A Survey of Aspect Mining Tools and Techniques

Andy Kellens[1]
Programming Technology Lab
Vrije Universiteit Brussel
Pleinlaan 2
B-1050 Brussels, Belgium
akellens@vub.ac.be

Kim Mens
Département d'Ingénierie Informatique
Université catholique de Louvain
Place Sainte Barbe, 2
B-1348 Louvain-la-Neuve, Belgium
kim.mens@info.ucl.ac.be

Project IWT 040116 "AspectLab"
Workpackage 6 - Deliverable 6.2.a

June 30, 2005

[1]Ph.D. scholarship funded by the "Institute for the Promotion of Innovation through Science and Technology in Flanders" (IWT Vlaanderen).

1 Introduction

Aspect-oriented software development (AOSD) tries to solve the problem of separating the core functionality of a software system from concerns that have a more system-wide behavior and that tend to cut across the chosen decomposition of the software system. This problem is sometimes referred to as the "tyranny of the dominant decompositon" [37]. To overcome this prevalent decomposition [18], the AOSD paradigm provides new language constructs which allow cross-cutting concerns to be written down in a new kind of module named *aspect*.

Almost ten years after its initial conception, this technology has now left the research lab and is starting to be adopted by industry, which poses new interesting research problems. In particular, in order to apply aspect-oriented techniques to legacy systems at use in industry, or to migrate those systems to an aspect-oriented solution, there is a need for tools and techniques that help in identifying the cross-cutting concerns in such systems and refactoring them into aspects. The study and development of such approaches is the objective of the emerging research domains of 'aspect mining' and 'aspect refactoring' :

Aspect Mining is the activity of discovering, in the source code of a given software system, those cross-cutting concerns that potentially could be turned into aspects. We refer to such concerns as 'aspect candidates'.

Aspect Refactoring is the activity of actually transforming the identified aspect candidates into real aspects in the source code.

In this survey we focus only on the activity of *aspect mining* and report on a number of different code-based techniques, tools and methodologies that have been designed to aid a software engineer in identifying aspect candidates in a legacy system. Apart from giving an introduction to the research problem of aspect mining and its envisaged solutions, the contributions of this survey are:

1. the definition and motivation of the problem of *aspect mining*;
2. the identification of the key issues to be solved;
3. the presentation of an exhaustive list of existing aspect mining approaches;
4. a comparative taxonomy of these approaches;

5. the identification of important open problems remaining;
6. an exploration of links with other research domains;
7. the proposition of possible avenues for future research to improve the state-of-the-art.

Points 1 and 2 are addressed in Section 2. Section 3 briefly summarizes some known techniques and approaches, making an explicit distinction between manual approaches supported by advanced browsing tools and more automated approaches that are often inspired by data mining, software comprehension or program analysis techniques. Because the remainder of the paper focusses on automated approaches only, Section 4 lists some researchers working on a variety of automated aspect mining techniques and Section 5 details these approaches. Section 7 then provides a comparative taxonomy of these different techniques, based on a set of objective criteria listed in Section 6. From this taxonomy, a list of open research problems are distilled as well as a list of possible avenues for future research. Related work from other research domains like data mining, software comprehension and program analysis is discussed in Section 8. Section 9 concludes the paper.

2 Aspect mining

The industrial adoption of the object-oriented paradigm in the nineties lead to a need for migrating legacy software systems to an object-oriented solution and a subsequent boost in research on software reverse engineering, reengineering, restructuring and refactoring. The same is currently happening for the aspect-oriented paradigm. To migrate legacy systems to AOSD there is a need for advanced browsers that can help software engineers in identifying the cross-cutting concerns in legacy systems or for more automated tools to discover such concerns, as well as a need to refactor the discovered cross-cutting concerns into aspects.

The reasons for wanting to migrate a legacy system to an aspect-oriented solution are multiple. In a system implemented using classic techniques, there exist certain concerns which cannot be localized using the available modularization mechanisms, but instead cut across the source code of the system. As a consequence of the existence of these cross-cutting concerns, the implementation of certain concerns is characterized by duplicated code, scattering of the concern throughout the entire system and tangling of the concern specific code with that of other concerns, making it harder to understand, maintain and evolve the system. Using aspect-oriented technology,

Définition
Importance

these cross-cutting concerns can be modularized using language features like pointcuts and aspects [26]. In the resulting system, the different concerns are cleanly separated making the system easier to maintain and extend.

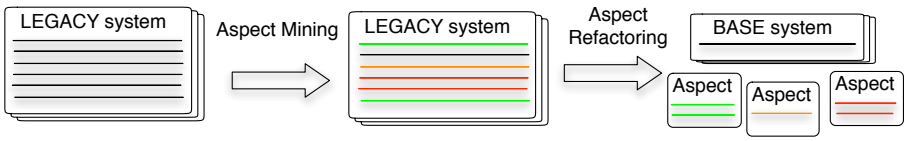

Figure 1: Migrating a legacy system to an aspect-oriented system

As mentioned earlier and summarized in Figure 1, the process of migrating a legacy system into a system using aspects consists out of two steps: the identification of aspect candidates and the refactoring of these candidates into aspects. In this survey we investigate techniques and tools that focus on identifying possible aspects. This is not a trivial task, due to the size and complexity of current-day systems and the lack of explicit documentation on the crosscutting concerns present in those systems. Therefore, a need exists for approaches which (semi-) automatically aid a developer in mining a legacy system for aspects.

As stated in the introduction, we define *Aspect Mining* as:

> The activity of discovering, in the source code of a given software system, those cross-cutting concerns that potentially could be turned into aspects.

Although most publications seem to agree on this definition and terminology, some papers use the term *aspect identification* synonymously with *aspect mining* (for instance [17] and [27]). We advocate usage of the latter term in order to avoid confusion with research that aims at the identification of aspects during requirements analysis [2, 32, 38] or architecture design [3], who use the term *aspect identification*.

Although aspect mining is still in its infancy, we think a survey like this one is relevant and necessary because

- many different researchers have started to work on aspect mining,
- using many different techniques.
- Though many techniques are promising, most are still premature.

- Some researchers [11] have identified the need for a combination of different techniques.

3 Known techniques and approaches

Before going in-depth on the different aspect mining techniques and providing a taxonomy, we start out by taking a look of the two major kinds of approaches we can recognize:

Dedicated browsers A first class of approaches consist out of advanced special-purpose code browsers which aid a developer in manually navigating the source code of a system to explore cross-cutting concerns. Although the primary goal of these approaches is not to explicitly mine for aspects, but rather to document and localize the cross-cutting concerns in order to maintain and evolve a system, the dedicated browsers can be used to identify possible aspects in a system.

A user of such a browsing approach starts out with a 'seed' of a concern, a starting point in the code, and uses the browser to further explore this concern. To do this the browser proposes other hotspots in the code which might be related to the concern or provides the user with a query language to manually traverse the concern. Examples of such approaches are Concern Graphs [33], Intensional Views [29], Aspect Browser [15], (Extended) Aspect Mining Tool [18][42],Prism [43], JQuery [20], ...

(Semi-)automatic identification of aspect candidates Complementary to the dedicated browsers, there exist a number of techniques which have as goal to automate the process of identifying aspects and which propose their user one or more aspect candidates. To this end, these techniques reason about the source code of the system or data that is acquired by executing or manipulating the code. All techniques seem to have at least in common that they search for symptoms of cross-cutting concerns using either techniques from data mining and data analysis like formal concept analysis and cluster analysis, or more classic code analysis techniques like program slicing, software metrics and heuristics, clone detection and pattern matching techniques, dynamic analysis, and so on.

In this particular survey we focus only on the second kind of techniques which semi-automatically assist a developer in the activity of identifying cross-cutting concerns in an existing system.

Approches

4 Researchers

This subsection gives a brief overview of different researchers working on automated aspect mining techniques, together with a brief description of their current research interests. [1]

BREU Silvia and KRINKE Jens use dynamic and static analysis to detect recurring calling patterns and extract cross-cutting concerns from them.

BRUNTINK Magiel and TOURWE Tom are involved in the Ideals project, where they study how cross-cutting concerns can be reverse engineered from large-scale industrial applications and how AOSD techniques can improve the quality of such applications.

CECCATO Mariano and TONELLA Paolo combine the techniques of dynamic analysis and formal concept analysis to mine non aspect-oriented source code for potential aspects.

GYBELS Kris and KELLENS Andy explored heuristic-based aspect mining techniques as well as the technique of inductive logic programming to automatically uncover the pointcuts in a non aspect-oriented software.

MARIN Marius, MOONEN Leon and VAN DEURSEN Arie studied the technique of fan-in analysis to semi-automatically identify aspects in Java source code that is not written in an aspect-oriented way.

MENS Kim and TOURWE Tom explore the technique of formal concept analysis to mine Smalltalk or Java source code for potential aspects and cross-cutting concerns, based on similarities in method and class identifiers.

SHEPHERD David has developed a framework (Timna) for combining aspect mining analyses. He has also used code clone detection to identify potential aspects.

[1] For an up-to date list of aspect mining and aspect refactoring researchers we refer to the Aspect Identification and Refactoring portal : `http://www.info.ucl.ac.be/ingidocs/people/km/AIRPort/AIRPort.htm`.

ZAIDMAN Andy uses datamining algorithms to uncover important classes in a system's architecture, that are prime candidates for the introduction of aspects.

5 Approaches

Now that we have explained the research problem and identified the key actors working on aspect mining, we take a closer look at different automated aspect mining approaches that have been proposed the last few years.

5.1 Analyzing recurring patterns of execution traces

Breu and Krinke propose an aspect mining technique named *DynAMiT* [7], which analyses program traces reflecting the run-time behaviour of a system in search of recurring execution patterns. To do so, they introduce the notion of *execution relations* between method invocations. Consider the following example of an event trace:

```
B() {
   C() {
      G()
      H()
         }
       }
A() {}
```

Breu and Krinke distinguish between 4 different execution relations: outside-before (e.g. B is called before A), outside-after (e.g. A is called after B), inside-first(e.g. G is the first call in C) and inside-last(e.g. H is the last call in C). Using these execution relations, their mining algorithm identifies aspect candidates based on recurring patterns of method invocations. If an execution relation occurs more than once, and recurs uniformly (for instance every invocation of method B is followed by an invocation of method A), it is considered to be an aspect candidate. Of course, to ensure that the aspect candidates are sufficiently crosscutting, there is an extra requirement that the recurring relations should appear in different 'calling contexts'. Although this approach is inherently dynamic, the authors have repeated the experiment using control-flow-graphs [25], a static technique, to calculate the execution relations. Breu also reports on a hybrid approach [6] where the dynamic information is complemented with static type information in order to remove ambiguities and improve on the results of the technique.

5.2 Formal concept analysis of execution traces

Tonella and Ceccato [39] developed *Dynamo*, a mining technique which applies *formal concept analysis* (FCA) [13] to execution traces in order to identify possible aspects. Formal concept analysis is a branch of lattice theory which, given a set of objects and attributes describing those objects, creates concepts which are maximal groups of objects that have common attributes. When analyzing a system using Dynamo, an instrumented version of the system is used to execute a number of use cases. The output of this execution is a number of execution traces. These traces are then be analyzed using the 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. Resulting concepts which are specific to one particular use case, i.e. whose extent contain the trace for the given use case only, are aspect candidates if the following two constraints hold:

- Scattering: The attributes (methods) of the concept belong to more than one class.
- Tangling: Different methods from a same class are contained by more than one use-case specific concept

5.3 Formal concept analysis of class and method names

Tourwé and Mens [40] propose an alternative aspect mining technique which relies on formal concept analysis. Unlike the Dynamo approach which we discussed previously, Tourwé and Mens' *DelfSTof* tool analyses the source-code of a system (experiments have been conducted on Smalltalk [40] and on Java code [12]). Their approach performs an identifier analysis using the FCA algorithm. The assumption behind this approach is that interesting concerns in the source-code are reflected by the use of naming conventions in the classes and methods of the system. As input to the FCA algorithm, the classes and methods in the system are used as objects. As attributes, the FCA algorithm takes as input substrings generated from the program entities used as objects. For instance, a class named `QuotedCodeConstant` is split up in the strings 'Quoted', 'Code' and 'Constant'. Substrings with little meaning, like 'a', 'with', ... are discarded from the results. The resulting concepts consist out of maximal groups of program entities which share a maximal number of substrings. After filtering out unimportant concepts, a large number of concepts remain which need to be inspected manually. Apart from being able to detect a number of programming idioms, design

patterns and certain refactoring opportunities [30], by restricting the concepts to those that are crosscutting (i.e. the involved methods and classes belong to at least two different class hierarchies) the same approach can be used for aspect mining as well [40].

5.4 Natural language processing on source code

Similar to the previous technique, Shepherd et al. [36] propose a technique that is based on the assumption that cross-cutting concerns are often implemented by rigorously using naming and coding conventions. Their approach uses natural language processing (NLP) information as an indicator for possible aspect candidates. They report on an experiment in which they use a NLP technique called *lexical chaining* [31] in order to find groups of related source-code entities which represent a cross-cutting concern. Lexical chaining takes as input a collection of words and outputs chains of words which are strongly related. In order to create the chain, the algorithm uses a semantical distance measure between two words. Shepherd et al. used WordNet [10], a catalogue of semantical paths between words to use this measure in combination with information about the part of speech of each word. In order to mine for cross-cutting concerns, they apply the chaining algorithm to the comments, method names, field names and class names of the system they are analyzing. In order to identify the aspect candidates, the user of their approach needs to manually inspect the resulting chains.

5.5 Detecting unique methods

Gybels and Kellens [16, 17] propose the use of heuristics to identify possible cross-cutting concerns. They observe that, in pre-AOP days, cross-cutting concerns were often implemented in an idiomatic way. 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, Gybels and Kellens propose the "Unique Methods" heuristic which is defined as: *"a method without a return value which implements a message implemented by no other method"*. After calculating all the Unique 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), the user has to manually inspect the resulting methods in order to find suitable aspect candidates.

Regardless of the simplicity of this approach, the authors demonstrated the applicability of their technique by detecting typical aspects like tracing, update notification and memory management in the context of a Smalltalk image.

5.6 Hierarchical clustering of related methods

Shepherd and Pollock [35] report on an experiment in which they used agglomerative hierarchical clustering [21] 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 a certain threshold. They implemented this technique as part of an aspect-oriented IDE named *AMAV* (Aspect 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 *cross-cutting pane* which displays the methods related to a cluster as well as an *editor pane*, in which the class context of a particular method is displayed.

He and Bai [19] propose another aspect mining technique based on cluster analysis. They start from the assumption that if methods appear together in a number of different modules, this may be a good indication that a hidden cross-cutting concern is present. As input for the 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 is a representation of the dissimilarity of methods, based on wether a method invokes another method in a different calling context.

5.7 Fan-in analysis

Marin et al. [27] noticed that many of the well-known cross-cutting concerns are implemented using a technique which exhibits a high fan-in. They propose using a fan-in metric in order to discover cross-cutting concerns in the source code. They define the fan-in of a method m as the number of distinct method bodies which can invoke m. Because of polymorphism, a call to a method m contributes to the fan-in of all methods refining m as well as all methods which refine m. Their mining algorithm comprises:

- Calculating the fan-in metric for all the methods of the system that is being analysed.
- Filtering the results: next to filtering *accessor* and *mutator* methods, as well as utility methods like for instance `toString()`, the number of considered methods is also limited by only considering the methods with a fan-in value higher than a certain threshold.
- Manually analyzing the remaining methods.

The authors present an experiment in which cross-cutting concerns where mined with a high precision: one third of all with high fan-in were seeds leading to an aspect. Moreover, 60% of the remaining two thirds were removed automatically. This technique seems most suited for finding aspect candidates with a large footprint. Due to the use of a threshold, aspects with a smaller footprint may be ignored by this approach.

5.8 Detecting clones as indicators of crosscutting concerns

As we already mentioned earlier, there exist a number of symptoms which may be good indicators of cross-cutting concerns in the source code of a system. One such symptom is 'code duplication': because the cross-cutting concerns could not be cleanly modularized, certain parts of the implementation show high levels of duplicated code. Two techniques use this observation to mine for aspect candidates.

1. A first technique, presented by Shepherd et al. [34] and implemented as an extension to the *Ophir* framework, makes use of *program dependence graphs* (PDG) to detect possible aspects. In a PDG, each statement in the code is represented by a node; the edges of the graph consist of control or data dependence relations between the statements. By comparing PDGs [23, 24], this technique is able to identify code duplication in the beginning of a method (i.e. aspect candidates for a 'before' advice). After filtering and coalescing the resulting PDGs, a number of possible aspect candidates remains.
2. Bruntink et al. [8, 9] make use of three other clone detection techniques, namely *token-based* [1], *AST-based* [4] and *metrics-based clone detection* [28]. They applied these techniques to a large system written in C in which different cross-cutting concerns where annotated by a developer. In order to measure the effectiveness of the applied clone

detection techniques in order to find aspects, the results of the techniques were compared with the manually documented cross-cutting concerns.

5.9 Analysing execution traces with a webmining algorithm

Although this technique, presented by Zaidman and Demeyer [41] is not explicitly reported as an aspect mining technique, the application of webmining techniques in order to identify "important" classes in a system is certainly related to the other approaches discussed in this survey. Their approach is based on the HITS [22] webmining algorithm. This algorithm analyzes the hyperlinks in a number of webpages and identifies hubs (pages which refer to many other pages) and authorities (pages which are referred a from a large number of places). Zaidman et al. apply this algorithm to dynamic information obtained by executing (part of) the software system in which they want to identify the most important classes. As a result they obtain a collection of classes in the system which are tightly coupled. Since such classes play an important role in a software system, the authors argue that the identified classes are ideal candidates to start exploration of the system.

6 Criteria of Comparison

In Section 7 we will compare the different aspect mining approaches summarized in the previous section, based on a number of different criteria which are listed in this section.

Static versus dynamic. Does the technique take as input data which can be obtained by statically analysing the source code, or dynamic information which is obtained by executing the program, or both?

Lexical, structural and semantic What kind of semantic information does the applied technique reason about? We can distinguish three different semantic levels:

Lexical Lightweight reasoning about the program at a lexical level: sequences of characters, regular expressions, and so on.

Structural Reasoning takes the structure of the program into account, for instance, parse trees.

Semantical Reasoning about annotated parse trees or source code by taking into account the language semantics, for example: type checking, binding of variables and function names to their definitions, ...

Note that the boundaries between these levels is not always as clear as it depends strongly on the kind of information that is stored in the program.

Tangling and scattering Cross-cutting concerns are characterized by high tangling and scattering. *Scattering* means that the code corresponding to an aspect or crosscutting concern is dispersed across the entire system, instead of being located in a single module. *Tangling* means that concern code is often intermixed with that of other concerns. The studied techniques differ in whether they explicitly take scattering and/or tangling into account, or only implicitly (as a side-effect of the results produced by the tecnique).

Incrementality Whereas some techniques offer a one step mining process that tries to discover all possible aspects in a system at once other techniques support a more incremental process where aspects can be identified (and subsequently introduced) one at a time.

Scalability What is the size of systems that the technique can be applied on? For some techniques there may be an upper limit in order to still produce results in a reasonable amount of time, whereas other techniques may only work on systems that have at least some minimum size.

Symptoms What are the "symptoms of aspects" that the different techniques try to exploit in order to mine for aspects? For example, as cross-cutting concerns cannot be modularized cleanly using classic techniques, some developers rigorously use certain naming conventions for their crosscutting code. Code duplication is another symptom that may indicate the presence of some aspects. So are particular modes of invocation that are consistently used throughout the code. In general, occurrences of such pre-AOP aspect idioms are the basis of quite a number of different aspect mining techniques.

7 Taxonomy

Tables 2 and 3 compares the techniques we discussed in Section 5, based on the criteria described in Section 6. To win some space we abbreviated the names of the techniques used, as summarized by table 1.

Abbreviated name	Short description of the technique	Section
Execution patterns	Analyzing recurring patterns of execution traces	5.1
Dynamic analysis	Formal concept analysis of execution traces	5.2
Identifier analysis	Formal concept analysis of class and method names	5.3
Language clues	Natural language processing on source code	5.4
Unique methods	Detecting unique methods	5.5
Clustering	Hierarchical clustering of related methods	5.6
Fan-in analysis	Fan-in analysis	5.7
Clone detection	Detecting clones as indicators of crosscutting concerns	5.8
Web mining	Analysing execution traces with a webmining algorithm	5.9

Table 1: List of techniques that were compared

	static	dynamic	lexical	structural	semantic
Execution patterns	-	X	-	-	X
Dynamic analysis	-	X	-	-	X
Identifier analysis	X	-	X	-	-
Language clues	X	-	-	-	X
Unique methods	X	-	-	-	X
Clustering	X	-	X	-	X
Fan-in analysis	X	-	-	-	X
Clone detection	X	-	-	X	-
Web mining	-	X	-	-	X

Table 2: A taxonomy of different automated aspect mining techniques (1)

As can be seen in table 2, most of the existing aspect mining techniques focus on analyzing static information, i.e. the source-code of a system. While the majority of the approaches are based on discovering aspects from analyzing more semantical relations in a system, like for instance looking for cross-cutting idioms, high fan-in, recurring method invocations, ... there also exists techniques based on the observation that cross-cutting concerns are often characterized by duplicate code or rigorous use of coding and

	scattering	tangling	incremental	symptoms	scalability
Execution patterns	X	-	-	Recurring invocations	?
Dynamic analysis	-	X	X	Scat/Tang	?
Identifier analysis	X	-	-	Nam. Conv.	X?
Language clues	X	-	-	Nam. Conv.	-
Unique methods	X	-	X	Idioms	X/-
Clustering	X	-	-	Nam. Conv.	X?
Fan-in analysis	X	-	X	High Scat.	X
Clone detection	X	-	-	Code Dupl.	-
Web mining	X	-	-	High coupling	?

Table 3: A taxonomy of different automated aspect mining techniques (2)

naming conventions.

From table 3 we can conclude that all approaches, except Dynamic Analysis, try to identify aspects based on the assumption that cross-cutting concerns are often characterized by high levels of scattering. The fact however that cross-cutting concerns are often tangled with other concerns is taken only into account by the Dynamic Analysis approach by Ceccato and Tonella.

From the taxonomy, we distill a list of open research problems as well as a list of possible avenues for future research (point 7).

Common benchmark We noticed that it is quite impossible to compare the quality of the approaches we described above. Although some approaches provide a detailed analysis of their effectiveness, most techniques are rather presented as a proof-of-concept in which it is demonstrated that useful aspects are found. In order to get better insights into the strengths and weaknesses of every approach, it might be advisable to validate the different techniques on a common case-study. JHotDraw [5] seems to be a good candidate for being the common benchmark for aspect mining techniques. In fact, Ceccato et al. [12] describe an experiment in which they use this graphical framework in order to compare three different mining techniques: Formal Concept Analysis of Execution Traces, Fan-in analysis and Identifier Analysis using Formal Concept Analysis.

Scalability Complementary with the need for a common case-study is also the need to validate the different techniques on large-scale systems. Al-

though a number of approaches have been validated on real-life systems, most approaches describe the application of their technique on a small system. Due to the size of industrial legacy systems, it is however imperative that aspect mining techniques are proven to be capable of mining such systems.

Hybrid Techniques All techniques we discussed concentrate on identifying aspect candidates which exhibit one certain symptom which may indicate the presence of a cross-cutting concern. If we however wish to find all aspects in a system, this is a serious limitation. Using a hybrid approach which combines a number of techniques which concentrate on different kinds of aspect candidates may alleviate this problem.

Another possibility for hybrid techniques is looking at combinations of aspect mining techniques based on source-code and techniques that identify aspects during the early phases of the life cycle like requirements analysis [2, 32, 38] or architecture design [3].

8 Related research areas

The field of aspect mining is related to a number of other research fields, with which a cross-fertilization might be beneficial:

Data mining A number of the aspect mining approaches we discussed makes use of data mining algorithms like for instance cluster analysis, formal concept analysis, ... in order to propose the user a number of aspect candidates. This does not come as a surprise, as in the past data mining techniques have already been successfully applied on large-scale data sets in order to retrieve groups of elements which conceptually belong together. This research domain is however quite extensive, containing other approaches which may be ideal candidates for being used a the basis of an aspect mining technique.

Software Comprehension Aspect Mining is closely related to techniques which aid a developer in comprehending a piece of software. While the goal of Software Comprehension techniques is more general than identifying cross-cutting concerns in legacy code, the results obtained in this field can lead to interesting insights concerning aspect mining.

Program Analysis Over the years, a lot of program analysis techniques have been developed like metrics, clone detection, slicing,Ê... While

these techniques have already been used in some aspect mining approaches, it might be interesting to also consider other techniques from this research area.

Feature Exploration Greevy and Ducasse [14] present a metric-based approach for identifying features (i.e. user-triggered functional requirements) in a system. Although it is not the focus of the paper, the authors remark that their technique is a suitable candidate for identifying cross-cutting features.

9 Conclusion

In this paper we made a survey of semi-automated aspect mining techniques. Each of them has its own strengths and weaknesses because it relies on different assumptions and uses different underlying analysis techniques. Because of this there seems a need for a more in-depth comparison of the results obtained by each of the techniques, for example on a common benchmark. For example, many techniques seem, at least partly, complementary, which suggests the possibility of several useful combinations of existing techniques. Of course, the best validation of the suggested aspect mining techniques would be to show that the identified aspects can effectively be refactored into aspects. Therefore, it is essential to investigate how to move from aspect identification to the actual refactoring of non aspect-oriented code into aspects.

References

[1] B. Baker. On finding duplication and near-duplication in large software systems. In *2nd Working Conference on Reverse Engineering*, number 86-95. IEEE Computer Society Press, 1995.

[2] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 158–167, Washington, DC, USA, 2004. IEEE Computer Society.

[3] L. Bass, M. Klein, and L. Northrop. Identifying aspects using architectural reasoning. Position papers presented at Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design,

Workshop of the 3rd International Conference on Aspect-Oriented Software Development, (Lancaster, UK, 2004).

[4] I. Baxter, A. Yahin, L. Moura, M. Sant' Anna, and L. Bier. Clone detection using abstract syntax trees. In *International Conference on Software Maintenance*. IEEE Computer Society Press, 1998.

[5] J. Brant. Hotdraw. Master's thesis, University of Illinois, 1992.

[6] S. Breu. Towards hybrid aspect mining: Static extensions to dynamic aspect mining. In *1st Workshop on Aspect Reverse Engineering*, 2004.

[7] S. Breu and J. Krinke. Aspect mining using event traces. In *Conference on Automated Software Engineering (ASE)*, September 2004.

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

[9] M. Bruntink, A. v. Deursen, R. v. Engelen, and T. Tourwé. An evaluation of clone detection techniques for identifying crosscutting concerns. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. IEEE Computer Society Press, 2004.

[10] A. Budanitski. Semantic distance in wordnet: an experimental, application-oriented evaluation of five measures., 2001.

[11] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwe. A qualitative comparison of three aspect mining techniques. In *Proceedings of the 13th International Workshop on Program Comprehension (IWPC 2005)*, pages 13–22. IEEE Computer Society Press, 2005.

[12] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonello, and T. Tourwé. A qualitative comparison of three aspect mining techniques. In *International Workshop on Program Comprehension (IWPC)*, 2005.

[13] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Spring-Verlag, 1999.

[14] O. Greevy and S. Ducasse. Correlating features and code using a compact two-sided trace analysis approach. In *European Conference on Software Maintenance and Reengineering*, 2005. To appear in proceedings of the 9th European Conference on Software Maintenance and Reengineering.

[15] W. Griswold, Y. Kato, and J. Yuan. Aspect browser: Tool support for managing dispersed aspects. In *First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems - OOPSLA 99*, 1999.

[16] K. Gybels and A. Kellens. An experiment in using inductive logic programming to uncover pointcuts
an experiment in using inductive logic programming to uncover pointcuts. In *First European Interactive Workshop on Aspects in Software*, 2004.

[17] K. Gybels and A. Kellens. Experiences with identifying aspects in smalltalk using 'unique methods'. In *Workshop on Linking Aspect Technology and Evolution*, 2005.

[18] J. Hannemann and G. Kiczales. Overcoming the prevalent decomposition in legacy code. In *Workshop on Advanced Separation of Concerns, International Conference on Software Engineering*, 2001.

[19] L. He and H. Bai. Aspect mining using clustering analysis. Technical report, Jilin University, 2004.

[20] D. Janzen and K. De Volder. Navigating and querying code without getting lost. In *International Conference on Aspect Oriented Software Development 2003*, 2003.

[21] S. Karanjkar. Development of graph clustering algorithms. Master's thesis, University of Minnesota, 1998.

[22] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[23] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *8th International Symposium on Static Analysis*, pages 40–56. Springer-Verlag, 2001.

[24] J. Krinke. Identifying similar code with program dependence graphs. In *8th Working Conference on Reverse Engineering*, pages 301–309. IEEE Computer Society Press, 2001.

[25] J. Krinke and S. Breu. Control-flow-graph-based aspect mining. In *1st Workshop on Aspect Reverse Engineering*, 2004.

[26] R. Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming.* Manning Publications, 2003.

[27] M. Marin, A. van Deursen, and L. Moonen. Identifying aspects using fan-in analysis. In *Working Converence on Reverse Engineering (WCRE)*, 2004.

[28] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *International Conference on Software Maintenance*, pages 244–254. IEEE Computer Society Press, 1996.

[29] K. Mens, B. Poll, and S. González. Using intentional source-code views to aid software maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM'03)*, pages 169–178. IEEE Computer Society Press, 2003.

[30] K. Mens and T. Tourwé. Delving source-code with formal concept analysis. *Elsevier Journal on Computer Languages, Systems & Structures*, 2005. To appear.

[31] J. Morris and G. Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Comput. Linguist.*, 17(1):21–48, 1991.

[32] A. Rashid, P. Sawyer, A. M. D. Moreira, and J. Araújo. Early aspects: A model for aspect-oriented requirements engineerin. In *RE '02: Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, pages 199–202, Washington, DC, USA, 2002. IEEE Computer Society.

[33] M. P. Robillard and G. C. Murphy. Concern graphs: Fnding and describing concerns using structural program dependencies. In *Proceedings of the 24th International Conference on Software engineering*, pages 406–416. ACM Press, 2002.

[34] 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.

[35] D. Shepherd and L. Pollock. Interfaces, aspects and views. In *Linking Aspect Technology and Evolution (LATE) Workshop*, 2005.

[36] D. Shepherd, T. Tourwé, and L. Pollock. Using language clues to discover crosscutting concerns. In *Workshop on the Modeling and Analysis of Concerns*, 2005.

[37] P. Tarr, H. Ossher, W. Harrison, and J. S. M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *International Conference on Software Engineering*, 1999.

[38] B. Tekinerdogan and M. Aksit. Deriving design aspects from canonical models. In S. Demeyer and J. Bosch, editors, *Workshop Reader of the 12th European Conference on Object-Oriented Programming, ECOOP 1998*, Lecture Notes in Computer Science, pages 410–413. Springer-Verlag, 1998.

[39] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In *11th IEEE Working Conference on Reverse Engineering*, 2004.

[40] T. Tourwé and K. Mens. Mining aspectual views using formal concept analysis. In *Source Code Analysis and Manipulation Workshop (SCAM)*, 2004.

[41] A. Zaidman, T. Calders, S. Demeyer, and J. Paredaens. Applying web-mining techniques to execution traces to support the program comprehension process. In *8th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 329–338. IEEE Computing Society, 2004.

[42] C. Zhang and H. Jacobsen. Extended aspect mining tool. http://www.eecg.utoronto.ca/∼czhang/amtex.

[43] C. Zhang and H. Jacobsen. A prism for research in software modularization through aspect mining. Technical report, University of Toronto, 2003.