Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 1 (March), pp. 72-80

Evaluation Measures for Partitioning based Aspect Mining Techniques

G. Czibula, G. S. Cojocar, I. G. Czibula

Gabriela Czibula, Grigoreta Sofia Cojocar, Istvan Gergely Czibula
Babeş-Bolyai University
1, M. Kogălniceanu Street, 400084, Cluj-Napoca, Romania
E-mail: {gabis, grigo, istvanc}@cs.ubbcluj.ro

Abstract: *Aspect mining* is a research direction that tries to identify crosscutting concerns in already developed software systems, without using aspect oriented programming. The goal is to identify them and then to refactor them to aspects, to achieve a system that can be easily understood, maintained and modified. In this paper we propose two new evaluation measures for evaluating the results of partitioning based aspect mining techniques. A small example on how to compute them is provided. The applicability of these measures to different aspect mining techniques is also discussed.
Keywords: partitioning, aspect mining, crosscutting concern, evaluation.

1 Introduction

Nowadays, software systems have become more and more complex and large. A software system is usually composed of many core concerns and (some) crosscutting concerns (like logging, exception handling). If core concerns can be cleanly separated and implemented using existing programming paradigms, this is not true for crosscutting concerns, as a crosscutting concern has a more system-wide behaviour that cuts across many of the core concerns implementation modules. The aspect oriented paradigm is one of the approaches proposed, so far, for the design and implementation of crosscutting concerns. Aspect oriented techniques allow crosscutting concerns to be implemented in a new kind of module called *aspect*, by introducing new language constructs like pointcuts and advices [13].

Kiczales et al. introduce for the first time aspect oriented programming (AOP) in [11]. Since 1997 the aspect oriented paradigm has been slowly adopted by the industry, too, leading to the appearance of new research problems like software reverse engineering, reengineering, and refactoring to use the aspect-oriented paradigm in order to benefit from the advantages it brings.

Aspect mining is a research direction that tries to identify crosscutting concerns in already developed software systems, without using AOP. The goal is to identify them and then to refactor them to aspects, to achieve a system that can be easily understood, maintained and modified. The task of crosscutting concerns identification cannot be successfully done using only a manual approach, as it is a difficult and error-prone process due to the complexity of software systems, their size, the lack of documentation, etc. As a consequence researchers have focused on developing tools and techniques that help developers to identify the crosscutting concerns in already developed software systems. The tools and techniques proposed, so far, try to discover the symptoms that an inadequate solution for a crosscutting concern implementation has over a software system: duplicated code, *scattering* of concerns throughout the entire system and *tangling* of concern-specific code with that of other concerns.

Although aspect mining is a relatively new research domain, many aspect mining techniques have been proposed. Some use metrics [14], some use formal concept analysis [2, 24, 25], or execution relations [1]. There are also a few approaches that use clone detection techniques

[3, 22] or natural language processing [18]. A few techniques use clustering in order to identify crosscutting concerns [8, 15, 20, 23].

There are very few comparisons made between the aspect mining techniques proposed so far [4, 10, 16, 17], and even less comparisons based on the obtained results [4, 17]. One important cause is the lack of measures for evaluating the results obtained and the quality of the results (i.e. how well did the technique manage in separating crosscutting concerns from non-crosscutting concerns, and in separating one crosscutting concern from other crosscutting concerns).

The main contribution of this paper is to propose two new evaluation measures for comparing partitioning based aspect mining techniques.

The paper is structured as follows. The context in which the measures are defined is introduced in Section 2. The new evaluation measures are defined in Section 3. A small example on how two compute the measure is given in Section 4. The applicability of the newly introduced measures is studied in Section 5. Some conclusions and further work are given in Section 6.

2 Formal Model

In [5] Cojocar and Şerban have proposed a formal model for partitioning based aspect mining. The measures that we proposed in this paper are based on their model. In the following we briefly introduce this model.

A software system S is viewed as a set of elements from the system: $S = \{s_1, s_2, \ldots, s_n\}$, where $s_i, 1 \leq i \leq n$. An *element* can be a statement, a method, a class, a module, etc. The number of elements of the system is denoted by n ($n = |S|$).

A crosscutting concern is considered as a set of elements $C \subset S$, $C = \{c_1, c_2, ..., c_m\}$, elements that implement this concern. CCC denotes the set of all crosscutting concerns that exist in the system S, $CCC = \{C_1, C_2, ..., C_q\}$, and q denotes the number of crosscutting concerns in the system S, $q = |CCC|$. It is considered that two different crosscutting concerns do not have elements in common, meaning that $C_i \cap C_j = \emptyset, \forall i, j, 1 \leq i, j \leq q, i \neq j$.

The problem of aspect mining is viewed as the problem of identifying a partition $\mathcal{K}$ of the software system S, such that $CCC \subset \mathcal{K}$.

Definition 1. Optimal partition of a system S.
Being given a partition $\mathcal{K} = \{K_1, K_2, ..., K_p\}$ of the system S, $\mathcal{K}$ is called an **optimal partition** of the system S with respect to the set $CCC = \{C_1, C_2, ..., C_q\}$ of all crosscutting concerns, iff:
(1) $p \geq q$;
(2) $\forall C \in CCC$, $\exists K_C \in \mathcal{K}$ such that $C = K_C$.

From the aspect mining point of view, $\mathcal{K}$ is an optimal partition of the system S if and only if the components of each crosscutting concern $C \in CCC$ are in the cluster K_C and K_C contains only the elements of C.

3 Evaluation measures

In this subsection we propose two measures for evaluating a partition of a software system from the aspect mining point of view. Such a partition can be obtained using a partitioning algorithm, such as a clustering algorithm.

In the following, let us consider a partition $\mathcal{K} = \{K_1, \ldots, K_p\}$ of a software system S and $CCC = \{C_1, C_2, ..., C_q\}$ the set of all crosscutting concerns from S. We assume that each crosscutting concern consists of a set of elements, i.e., $C_i = \{c_{i_1}, c_{i_2}, \ldots, c_{i_{m_i}}\}$.

Definitions 2 and 6 introduce evaluation measures for a partition of a software system from the aspect mining point of view.

Definition 2. Cohesion of Recovered Crosscutting Concerns - CORE.

Let $\mathcal{K}$ be a partition of a software system identified by a partitioning based aspect mining technique.

The cohesion of crosscutting concerns CCC recovered in partition $\mathcal{K}$, denoted by $CORE(CCC, \mathcal{K})$, is defined as: $CORE(CCC, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} core(C_i, \mathcal{K})$. $core(C_i, \mathcal{K})$ is the cohesion of crosscutting concern C_i in partition $\mathcal{K}$ and is defined as: $core(C_i, \mathcal{K}) = \frac{\sum_{k \in M_{C_i}} \frac{|C_i \cap k|}{|C_i \cup k|}}{|M_{C_i}|}$, where M_{C_i} is defined as: $M_{C_i} = \{k \mid k \in \mathcal{K},\ C_i \cap k \neq \emptyset\}$.

For a given crosscutting concern $C \in CCC$, $core(C, \mathcal{K})$ defines the degree to which the components of C belong together.

Lemma 3. *If $\mathcal{K}$ is a partition of the software system S and CCC is the set of crosscutting concerns in S, then the following inequality holds:* $0 \leq CORE(CCC, \mathcal{K}) \leq 1$.

For lack of space, we will not give the proof of Lemma 3.

Remark 4. Larger values for $CORE$ indicate better partitions with respect to CCC, meaning that $CORE$ has to be maximized.

In the following we give a necessary and sufficient condition for a partition $\mathcal{K}$ to be an **optimal partition**, with respect to the set of crosscutting concerns from the software system S.

Lemma 5. *If $\mathcal{K} = \{K_1, K_2, \ldots, K_p\}$ is a partition of the software system S, and CCC is the set of crosscutting concerns in S, then $\mathcal{K}$ is an **optimal partition** iff $CORE(CCC, \mathcal{K}) = 1$.*

For lack of space, we will not give the proof of Lemma 5.

The next measure determines the percentage of elements that must be analyzed in order to discover all the crosscutting concerns from the system. Usually, partitioning based aspect mining techniques return the clusters to be analyzed in a specific order.

Let σ be a permutation of the set $\{1,\ 2, \ldots\ p\}$. σ denotes the order in which the clusters from a partition of the software system are analyzed: $K_{\sigma(1)}$ is the first analyzed cluster, $K_{\sigma(2)}$ is the second, etc. The permutation σ is particular to each partitioning based aspect mining technique.

Definition 6. Complexity of Crosscutting Concerns Discovery - CODI. Let σ be a permutation of the set $\{1,\ 2, \ldots\ p\}$. The complexity of crosscutting concerns CCC discovery in partition K, denoted by $CODI(CCC, \mathcal{K}, \sigma)$, is defined as: $CODI(CCC, \mathcal{K}, \sigma) = \frac{1}{q} \sum_{i=1}^{q} Codi(C_i, \mathcal{K}, \sigma)$. $Codi(C_i, \mathcal{K}, \sigma)$ is the percentage of the elements that need to be analyzed in the partition $\mathcal{K}$ in order to discover the crosscutting concern C_i, and it is defined as: $Codi(C_i, \mathcal{K}, \sigma) = \frac{1}{m_i} \sum_{j=1}^{m_i} codi(c_{ij}, \mathcal{K}, \sigma)$. $codi(c_{ij}, \mathcal{K}, \sigma)$ is the percentage of the elements that need to be analyzed in the partition $\mathcal{K}$ in order to discover the element c_{ij} of crosscutting concern C_i, and it is defined as: $codi(c_{ij}, \mathcal{K}, \sigma) = \frac{1}{n} \sum_{l=1}^{r} |K_{\sigma(l)}|.$, where $r \in \{1,\ , 2, \ldots, p\}$ and it has the property that $c_{ij} \cap K_{\sigma(r)} \neq \emptyset$.

In our view $CODI(CCC, \mathcal{K}, \sigma)$ gives the complexity of crosscutting concerns discovery, and defines the percentage of the number of elements that need to be analyzed in the partition in order to discover all the crosscutting concerns that are in the system S. We consider that a crosscutting concern was discovered when all the elements that implement it were analyzed.

Lemma 7. *If $\mathcal{K}$ is a partition of the software system S, σ is a permutation of $\mathcal{K}$ and CCC is the set of crosscutting concerns in S, then the following inequality holds:* $0 < CODI(CCC, \mathcal{K}, \sigma) \leq 1$.

For lack of space, we will not give the proof of Lemma 7.

Remark 8. Smaller values for $CODI$ indicate shorter time for analysis, meaning that $CODI$ has to be minimized.

Based on the evaluation measures defined above, the comparison of the results obtained by different aspect mining techniques can be made from two different criteria:

1. **Partitioning**. The degree to which each crosscutting concern is well placed in the partition (using measure $CORE$).
2. **Ordering**. How relevant is the order in which the clusters are analyzed (using measure $CODI$).

In order to compare two partitions obtained by partitioning based aspect mining techniques, we introduce Definition 9. The definition is based on the properties of the evaluation measures defined above and considers both criteria presented above.

Definition 9. If $\mathcal{K}_1$ and $\mathcal{K}_2$ are two partitions of the software system S, CCC is the set of crosscutting concerns in S, and σ_1 and σ_2 are permutations of $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively, then $\mathcal{K}_1$ is **better** than $\mathcal{K}_2$ iff the following inequalities hold: $CORE(CCC, \mathcal{K}_1) \geq CORE(CCC, \mathcal{K}_2)$, $CODI(CCC, \mathcal{K}_1, \sigma_1) \leq CODI(CCC, \mathcal{K}_2, \sigma_2)$.

For the above definition we can remark the following:

Remark 10. If at least one of the inequalities from Definition 9 is not satisfied, we cannot decide which of the partitions $\mathcal{K}_1$ or $\mathcal{K}_2$ is better from the aspect mining point of view (considering simultaneously both criteria).

Remark 11. However, the importance of the above mentioned comparison criteria may depend on the user of the aspect mining technique. In our view, the most important criterion is **Partitioning** (how well the crosscutting concerns are grouped) and the last one is **Ordering** (how quickly the crosscutting concerns are discovered).

4 Example

In the following, a small example showing how to compute $CORE$ and $CODI$ measures is presented.

Let $S = \{s_1, s_2, ..., s_{17}\}$ be a software system with 17 elements, and let $C_1 = \{s_2, s_3, s_{17}\}$ and $C_2 = \{s_1, s_4, s_8\}$ be the crosscutting concerns that exist in the system S ($CCC = \{C_1, C_2\}$).

Let $\mathcal{K} = \{K_1, K_2, K_3, K_4, K_5\}$ be a partition of the software system S, where: $K_1 = \{s_2, s_{16}\}$; $K_2 = \{s_3, s_7, s_8, s_9, s_{17}\}$; $K_3 = \{s_1, s_4, s_5, s_{12}\}$; $K_4 = \{s_6, s_{10}, s_{13}\}$; $K_5 = \{s_{11}, s_{14}, s_{15}\}$.

CORE

Using Definition 2, we have to compute $core(C, \mathcal{K})$ for each $C \in CCC$.

C_1 First we have to determine the set M_{C_1}. It consists of two elements: $M_{C_1} = \{K_1, K_2\}$. The value of $core(C_1, \mathcal{K}) = \frac{1}{2} \sum_{k \in M_{C_1}} \frac{|C_1 \cap k|}{|C_1 \cup k|}$. We obtain that $core(C_1, \mathcal{K}) = \frac{1}{2}(\frac{|C_1 \cap K_1|}{|C_1 \cup K_1|} + \frac{|C_1 \cap K_2|}{|C_1 \cup K_2|}) = \frac{1}{2}(\frac{1}{4} + \frac{2}{6}) = \frac{7}{24}$.

C_2 The set M_{C_2} also consists of two elements: $M_{C_2} = \{K_2,\ K_3\}$. Based on Definition 2 $core(C_2, \mathcal{K}) = \frac{1}{2} \sum_{k \in M_{C_2}} \frac{|C_2 \cap k|}{|C_2 \cup k|}$. We obtain that $core(C_2, \mathcal{K}) = \frac{1}{2}(\frac{|C_2 \cap K_2|}{|C_2 \cup K_2|} + \frac{|C_2 \cap K_3|}{|C_2 \cup K_3|}) = \frac{1}{2}(\frac{1}{7} + \frac{2}{5}) = \frac{19}{70}$

Based on the Definition 2, $CORE(CCC, \mathcal{K}) = \frac{1}{2}(core(C_1, \mathcal{K}) + core(C_2, \mathcal{K})) = \frac{1}{2}(\frac{7}{24} + \frac{19}{70}) = \frac{473}{1680}$.

CODI

We consider the permutation σ in which the partition $\mathcal{K}$ is analyzed to be the identity permutation.

Using Definition 6, we have to compute $Codi(C, \mathcal{K}, \sigma)$ for each $C \in CCC$.

C_1 Based on the definition, $Codi(C_1, \mathcal{K}, \sigma) = \frac{1}{3}[codi(s_2, \mathcal{K}, \sigma) + codi(s_3, \mathcal{K}, \sigma) + codi(s_{17}, \mathcal{K}, \sigma)] = \frac{1}{3}[\frac{|K_1|}{17} + \frac{|K_1|+|K_2|}{17} + \frac{|K_1|+|K_2|}{17}] = \frac{1}{3}(\frac{2}{17} + \frac{2+5}{17} + \frac{2+5}{17}) = \frac{16}{51}$

C_2 Based on the definition, $Codi(C_2, \mathcal{K}, \sigma) = \frac{1}{3}[codi(s_1, \mathcal{K}, \sigma) + codi(s_4, \mathcal{K}, \sigma) + codi(s_8, \mathcal{K}, \sigma)] = \frac{1}{3}[\frac{|K_1|+|K_2|+|K_3|}{17} + \frac{|K_1|+|K_2|+|K_3|}{17} + \frac{|K_1|+|K_2|}{17}] = \frac{1}{3}(\frac{2+5+4}{17} + \frac{2+5+4}{17} + \frac{2+5}{17}) = \frac{29}{51}$

Based on the definition, $CODI(CCC, \mathcal{K}, \sigma) = \frac{1}{2}[Codi(C_1, \mathcal{K}, \sigma) + Codi(C_2, \mathcal{K}, \sigma)] = \frac{1}{2}(\frac{16}{51} + \frac{29}{51}) = \frac{45}{102}$.

For this example, an **optimal partition** is: $K_1 = \{s_6, s_9, s_{10}\}$; $K_2 = \{s_2, s_3, s_{17}\}$; $K_3 = \{s_{11}, s_{12}, s_{14}, s_{15}\}$; $K_4 = \{s_5, s_7, s_{13}, s_{16}\}$; $K_5 = \{s_1, s_4, s_8\}$.

5 Applicability

In this section we present some of the existing approaches in aspect mining. For each approach we analyze the applicability of the evaluation measures proposed in this paper.

Marin et al [14] have proposed an aspect mining technique that uses the *fanin* metric [9]. Their idea is to search for crosscutting concerns among the methods that have the value of the fanin metric greater than a given threshold. The result obtained by this technique can be viewed as a *partition* of the software system to be mined. The partition contains two clusters: the first one contains the methods that have the fanin greater than the given threshold and the second contains the remaining methods. So, the evaluation measures $CORE$ and $CODI$ can be applied for the result of this technique.

A graph based approach in Aspect Mining is introduced in [19]. The basic idea of this technique is to determine methods that are similar. The approach is to construct a graph between the methods of the software system, to determine the connex components of this graph, called *clusters*, and then to identify crosscutting concerns in the obtained clusters. As the set of connex components determined by this technique represents a *partition* of the analyzed software

system, the evaluation measures *CORE* and *CODI* can also be applied for the result of this technique.

There are a few aspect mining techniques proposed in the literature that use *clustering* in order to identify crosscutting concerns [8, 20, 23].

He and Bai [8] have proposed an aspect mining technique based on dynamic analysis. They obtain execution traces for each use case, but they apply clustering and association rules to discover aspect candidates.

Shepherd and Pollock [23] have proposed an aspect mining tool based on clustering. They use hierarchical clustering to find methods that have common substrings in their names. The obtained clusters are then manually analyzed to discover crosscutting concerns.

A clustering approach for identifying crosscutting concerns is proposed and a partitional clustering algorithm named *kAM* is introduced in [20].

An *evolutionary* approach in aspect mining is introduced in [21] and two *genetic clustering* algorithms used to identify crosscutting concerns are proposed. The clustering approach proposed in [21] is based on the use of genetic algorithms [6].

As all the above presented clustering techniques provide a *partition* of the software system, the applicability of *CORE* and *CODI* evaluation measures is assured.

There are in the literature some aspect mining techniques, briefly presented in the following, that do not provide a partition of the entire analyzed software system, but a subset of it. *CORE* and *CODI* measures cannot be applied for these techniques, as they require a partition of the entire system. However, the proposed measures can be extended in order to also consider these kind of situations. In the future we plan to tackle these particular cases.

Breu and Krinke [1] have proposed an aspect mining technique based on dynamic analysis. The mined software system is run and program traces are generated. From program traces, recurring execution relations that satisfy some constraints are selected. Among these recurring execution relations they search for aspect candidates. This approach is adapted to static analysis in [12]. In this approach the recurring execution relations are obtained from the control flow graph of the program.

Tonella and Ceccato [24] have also proposed an aspect mining technique based on dynamic analysis. An instrumented version of the mined software system is run and execution traces for each use case are obtained. Formal concept analysis [7] is applied on these execution traces and the concepts that satisfy some constraints are considered as aspect candidates.

Tourwé and Mens [25] have proposed an aspect mining technique based on identifier analysis. The identifiers associated with a method or class are computed by splitting up its name based on where capitals appear in it. They apply formal concept analysis on the identifiers to group entities with the same identifiers. The groups that satisfy some constraints and that contain a number of elements larger than a given threshold are considered as aspect candidates.

Bruntink et al [3] have studied the effectiveness of clone detection techniques in aspect mining. They did not propose a new aspect mining technique, but they tried to evaluate how useful clone detection techniques are in aspect mining.

Shepherd et al [22] have proposed an aspect mining technique based on clone detection. They search for code duplication in the source code using the program dependency graph. The obtained results are further analyzed to discover crosscutting concerns.

Breu and Zimmermann [2] have proposed an history based aspect mining technique. They mine CVS repositories for add-call transactions on which they apply formal concept analysis. Concepts that satisfy some constraints are considered aspect candidates.

Sampaio et al [18] have proposed an aspect mining technique to discover aspect candidates early in the development lifecycle. They use natural language processing techniques on different documents (requirements, interviews, etc.) to discover words that are used in many sentences.

The words that have a high frequency and have the same meaning in all the sentences are considered aspect candidates.

6 Conclusions and further work

In this paper we have proposed two new measures (*CORE* and *CODI*) for evaluating the results of partitioning based aspect mining techniques. We have also proved three important lemmas related to the proposed evaluation measures. A small example on how to compute these measures was provided. We have also discussed the applicability of these measures to different aspect mining techniques.

Further work may be done in the following directions: to adapt the measures to consider the case in which the crosscutting concerns have overlapping elements (the same element belongs to different crosscutting concerns); to evaluate some of the existing aspect mining techniques using the proposed measures; to extend the proposed evaluation measures in order to consider the situation in which the result obtained by an aspect mining technique is not a partition of the entire software system.

Acknowledgements

This work was supported by CNCSIS -UEFISCSU, project number PNII - IDEI 2286/2008.

Bibliography

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

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

[3] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwé. On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.

[4] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé. A Qualitative Comparison of Three Aspect Mining Techniques. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 13–22. IEEE Computer Society, 2005.

[5] G. S. Cojocar(Moldovan) and G. Serban. A Formal Model for Partitioning based Aspect Mining. *INFOCOMP Journal of Computer Science, Brazil*, 6(3):19–26, 2007.

[6] C. Cubillos, E. Urra and N. Rodríguez. Application of Genetic Algorithms for the DARPTW Problem. *International Journal of Computers, Communication and Control*, Vol. IV, No. 2:127-136, 2009.

[7] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

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

[9] B. Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[10] A. Kellens, K. Mens, and P. Tonella. A Survey of Automated Code-level Aspect Mining Techniques. *Transactions on Aspect-Oriented Software Development, Special Issue on Software Evolution*, VI(LNCS 4640):145–164, 2007.

[11] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings European Conference on Object-Oriented Programming*, volume LNCS 1241, pages 220–242. Springer-Verlag, 1997.

[12] J. 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.

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

[14] M. Marin, A. van, Deursen, and L. 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.

[15] G. S. Moldovan and G. 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.

[16] B. Nora, G. Said, and A. Fadila. A Comparative Classification of Aspect Mining Approaches. *Journal of Computer Science*, 2(4):322–325, 2006.

[17] C. K. Roy, M. G. Uddin, B. Roy, and T. R. Dean. Evaluating Aspect Mining Techniques: A Case Study. In *ICPC '07: Proceedings of the 15th IEEE International Conference on Program Comprehension*, pages 167–176, Washington, DC, USA, 2007. IEEE Computer Society.

[18] A. Sampaio, N. Loughran, A. Rashid, and P. 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.

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

[20] G. Serban and G. S. Moldovan. A New k-means Based Clustering Algorithm in Aspect Mining. In *Proceedings of 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*, pages 69–74, Timisoara, Romania, September, 26-29 2006. IEEE Computer Society.

[21] G. Serban and G. S. Moldovan. Aspect Mining using an Evolutionary Approach. *WSEAS Transactions on Computers*, 6(2):298–305, 2007.

[22] D. Shepherd, E. Gibson, and L. 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.

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

[24] P. Tonella and M. 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.

[25] T. Tourwé and K. 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.