STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 4, 2012

ASPECT MINING. PAST, PRESENT, FUTURE

GRIGORETA S. COJOCAR

Abstract. Aspect mining is a research domain that tries to identify crosscutting concerns in already developed software systems. The goal is to refactor the analyzed system to use aspect oriented programming in order to ease the maintainability and evolution of the system. In this paper we briefly describe the aspect mining techniques proposed so far, we analyze them using three new criteria, and we discuss some possible future research directions considering the current state of the art.

1. Introduction

Ever increasing software systems made designing and implementing them a complex task. Software systems are composed of many different concerns, where a concern is a specific requirement or consideration that must be addressed in order to satisfy the overall system. The concerns are divided in core concerns and crosscutting concerns. The core concerns capture the central functionality of a module, while crosscutting concerns capture system-level, peripheral requirements that cross multiple modules. The current paradigms like procedural or object oriented programming provide good solutions for the design and implementation of core concerns, but they cannot deal properly with crosscutting concerns. Different approaches have been proposed for the design and implementation of crosscutting concerns: subject oriented programming [38], composition filters [1], adaptive programming [23], generative programming [10], aspect oriented programming (AOP) [19]. From these approaches, the aspect oriented programming approach has known the greatest success both in industry and academia.

In order to design and implement a crosscutting concern, AOP introduces four new concepts: *join point* (i.e., a well-defined point in the execution of a program), *pointcut* (i.e., groups a set of join points and exposes some of the values in the execution context of those join points), *advice* (i.e., a piece of

Received by the editors: November 24, 2012.

2010 *Mathematics Subject Classification.* 68N19, 68N99.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering*.

Key words and phrases. aspect mining, analysis, crosscutting concerns.

code that is executed at each join point in a pointcut), and a new modularization unit called *aspect*. The aspect is woven to generate the final system, using a special tool called *weaver*. Some of the benefits that the use of AOP brings to software engineering are: better modularization, higher productivity, software systems that are easier to maintain and to evolve. Nowadays, there are many programming language extensions to support AOP: AspectJ for Java [3], AspectC++ for C++ [2], etc.

For more than a decade researchers have tried to develop techniques and tools to (automatically) identify crosscutting concerns in already developed software systems, without using AOP. This area of research is called *Aspect Mining*. The goal is to identify the crosscutting concerns, and then to refactor them to aspects, in order to obtain a system that can be easily understood, maintained and modified.

In order to identify crosscutting concerns, the techniques try to discover one or both symptoms that appear when designing and implementing crosscutting concerns using the existing paradigms: code *scattering* and code *tangling*. Code scattering means that the code that implements a crosscutting concern is spread across the system, and code tangling means that the code that implements some concern is mixed with code from other (crosscutting) concerns.

Until now, many different approaches have been used for aspect mining, and different techniques have been proposed. In this paper we try to analyze the state of the art of aspect mining from the perspective of the proposed goal and to identify other possible future research directions in this field. The main contributions of this paper are to analyze the existing aspect mining techniques using three new criteria: industry usage, IDE integration and integration with AO refactoring, and to discuss possible new research directions in this field.

The paper is structured as follows. Section 2 presents an overview of the aspect mining techniques proposed so far. Section 3 analyzes the aspect mining techniques using three new different criteria. In Section 4 we discuss some possible future research directions considering the current state of the art of this field.

2. Overview of Aspect Mining Techniques

The first approaches in aspect mining were query-based search techniques. The developer had to introduce a so-called seed (eg., a word, the name of a method or of a field) and the associated tool showed all the places where the seed was found. Very soon, researchers discovered that this approach to aspect mining has some important disadvantages: the tool user had to have an in-depth knowledge of the analyzed system, as he/she had to figure out the seed(s) to be introduced, and the large amount of time needed in order to filter the results displayed. There are many query based aspect mining tools proposed: Aspect Browser [12], The Aspect Mining Tool(AMT) [13], Multi-Visualizer(AMTEX) [44], Feature Exploration and Analysis Tool(FEAT) [33], QJBrowser [31], JQuery [17], Prism [45] and Theme/Doc [4]. Except for the last one, all the other techniques are performing the search in the source code of the mined system. The Theme/Doc tool is searching for the seed in the requirements specifications.

Starting with 2004 researchers have focused on developing aspect mining techniques that do not require an initial seed from the user. These techniques try to identify the crosscutting concerns starting just from some kind of system representation (the source code, the requirements documentation, some execution traces, etc.), and are called automated aspect mining techniques. Different approaches were used: metrics, clustering, clone detection techniques, association rules, formal concept analysis, natural language processing, etc. In the following we briefly describe the automated aspect mining techniques proposed so far.

Marin et al. [26] have proposed an aspect mining technique that looks for methods that are called from many different call sites and whose functionality is needed across different methods, classes, and packages. The authors aim at finding such methods by computing the *fan-in* metric for each method using the static call graph of the system. Their approach relies on the observation that scattered, crosscutting functionality that largely affects the code modularity is likely to generate high fan-in values for key methods implementing this functionality.

Tonella and Ceccato [40] have proposed to use dynamic code analysis, feature location and formal concept analysis [11] for aspect mining, as follows. Execution traces are obtained by running an instrumented version of the program under analysis for a set of scenarios (use cases). The relationship between execution traces and executed computational units (methods) is subjected to concept analysis. The execution traces associated with the use-cases are the objects of the concept analysis context, while the executed methods are the attributes. In the resulting concept lattice, the concepts that satisfy both the scattering and the tangling conditions are considered as aspect candidates.

Breu and Krinke have proposed an aspect mining technique based on execution relations [6]. The proposed approach has two versions: a dynamic one [6] and a static one [20, 21]. They introduce the notion of *execution relation*, that describes the kind of relation that may exist between the executions of two methods. In the dynamic version the execution relations are extracted from program traces, and in the static version the execution relations are extracted from the control flow graph. They identify recurring execution patterns which describe certain behavioral aspects of the software system, and expect these patterns to be potential crosscutting concerns which describe recurring functionality in the program and thus are possible aspects. The authors have focused only on method executions as they wanted to analyze object-oriented systems where logically related functionality is encapsulated in methods.

Sampaio et al. [34] have proposed an approach for mining aspects from requirements related documents. Their approach builds upon the ideas of Theme/ Doc approach [4], but uses corpus-based natural language processing techniques in order to effectively enable the identification of aspects in semi-automated way. The main goal of their approach is to determine potential aspect candidates in requirements documents regardless of how they are structured (e.g., informal descriptions, interviews, structured documents).

Kim and Tourwé have proposed an aspect mining technique that relies on the assumption that naming conventions are the primary means for programmers to associate related but distant program entities [41]. Their technique tries to identify potential aspects and crosscutting concerns by grouping program entities with similar names. They apply formal concept analysis where the objects are all the classes and methods in the analyzed program and the attributes are the identifiers associated with those classes and methods. The authors chose for inspection only the groups that contain at least a given number of objects (a given threshold) and that are crosscutting (i.e., the involved methods and classes belong to at least two different class hierarchies).

Breu and Zimmermann tried to solve the problem of aspect mining taking a historical perspective [7]: they mine the history of a project (version archives) and identify code changes that are likely to be crosscutting concerns. Their analysis is based on the hypothesis that crosscutting concerns evolve within a project over time. A code change is likely to introduce such a concern if the modification gets introduced at various locations within a single code change.

Some authors tried to use clone detection techniques that aim at finding duplicated code, which may have been slightly adapted from the original. They base their research on the observation that typically source code implementing a crosscutting concern involves a great deal of duplications. Since the code belonging to a crosscutting concern cannot be cleanly captured inside a single abstraction, using the current programming paradigms, it cannot be reused. Therefore, developers are forced to write the same code over and over again, and are tempted to just copy paste the code and adapt it slightly to the context.

Shepherd et al [36] proposed the first automatic aspect mining technique based on clone detection. They based their analysis on AspectJ, particularly on the `before` advice. The technique consists in identifying initial refactoring candidates for the `before` advice using a control-based comparison, followed by filtering based on data dependence information. They used two types of clone detection techniques for identifying crosscutting concerns: PDG-based and AST-based clone detection techniques.

Bruntink et al. [9] tried to evaluate the usefulness and accuracy of clone detection techniques in aspect mining. The existing clone detectors usually produce output consisting of pairs of clones, i.e., they report which pairs of code fragments are similar enough to be called clones. The authors then investigate the groups of code fragments that are all clones of each other, called *clone classes*, in order to find aspect candidates.

Bruntink has extented the approach described in [9] considering metrics that grade the obtained clone classes [8]. The considered metrics were defined with the purpose of improving maintainability when aspects are used.

Orlando Mendez has also studied the applicability of clone detection techniques to aspect mining [30]. However, he used only one clone detector and applied it for one case-study.

Many authors have tried to use clustering for crosccutting concerns identification. *Clustering* is a division of data into groups of similar objects [5, 16]. Each group, called *cluster*, consists of objects that are similar between themselves and dissimilar to objects of other groups.

Shepherd and Pollock [37] used clustering to find methods with similar name as an indication of crosscuttingness. They perform agglomerative hierarchical clustering in order to group methods. The objects to be clustered are the names of the methods from the software system under analysis. The authors have developed a tool that helps users navigate and analyze the obtained clusters. The rest of the approach is just manual analysis of the obtained results using the tool.

Moldovan and Şerban have proposed a clustering based aspect mining approach that tries to discover crosscutting concerns by finding attributes of the *code scattering* symptom [28]. The authors use the vector space model based approach with two different vector space models, and different clustering algorithms (hard k-means clustering, fuzzy clustering, hierarchical agglomerative clustering, genetic clustering, etc) in order to group the methods from the software system into clusters.

Şerban and Moldovan also proposed an approach based on graph [35]. This approach is similar to the clustering one, but they use graphs, and in order to obtain a partition of the software system under analysis.

He and Bai [14] have proposed an aspect mining technique based on dynamic analysis and clustering that also uses association rules. They first use clustering to obtain crosscutting concern candidates and then use association rules to determine the position of the source code belonging to a crosscutting concern in order to ease refactoring. Execution traces are generated for an instrumented version of the software system, and for specified scenarios and inputs. Every scenario has a called-method sequence. If there exists a group of codes that has similar action, i.e., similar called-method sequence, and it frequently appears in execution traces, then a crosscutting concern may exist. Similar called-method sequences are considered possible crosscutting concerns code. Clustering analysis is used to find similar called-method sequences. The scenarios are the objects to be clustered, and the methods from the software systems are the attributes.

Maisikeli has proposed a dynamic aspect mining technique that uses a neural network clustering method called Self Organizing Map (SOM) [24]. He used a set of legacy benchmark programs to determine the most relevant software metrics that can be used for aspect mining (i.e., dynamic fan-in/fan-out, information flow, method spread, method cohesion contribution, etc.). The mined software system is executed to compute the value of these metrics. Based on these metrics he constructs a vector space model that is submitted as input to SOM for clustering. The results obtained by SOM are then manually analyzed to identify crosscutting concerns.

Rand Mcfadden has expanded the approach of Moldovan and Serban by using model based clustering [32]. She investigated the performance of different model based clustering algorithms with six vector space models (the two defined by Moldovan and Serban, and four new ones).

Vidal et al. have proposed another aspect mining technique based on dynamic analysis and association rule mining [42]. They execute the system using a set of scenarios in order to obtain execution traces. These execution traces are given as input to an association rule algorithm to find interesting associations among methods. The rules obtained are classified and filtered out in order to remove redundant rules or rules with utility methods.

Huang et al. have proposed an aspect mining technique inspired by the link analysis of information retrieval technology [15]. They try to discover the crosscutting concerns in the concern graphs extracted from the program using a two-state model. They compute the program elements that are in the *scatter* and *centralization* states, and use a ranking technique to select the crosscutting concerns candidates.

3. Analysis of Aspect Mining Techniques

Many different aspect mining techniques have been proposed so far, some of them in the last two years. However, if the techniques proposed in the beginning used very different apprroaches, the last ones (proposed in the last two-three years) are more an improvement of some of the previously proposed techniques. Even so, the results obtained by the new aspect mining techniques did not improve significantly. They obtained better results, but not much better.

In 2008, Mens et al. [27] have conducted an analysis of the problems the proposed aspect mining techniques were encountering. They have identified as main problems: poor precision, poor recall, subjectivity, scalability, lack of empirical validation. They have also identified the causes of these problems. In their opinion there are three main root causes: inappropriateness of the techniques used to mine for aspects, lack of a precise definition of what constitutes an aspect, and inadequate representation of the aspect mining results.

Even though the study conducted by Mens et al. describes most of the problems that exist in the aspect mining research field, there are other criteria that must also be considered when analyzing the aspect mining techniques, like the usage of aspect mining techniques in industry, integration of techniques with IDEs, and link of these techniques with aspect oriented refactoring tools. In the following we analyze the aspect mining techniques using these criteria.

3.1. **Aspect mining techniques used in industry.** Have any of the aspect mining techniques been used for complex projects? In Section 2 were briefly described the aspect mining techniques proposed so far. Most of the techniques used as case-studies JHotDraw version v5.4b1 [18] and Carla Laffra implementation of Dijkstra algorithm [22]. However, these case studies cannot be considered as complex. The former is a small to medium size software system, but the later case study is a small one consisting of only 6 classes. There are just a few case studies used (i.e., Tomcat, Eclipse v3.2M3) that can be considered as more complex. There are also no other reports or surveys describing the use of any of the aspect mining techniques for more complex software systems.

3.2. **IDE Integration.** Even though there are so many aspect mining techniques proposed, most of them cannot be used as there is no associated tool available. The only technique that can be used by others is the Fanin technique that has an Eclipse plugin available. The technique proposed by Vidal et al. [42] also described the use of an Eclipse-based tool, called AspectRT, however it is not publicly available. There are also a few techniques which can be recreated by following and using the same tools as the proponents of the techniques. However, for most of them, there is no tool available which makes it difficult for others to use them for other case studies or software systems.

3.3. **Integration with AO Refactoring.** There are a few reports available [39, 43] about using the results obtained by different aspect mining techniques in order to refactor the mined system to use aspect oriented constructs. For both reports the conclusion was that not all the crosscutting concerns that exist in a software system can be easily redesigned and implemented using AOP. For some crosscutting concerns, even the aspect oriented paradigm is not a good solution.

Some refactorings were proposed in order to ease the migration to an aspect oriented system [29]. However, except for the Vidal et al. technique, none of the techniques take into the consideration the subsequent refactoring step. This may be due to the facts that all the techniques still require a large amount of user involvement in order to analyze the results obtained by them, and that there are still a large number of false positives in the results presented. Some of the authors have considered refactoring the case studies used for aspect mining [25], however the approach used is mainly manually, without tool support.

4. Future of Aspect Mining

Considering the problems discovered by Mens et al. [27], and the additional criteria discussed in Section 3, it is very unlikely that any of the existing aspect mining techniques will be adopted by the industry in the near future. Without a major change in the approach used for aspect mining, the industry practitioners will not consider using an aspect mining technique. In the following we discuss some possible research directions, that might ease the adoption from industry.

4.1. **Top-down approaches.** In the beginning, the researchers have considered using a top-down approach for aspect mining. Using a catalog of known crosscutting concerns, these kind of approaches should focus on identifying the crosscutting concerns from the catalog. This may reduce execution time, the large number of false positives, and the user involvement in analyzing the obtained results.

4.2. **Identify only refactorable crosscutting concerns.** The case studies used for aspect mining and AO refactoring have already shown that only a susbset of the crosscutting concerns (CCCs) that exist in an object-oriented software system may be refactored into aspects. Future aspect mining techniques should focus on identifying only the refactorable crosscutting concerns. This may ease the integration of the next step: refactoring to use AOP. In this case, the techniques should also considered the right level of granularity for refactorable CCCs. The existing aspect mining techniques consider different levels of granularity: statement for clone-detection based techniques, methods for almost all techniques. However, there is still the question of which level is better: statement or method? If we consider only the refactorable CCCs, we should already know how they will be refactored, and it might help in identifying the granularity level used for mining CCCs.

4.3. **Create a catalogue of refactorable CCCs.** In order to identify the refactorable CCCs, we first need to know the crosscutting concerns that are refactorable into aspects. For that we need to create a catalogue. We may start by putting the most known crosscutting concerns that can be designed and implemented using AOP, like security, transaction management, logging and add new CCC as they appear in practice.

4.4. **Tool support.** It is very important that future aspect mining techniques consider developing an associated tool, that can be integrated with existing IDEs. Without such tools, the industry might not adopt/use the technique, even thought it may obtain good results.

5. Conclusions

In this paper we have have briefly described the existing aspect mining techniques, and, then, we have analyzed them using criteria like industry adoption, IDE integration, and subsequent refactoring. We have also discussed some future directions that should be considered for aspect mining.

Many different aspect mining techniques have been proposed so far, however case studies have shown that they do not perform very good: they have low precision, a large number of false positive, they still require a large amount of user involvement, and they cannot be integrated with refactoring tools.

Trying to discover all crosscutting concerns that exist in software systems is not suited for aspect mining and aspect oriented refactoring. Other approaches should be considered for aspect mining in order to be able to get near to one of the objective of aspect mining, that of refactoring the identified crosscutting concerns into aspects. Considering a top down approach, in the future, may be more efficient as it may reduce the number of results presented to the user, it may increase precision, it may decrease the time needed to identify the crosscutting concerns, and it may also make possible integration with refactoring tools.

References

[1] Mehmet Aksit. *On the Design of the Object Oriented Language Sina.* PhD thesis, Department of Computer Science, University of Twente, The Netherlands, 1989.

[2] AspectC++ Homepage. http://www.aspectc.org/.

[3] AspectJ Project. http://eclipse.org/aspectj/.

[4] Elisa Baniassad and Siobhán Clarke. Finding Aspects in Requirements with Theme/Doc. In *Proceedings of Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design*, Lancaster, UK, March 2004.

[5] Pavel Berkhin. Survey of Clustering Data Mining Techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[6] Silvia Breu and Jens Krinke. Aspect Mining Using Event Traces. In *Proceedings of International Conference on Automated Software Engineering (ASE)*, pages 310–315, 2004.

[7] Silvia Breu and Thomas Zimmermann. Mining Aspects from Version History. In Sebastian Uchitel and Steve Easterbrook, editors, *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*. ACM Press, September 2006.

[8] Magiel Bruntink. Aspect Mining Using Clone Class Metrics. In *Proceedings of the 2004 Workshop on Aspect Reverse Engineering (co-located with WCRE 2004)*, November 2004. Published as CWI technical report SEN-E0502, February 2005.

[9] Magiel Bruntink, Arie van Deursen, Remco van Engelen, and Tom Tourwé. On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.

[10] Krzysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[11] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[12] William G. Griswold, Yoshikiyo Kato, and Jimmy J. Yuan. AspectBrowser: Tool Support for Managing Dispersed Aspects. Technical Report CS1999-0640, UCSD, March 2000.

[13] Jan Hannemann and Gregor Kiczales. Overcoming the Prevalent Decomposition of Legacy Code. In *Advanced Separation of Concerns Workshop,at the International Conference on Software Engineering (ICSE)*, May 2001.

[14] Lili He and Hongtao Bai. Aspect Mining using Clustering and Association Rule Method. *International Journal of Computer Science and Network Security*, 6(2):247–251, February 2006.

[15] Jin Huang, Yansheng Lu, and Jing Yang. Aspect mining using link analysis. In *Proceedings of the 2010 Fifth International Conference on Frontier of Computer Science and Technology*, pages 312–317. IEEE Computer Society, 2010.

[16] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[17] Doug Janzen and Kris De Volder. Navigating and Querying Code Without Getting Lost. In *Proceedings of Aspect-Oriented Software Development*, pages 178–187, Boston, USA, 2003. ACM Press.

[18] JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[19] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume LNCS 1241, pages 220–242. Springer-Verlag, 1997.

[20] Jens Krinke. Mining control flow graphs for crosscutting concerns. In *13th Working Conference on Reverse Engineering: IEEE International Astrenet Aspect Analysis (AAA) Workshop*, pages 334–342, 2006.

[21] Jens Krinke and Silvia Breu. Control-Flow-Graph-Based Aspect Mining. In *Workshop on Aspect Reverse Engineering (WARE)*, 2004.

[22] Carla Laffra. Dijkstra's Shortest Path Algorithm.
http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/
DijkstraApplet.html.

[23] Karl J. Lieberherr. Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes. In J. van Leeuwen, editor, *Information Processing '92, 12th World Computer Congress*, pages 179–185, Madrid, Spain, 1992. Elsevier.

[24] Sayyed Garba Maisikeli. *Aspect mining using self-organizing maps with method level dynamic software metrics as input vectors*. PhD thesis, 2009.

[25] M. Marin. Refactoring JHotDraws Undo concern to Aspectj. In *Proceedings of the First Workshop on Aspect Reverse Engineering (WARE)*, 2004.

[26] Marius Marin, Arie van, Deursen, and Leon Moonen. Identifying Aspects Using Fan-in Analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004).*, pages 132–141. IEEE Computer Society, 2004.

[27] Kim Mens, Andy Kellens, and Jens Krinke. Pitfalls in Aspect Mining. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, WCRE '08, pages 113–122, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Grigoreta Sofia Moldovan and Gabriela Serban. Aspect Mining using a Vector-Space Model Based Clustering Approach. In *Proceedings of Linking Aspect Technology and Evolution (LATE) Workshop*, pages 36–40, Bonn, Germany, March, 20 2006. AOSD'06.

[29] M. P. Monteiro and J. M. Fernandes. Towards a catalog of aspect-oriented refactorings. In *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 111–122, 2005.

[30] Orlando Alejo Mendez Morales. Aspect Mining Using Clone Detection. Master's thesis, Delft University of Technology, The Netherlands, August 2004.

[31] Rajeswari Rajagopolan and Kris De Volder. A Query Based Browser Model. Master's thesis, University of British Columbia, Canada, July 2002.

[32] Renata Rand Mcfadden. *Aspect mining using model-based clustering*. PhD thesis, 2011. AAI3445077.

[33] Martin P. Robillard and Gail C. Murphy. Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 406–416, 2002.

[34] Américo Sampaio, Neil Loughran, Awais Rashid, and Paul Rayson. Mining Aspects in Requirements. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005)*, Chicago, Illinois, USA, 2005.

[35] Gabriela Serban and Grigoreta Sofia Moldovan. A Graph Algorithm for Identification of Crosscutting Concerns. *Studia Universitatis Babes-Bolyai, Informatica*, LI(2):53–60, 2006.

[36] David Shepherd, Emily Gibson, and Lori Pollock. Design and Evaluation of an Automated Aspect Mining Tool. In *2004 International Conference on Software Engineering and Practice*, pages 601–607. IEEE, June 2004.

[37] David Shepherd and Lori Pollock. Interfaces, Aspects, and Views. In *Proceedings of Linking Aspect Technology and Evolution Workshop(LATE 2005)*, March 2005.

[38] Subject oriented programming. http://www.research.ibm.com/sop/.

[39] Maximilian Störzer, Uli Eibauer, and Stefan Schöffmann. Aspect mining for aspect refactoring: An experience report. In *Towards Evaluation of Aspect Mining*, Nantes, France, July 2006. at ECOOP 2006.

[40] Paolo Tonella and Mariano Ceccato. Aspect Mining through the Formal Concept Analysis of Execution Traces. In *Proceedings of the IEEE Eleventh Working Conference on Reverse Engineering (WCRE 2004)*, pages 112–121, November 2004.

[41] Tom Tourwé and Kim Mens. Mining Aspectual Views using Formal Concept Analysis. In *SCAM '04: Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE*

International Workshop on (SCAM'04), pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

[42] Santiago Vidal, Esteban S. Abait, Claudia Marcos, Sandra Casas, and J. Andrés Díaz Pace. Aspect mining meets rule-based refactoring. In *Proceedings of the 1st Workshop on Linking Aspect Technology and Evolution*, PLATE '09, pages 23–27, New York, NY, USA, 2009. ACM.

[43] Isaac Yuen and Martin P. Robillard. Bridging the gap between aspect mining and refactoring. In *Proceedings of the 3rd workshop on Linking aspect technology and evolution*, LATE '07, New York, NY, USA, 2007. ACM.

[44] Charles Zhang, Gilbert Gao, and Arno Jacobsen. Multi Visualizer. http://www.eecg.utoronto.ca/ czhang/amtex/.

[45] Charles Zhang and Hans-Arno Jacobsen. PRISM is Research In aSpect Mining. In *OOPSLA*, Vancouver, British Columbia, Canada, 2004. ACM Press.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania

E-mail address: `grigo@cs.ubbcluj.ro`