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.
A few techniques use clustering in order to identify crosscutting concerns \[8,15,20,23\] or natural language processing \[18\]. 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 \(C \subset S, C = \{c_1, c_2, \ldots, c_m\}\), elements that implement this concern. \(CCC\) denotes the set of all crosscutting concerns that exist in the system \(S\), \(CCC = \{C_1, C_2, \ldots, C_q\}\), 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 \(K\) of the software system \(S\), such that \(CCC \subset K\).

#### Definition 1. Optimal partition of a system \(S\).

Being given a partition \(\mathcal{K} = \{K_1, K_2, \ldots, K_p\}\) of the system \(S\), \(\mathcal{K}\) is called an optimal partition of the system \(S\) with respect to the set \(CCC = \{C_1, C_2, \ldots, C_q\}\) 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 \(CCC = \{C_1, C_2, \ldots, C_q\}\) the set of all crosscutting concerns from \(S\). We assume that each crosscutting concern consists of a set of elements, i.e., \(C_i = \{c_{i_1}, c_{i_2}, \ldots, c_{im_i}\}\).

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 $\textit{CORE}(CCC, \mathcal{K})$, is defined as: 

$$\textit{CORE}(CCC, \mathcal{K}) = \frac{1}{q} \sum_{i=1}^{q} \textit{core}(C_i, \mathcal{K}).$$

$\textit{core}(C_i, \mathcal{K})$ is the cohesion of crosscutting concern $C_i$ in partition $\mathcal{K}$ and is defined as: 

$$\textit{core}(C_i, \mathcal{K}) = \frac{\sum_{k \in M_{C_i}} |C_i \cap k|}{|M_{C_i}|},$$

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 $C \in CCC$, $\textit{core}(C, \mathcal{K})$ 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: $0 \leq \textit{CORE}(CCC, \mathcal{K}) \leq 1$.

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

Remark 4. Larger values for $\textit{CORE}$ indicate better partitions with respect to $CCC$, meaning that $\textit{CORE}$ 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 $\textit{CORE}(CCC, \mathcal{K}) = 1$.

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 $\sigma$ be a permutation of the set $\{1, 2, \ldots, p\}$. $\sigma$ 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 $\sigma$ is particular to each partitioning based aspect mining technique.

Definition 6. Complexity of Crosscutting Concerns Discovery - CODI. Let $\sigma$ be a permutation of the set $\{1, 2, \ldots, p\}$. The complexity of crosscutting concerns $CCC$ discovery in partition $\mathcal{K}$, denoted by $\textit{CODI}(CCC, \mathcal{K}, \sigma)$, is defined as: 

$$\textit{CODI}(CCC, \mathcal{K}, \sigma) = \frac{1}{q} \sum_{i=1}^{q} \textit{Codi}(C_i, \mathcal{K}, \sigma).$$

$\textit{Codi}(C_i, \mathcal{K}, \sigma)$ 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: 

$$\textit{Codi}(C_i, \mathcal{K}, \sigma) = \frac{1}{m_i} \sum_{j=1}^{m_i} \textit{codi}(c_{ij}, \mathcal{K}, \sigma).$$

$\textit{codi}(c_{ij}, \mathcal{K}, \sigma)$ 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: 

$$\textit{codi}(c_{ij}, \mathcal{K}, \sigma) = \frac{1}{r} \sum_{l=1}^{r} |K_{\sigma(l)}|,$$

where $r \in \{1, 2, \ldots, p\}$ and it has the property that $c_{ij} \cap K_{\sigma(l)} \neq \emptyset$. 

In our view $\text{CODI}(\text{CCC}, \mathcal{K}, \sigma)$ 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 $S$, $\sigma$ is a permutation of $\mathcal{K}$ and $\text{CCC}$ is the set of crosscutting concerns in $S$, then the following inequality holds: $0 < \text{CODI}(\text{CCC}, \mathcal{K}, \sigma) \leq 1$.

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

**Remark 8.** Smaller values for $\text{CODI}$ indicate shorter time for analysis, meaning that $\text{CODI}$ 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 $\text{CORE}$).

2. **Ordering.** How relevant is the order in which the clusters are analyzed (using measure $\text{CODI}$).

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 $S$, $\text{CCC}$ is the set of crosscutting concerns in $S$, and $\sigma_1$ and $\sigma_2$ 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: $\text{CORE}(\text{CCC}, \mathcal{K}_1) \geq \text{CORE}(\text{CCC}, \mathcal{K}_2)$, $\text{CODI}(\text{CCC}, \mathcal{K}_1, \sigma_1) \leq \text{CODI}(\text{CCC}, \mathcal{K}_2, \sigma_2)$.

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 $\text{CORE}$ and $\text{CODI}$ measures is presented.

Let $S = \{s_1, s_2, \ldots, s_{17}\}$ 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 $S$ ($\text{CCC} = \{C_1, C_2\}$).

Let $\mathcal{K} = \{\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, \mathcal{K}_5\}$ be a partition of the software system $S$, where: $\mathcal{K}_1 = \{s_2, s_{16}\}$; $\mathcal{K}_2 = \{s_3, s_7, s_8, s_9, s_{17}\}$; $\mathcal{K}_3 = \{s_1, s_4, s_5, s_{12}\}$; $\mathcal{K}_4 = \{s_6, s_{10}, s_{13}\}$; $\mathcal{K}_5 = \{s_{11}, s_{14}, s_{15}\}$.
CORE

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

\( C_1 \) 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 \( \text{core}(C_1, K) = \frac{1}{2} \sum_{k \in M_{C_1}} \frac{|C_1 \cap k|}{|C_1 \cup k|} \). We obtain that \( \text{core}(C_1, K) = \frac{1}{2} \left( \frac{|C_1 \cap K_1|}{|C_1 \cup K_1|} + \frac{|C_1 \cap K_2|}{|C_1 \cup K_2|} \right) = \frac{1}{2} \left( \frac{1}{3} + \frac{2}{5} \right) = \frac{7}{25} \).

\( C_2 \) The set \( M_{C_2} \) also consists of two elements: \( M_{C_2} = \{K_2, K_3\} \). Based on Definition 2

\( \text{core}(C_2, K) = \frac{1}{2} \sum_{k \in M_{C_2}} \frac{|C_2 \cap k|}{|C_2 \cup k|} \) We obtain that \( \text{core}(C_2, K) = \frac{1}{2} \left( \frac{|C_2 \cap K_2|}{|C_2 \cup K_2|} + \frac{|C_2 \cap K_3|}{|C_2 \cup K_3|} \right) = \frac{1}{2} \left( \frac{2}{5} + \frac{2}{5} \right) = \frac{9}{20} \).

Based on the Definition 2, \( \text{CORE}(\text{CCC}, K) = \frac{1}{2}(\text{core}(C_1, K) + \text{core}(C_2, K)) = \frac{1}{2} \left( \frac{7}{25} + \frac{9}{20} \right) = \frac{473}{1680} \).

CODI

We consider the permutation \( \sigma \) in which the partition \( K \) is analyzed to be the identity permutation.

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

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

\( C_2 \) Based on the definition, \( \text{Codi}(C_2, K, \sigma) = \frac{1}{3}[\text{codi}(s_1, K, \sigma)+\text{codi}(s_4, K, \sigma)+\text{codi}(s_8, K, \sigma)] = \frac{1}{3}[\frac{|K_1|+|K_2|+|K_3|}{17} + \frac{|K_1|+|K_2|+|K_3|}{17}] = \frac{1}{3} \left( \frac{2+5+4}{17} + \frac{2+5+4}{17} + \frac{2+5+4}{17} \right) = \frac{20}{51} \).

Based on the definition, \( \text{CODI}(\text{CCC}, K, \sigma) = \frac{1}{2}[\text{Codi}(C_1, K, \sigma)+\text{Codi}(C_2, K, \sigma)] = \frac{1}{2} \left( \frac{16}{51} + \frac{20}{51} \right) = \frac{45}{102} \).

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 \textit{fanin} metric [9]. Their idea is to search for crosscutting concerns among the methods that have the value of the \textit{fanin} metric greater than a given threshold. The result obtained by this technique can be viewed as a \textit{partition} of the software system to be mined. The partition contains two clusters: the first one contains the methods that have the \textit{fanin} greater than the given threshold and the second contains the remaining methods. So, the evaluation measures \(\text{CORE}\) and \(\text{CODI}\) 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 \textit{clusters}, and then to identify crosscutting concerns in the obtained clusters. As the set of connex components determined by this technique represents a \textit{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


