects*, that is the call of method 'a' is always followed by the call of method 'b', and *inside-aspects*, that is the call of method 'b' is always inside a call of method 'a'. They also distinguish subclasses such as *outside-aspects* that are *before* or *after* a specific method call, and *inside-aspects* that are *first-* or *last-in* a specific method call.

The research of Rashid and Loughran [8] addresses the question how aspects found in an existing system should be represented. The approaches considered are the *storing of the aspect as an object* (binary/character) together with the object's description (meta-data) that can be queried, the *mapping of the aspect anatomy to a database model* that will allow the storing of aspect's properties, and the *hybrid approach*, a combination of the previous two. The mapping of the aspects to a relational database model is presented in more detail in [9].

Aspect mining is used by Zhang and Jacobsen [12] to conduct quantitative analysis of the tangling and scattering phenomenon in existing legacy implementations of middleware platforms/architectures (CORBA). New aspects specific to the chosen platform and defined as abstractions that crosscut the major architectural components considered by the authors are reported. These are the dynamic programming model and portable interceptors. The AOP based re-factorization of a number of aspects identified was performed, and the quantified results aim to demonstrate a reduction of the complexity and improvement of performances compared to the original implementation. The tool used in the analysis, AMTEX[4], was built on top of the AMT tool described earlier.

Concern Graphs [11] represent another way to document and analyze concerns. They are based on localizing an abstracted representation of the program elements contributing to the implementation of the concern. The structure of a concern is stored in a concern graph and, at the same time, the relationships between the concern's elements, such as classes, methods and fields are documented. This approach is implemented in the tool FEAT. It supports the analysis of the dependencies between a concern and the rest of the program and allows the viewing of the source code, in a Java system, associated to a concern graph element. For concern identification, FEAT supports the use of structural queries and integrated lexical searches. The tool displays a concern graph as a collection of trees with respect to certain convention, e.g. the root of each tree is a class that contributes to the implementation of the concern. FEAT is also implemented as an Eclipse plug-in.

JQuery [6] is another Eclipse plug-in which provides a generic browser that allows the user to define logic queries in a specific query language. The queries can be run against the source code of a Java working set of interest, and the results can be viewed and browsed. The organization and nav-

[4]`www.eecg.utoronto.ca/~czhang/amtex`

igation of the code can be based on different structural relationships, regular expression matches and complex searches for structural patterns. While JQuery is generally useful, the authors argue that good navigation support is particularly important when exploring cross cutting concerns.

3. Research Questions

When looking at the existing work from a reverse engineering and software exploration perspective, one can see that the knowledge inference step (which "invents" the higher levels of abstraction) is underdeveloped: existing aspect mining work concentrates on how to distill concern-related data from the sources (signatures, lexical pattern matching), and how to present such data via, for example, visualization. These steps may support the human engineer in aspect mining, but are unable to come up with any aspect themselves.

In this section we focus a bit more on the knowledge inference step and explore a number of research directions can help to advance the state of the art in aspect mining and refactoring.

Clone Detection The lack of support for aspects forces developers to scatter code over various modules. The actual code, dealing for example with logging or error handling, is likely to be similar for the various modules. This suggests that *clone detection*, a general technique for detecting duplicated code, may be beneficial for aspect identification. This raises the following questions:

- What specific clone detection techniques are suitable for aspect identification?
- What error margins (false positive, false negative) do we obtain when using clone detection for aspect identification?
- How can aspect-specific clone detection be integrated in software exploration tools?

Slicing Slicing is a program analysis technique aimed at isolating code affecting the value of certain variable only. An interesting research question is how slicing can be best put to use for the isolation candidate aspects from the rest of the code. One issue to address concerns the nature of the slicing techniques needed for aspect isolation. Another question that needs to be addressed is how information obtained by traditional slicing techniques can be used for the purpose of aspect identification.

Dynamic Analysis A further technique for finding, e.g., code devoted to logging is to run an instrumented system once with logging switched on, and a second time with logging switched off. An analysis of the execution traces will help to find all code devoted to logging, thus delivering a first step towards reconstruction of all logging code.

This raises questions on domain-specific aspect mining techniques, trace analysis techniques, and appropriate code instrumentation techniques. An initial step towards the use of such dynamic analysis is discussed in [1]. However, there remain a lot of open issues that need further investigation.

Cluster and Concept Analysis Aspect-oriented programming is about remodularization. An active area of reverse engineering research is concerned with the use of automated cluster analysis techniques aimed at grouping pieces of functionality (procedures, programs) into coherent modules.

AOP deals with those pieces of functionality that are hard (impossible) to modularize well. An interesting question is how the presence of cross cutting concerns affects automated remodularization attempts.

A potentially more promising route is the use of *concept analysis*. Concept analysis shows all possible modularizations in a concise *concept lattice*. Further research is needed in order to understand how cross cutting concerns would show up in concept lattices. A comparison of cluster and concept analysis is provided in [2].

Refactoring Refactoring involves the systematic categorization of code smells and elementary program transformations, as well as techniques to automate smell detection and the refactoring process. In relation to aspect identification and migration the following questions arise:

- How can existing code smells be used to identify candidate aspects?
- How can the introduction of aspects be described in terms of a catalog of new refactorings?
- What software exploration techniques are needed to support the detection of these aspect-related smells?
- What testing implications does migration to an aspect-oriented software development approach bear?

Case Studies Aspect mining research results will have to be validated by means of a series of case studies. To ensure repeatability of the experiments, the subject systems should be selected from the open source domain. In addition, subject systems from industrial partners interested in aspect identification in their systems can be used to ensure practical applicability of the proposed techniques.

Open questions in this area are how these subject systems should be selected (i.e., what selection criteria to use) and how the mining results can be validated (both techniques and criteria). For example, to assess the completeness of a mining technique, the potential aspects in the subject systems should be known in advance but it's highly unlikely that this will be case for any industrial or open source system.

Assessing the Value of Aspect Isolation Mining and refactoring aspects is done with a certain goals in mind, for example improving the maintainability of code, or improving it's comprehensibility or extensibility. Consequently, before actually performing the source code transformation that would isolate certain candidate aspects from the sources and capture them in an aspect-oriented way, one needs to assess the value of that migration with respect to these goals.

This need for assessing the value of aspect introduction raises the following questions:

- How would one go and measure the value of aspect introduction in a system?
- What are suitable metrics to measure the effects of aspect introduction?
- Do aspect-oriented systems need different metrics than 'ordinary' systems?

4. Concluding Remarks

Aspect mining research is concerned with the development of concepts, principles, methods and tools supporting the identification of aspects in object-oriented software systems as well as the subsequent refactoring of such systems into aspect-oriented systems.

In this paper, we explored the state of the art in aspect 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 aspects; and (2) aspect smells requiring (aspect specific) refactorings. Moreover, aspect mining can be used on completed systems, in order to offer better support for future evolution of the system.

Additionally, aspects can serve the role of landmarks during the exploration of a software system[10, Chap. 1]. Last but not least, aspect mining research will be useful in order to assess the actual value of aspect-oriented development.

References

[1] S. Breu and J. Krinke. Aspect mining using dynamic analysis. In *Workshop on Software-Reengineering*, Bad Honnef, 2003.

[2] A. van Deursen and T. Kuipers. Identifying objects using cluster and concept analysis. In *Proc. Int. Conf. on Software Engineering (ICSE)*, pages 246–255. ACM, 1999.

[3] S. G. Eick, J. L. Steffen, and E. E. Sumner. Seesoft – tool for visualizing line oriented software. *IEEE Transactions on Software Engineering*, 18(11):957–968, November 1992.

[4] W. G. Griswold, Y. Kato, and J. J. Yuan. Aspect-browser: Tool support for managing dispersed aspects. Technical Report CS1999-0640, 3, 2000.

[5] J. Hannemann and G. Kiczales. Overcoming the prevalent decomposition of legacy code. In *Proc. Workshop on Advanced Separation of Concerns*. IEEE, 2001.

[6] D. Janzen and K. De Volder. Navigating and querying code without getting lost. In *Proc. 2nd Int. Conf. on Aspect-Oriented Software Development (AOSD)*, pages 178–187. ACM Press, March 2003.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *11th Europeen Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.

[8] N. Loughran and A. Rashid. Mining aspects. In *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, Enschede, The Netherlands, 2002. AOSD Satellite workshop.

[9] N. Loughran and A. Rashid. Relational database support for aspect-oriented programming. In *Proceedings of NetObjectDays Conference*, number 2591 in LNCS, pages 233–247, 2002.

[10] L. Moonen. *Exploring Software Systems*. PhD thesis, Faculty of Natural Sciences, Mathematics, and Computer Science, University of Amsterdam, December 2002.

[11] M. P. Robillard and G. C. Murphy. Concern graphs: Finding and describing concerns. In *Proc. Int. Conf. on Software Engineering (ICSE)*. IEEE, 2002.

[12] C. Zhang and H-A. Jacobsen. Quantifying aspects in middleware platforms. In *Proc. 2nd Int. Conf. on Aspect-Oriented Software Development (AOSD)*, pages 130–139. ACM Press, March 2003.