Hierarchical Clustering for Identifying Crosscutting Concerns in Object Oriented Software Systems

ISTVAN GERGELY CZIBULA[1]
GABRIELA CZIBULA[1]
GRIGORETA SOFIA COJOCAR[1]

Babeş-Bolyai University
Department of Computer Science
1, M. Kogalniceanu Street 400084 - Cluj-Napoca, Romania
[1] (istvanc,gabis,grigo)@cs.ubbcluj.ro

Abstract. Crosscutting concerns are parts of a program that affect or crosscut other concerns. Usually these concerns cannot be cleanly decomposed from the rest of the system, and they are mixed with many core concerns from the system leading to code scattering and code tangling, and, also, to systems that are hard to explore and understand. Identifying crosscutting concerns automatically improves both the maintainability and the evolution of the software systems. Aspect mining is a research direction that tries to identify crosscutting concerns in already developed software systems, without using the aspect oriented paradigm. The goal is to identify them and then to refactor them to aspects, to obtain a system that can be easily understood, maintained and modified. In this paper we are focusing on the problem of identifying crosscutting concerns in object oriented software systems using a hierarchical agglomerative clustering approach. We experimentally validate our approach on the open source case study JHotDraw and on a real software system. A comparison of our approach with similar existing work is also provided.

Keywords: Aspect mining, crosscutting concern, clustering.

(Received May 21, 2009 / Accepted August 11, 2009)

1 Introduction

Separation of concerns [21] is a very important principle of software engineering that, in its most general form, refers to the ability to identify, encapsulate and manipulate those parts of software that are relevant to a particular concept, goal, or purpose.

Crosscutting concerns [12] are parts of a program which affect or crosscut other concerns. Usually these concerns cannot be cleanly decomposed from the rest of the system, and they are mixed with many core concerns from the system leading to code scattering and code tangling, and, also, to systems that are hard to explore and understand. Identifying crosscutting concerns automatically improves both the maintainability and the evolution of the software system. Crosscutting concerns are a relevant source of problems to program comprehension and software maintenance. Examples of crosscutting concerns are persistence, synchronization, exception handling, error management and logging.

The aspect oriented programming paradigm (AOP) is one of the approaches proposed, so far, for designing and implementing crosscutting concerns [12]. 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.

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. There exists many reasons for migrating a legacy system to an aspect oriented based system. An inadequate solution for crosscutting concerns implementation has a negative impact on the final system with consequences like duplicated code, *scattering* of concerns throughout the entire system and *tangling* of concern-specific code with that of other concerns. These consequences lead to software systems that are hard to maintain and to evolve. When aspect oriented techniques are used, the crosscutting concerns are cleanly separated from the core concerns, the latters becoming oblivious of them.

Unsupervised classification, or *clustering*, as it is more often referred as, is a data mining activity that aims to differentiate groups (classes or clusters) inside a given set of objects [7], being considered the most important *unsupervised learning* problem. The resulting subsets or groups, distinct and non-empty, are to be built so that the objects within each cluster are more closely related to one another than objects assigned to different clusters.

The main contribution of this paper is to introduce a hierarchical agglomerative clustering approach for identifying crosscutting concerns in existing software systems.

The rest of the paper is structured as follows. Section 2 presents some existing work in the field of *aspect mining*. A hierarchical agglomerative clustering approach for identifying crosscutting concerns is proposed in Section 3. An experimental evaluation of the proposed approach and a comparison of our approach with similar existing approaches is presented in Section 4. Section 5 contains some conclusions of the paper and also outlines further research directions.

2 Related Work

Aspect mining is a relatively new research domain. However, many aspect mining techniques have been proposed. Some use metrics [16], some use formal concept analysis [2, 27, 28], or execution relations [1]. There are also a few approaches that use clone detection techniques [3, 25] or natural language processing [22]. A few techniques use clustering in order to identify crosscutting concerns [8, 19, 24, 26].

In the following we will briefly present some of the existing approaches in aspect mining.

Marin *et al* [16] 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.

A graph based approach in aspect mining is introduced in [23]. 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.

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. 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 [14]. In this approach the recurring execution relations are obtained from the control flow graph of the program.

Tonella and Ceccato [27] 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 [6] is applied on these execution traces and the concepts that satisfy some constraints are considered as aspect candidates.

Tourwé and Mens [28] 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* [25] 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* [22] have proposed an aspect min-

ing 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.

There are a few aspect mining techniques proposed in the literature that use *clustering* in order to identify crosscutting concerns [8, 24, 26].

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 [26] 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 [24]. *kAM* algorithm is based on the idea of k-means clustering and uses an heuristic for choosing the initial centroids and the initial number of clusters. The similarity between two methods is computed using a vector space model based approach.

3 A Hierarchical Clustering Algorithm for Crosscutting Concerns Identification (HACO)

In this section we introduce a hierarhical agglomerative clustering algorithm (*HACO*) (Hierarchical Clustering Algorithm for Crosscutting Concerns Identification) for identifying crosscutting concerns in existing software systems. In order to discover the crosscutting concerns from the system, we first analyze the source code of the software system to be mined. All classes, methods and relations between them are computed. Afterwards, *HACO* algorithm is used to identify a **partition** of a software system S in which the methods belonging to a crosscutting concern should be grouped together. The final step is to manually analyze the obtained results.

Let us consider that the software system to be mined consists of a set of classes $\mathcal{CLS} = \{c_1, c_2, \ldots, c_s\}$, each class containing one ore more methods. In our clustering approach, the objects to be clustered are the methods from the software system S, i.e., $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$. Our focus is to group the methods such that the methods that belong to the same crosscutting concern are placed in the same cluster.

In order to apply a clustering approach for identifying the crosscutting concerns from a software system S, the dissimilarity degree between any two methods have to be considered. The idea is that methods from the same crosscutting concerns to have low dissimilarity values between them, and methods from different (crosscutting) concerns to have high dissimilarity values between them. We will consider the fact that crosscutting concerns in non AO systems have two symptoms: *code scattering* and *code tangling*. The *code scattering* symptom means that the code that implements a crosscutting concern is spread across the system. The *code tangling* symptom means that the code that implements some concern is mixed with code from other (crosscutting) concerns.

In the following we will define a distance function in order to express the dissimilarity degree between the methods from the software system from the aspect mining point of view. This means that methods from different crosscutting concerns have to be distant, and methods from the same crosscutting concern have to be close to each other.

For a given method m we denote by $\mathcal{C}(m)$ a collection consisting of: the method itself, the class in which the method is defined, the classes and methods that invoke m and the classes in which the classes and methods that invoke m are contained. The distance function that we propose will consider both the *scattering* and *tangling* symptoms. Consequently, we will consider the distance $d(m_i, m_j)$ between two methods m_i and m_j as expressed in Equation (1).

$$d(m_i, m_j) = \begin{cases} 0 & i = j \\ 1 - \frac{|\mathcal{C}(m_i) \cap \mathcal{C}(m_j)|}{|\mathcal{C}(m_i)| + |\mathcal{C}(m_j)|} & \text{if } \mathcal{C}(m_i) \cap \mathcal{C}(m_j) \neq \emptyset \\ \infty & otherwise \end{cases}, \tag{1}$$

In our view, the distance between two methods as defined in (1) expresses the following idea: if two methods are invoked by common methods or classes, they should belong to the same cluster. This means that scattered and tangled methods would be placed in the same cluster, and the methods that do not represent aspects would not be placed together with methods from aspects as the latters are invoked from multiple places. Consequently, methods that belong to the same crosccutting concern are close (considering distance d) to each other.

Many programming languages allow the definition of inner classes, classes that are defined inside of another class. The definition of classes inside of other classes are used in order to group classes that are related. This semantic information can be relevant for identifying crosscutting concerns. Grouping of classes frequently appears in real life software projects, and

our distance consider this situation, too. The collection $\mathcal{C}(m)$ contains not only the class c that invokes the method, but also the class that contains class c.

Based on the definition of distance d (Equation (1)) it can be easily proved that d is a semi-metric function, so a clustering approach can be applied.

HACO is based on the idea of hierarchical agglomerative clustering, and uses an heuristic for determining the number of clusters. In order to determine the number k of clusters, we are focusing on determining k representative methods from the software system S. The method chosen as the first representative method is the most "distant" method from the set of all methods (the method that maximizes the sum of distances from all the other methods). At each step we select from the remaining methods the most distant method relative to the already chosen methods. If the selected method is close enough to the already chosen representative methods, then the process is stopped, otherwise the selected method is considered as a new representative method. We consider a method close to a set of representative methods if the distances between the method and any of the representative methods from the considered set are less than a given threshold, $distMin$. We have chosen the value 0.75 for the threshold based on the following intuition: as distances greater than 1 are obtained only for unrelated methods (Equation (1)), the threshold value has to be less or equal to 1. The chosen value for the threshold was experimentally confirmed, but the most appropiate value for $distMin$ may depend on the analyzed system. In the future we plan to find the most appropriate value for the threshold $distMin$ using supervised learning techniques [17].

We have considered *complete link* [11] as linkage metric in the agglomerative hierarchical clustering process. The main steps of *HACO* algorithm are:

- Each method from the software system is put in its own cluster (singleton).
- The following steps are repeated until k clusters are reached (k is determined with the heuristic presented above):
  - Select the two most similar clusters K_i and K_j from the current partition, i.e, the pair of clusters that minimize the distance between them.
  - The clusters K_i and K_j will be merged, otherwise the partition remains unchanged.

HACO algorithm provides a partition of a software system S, partition that ideally would contain separate clusters for each crosscutting concern.

We mention that our aproach can be applied irrespective of the software system size, however the size influences the computational complexity of the technique.

4 Experimental Results

In this section we want to evaluate how well did *HACO* algorithm succeed in grouping the elements from crosscutting concerns in clusters.

In order to evaluate the results we use two quality measures, called *DISP* and *DIV*, that were previously introduced by Moldovan and Serban [18].

We give below the formal definition of *DISP* and *DIV* measure. In the following CCC denotes the set of crosscutting concerns existing in a software system, $\mathcal{K}$ denotes a partition of the set $\mathcal{M}$ of methods from the software system to be mined. The partition $\mathcal{K}$ can be obtained using a clustering algorithm (*HACO* in this paper).

Definition 1 [18] ***DISPersion of crosscutting concerns - DISP.***
The dispersion of the set CCC of crosscutting concerns in the partition $\mathcal{K}$, denoted by $DISP(CCC, \mathcal{K})$, is defined as

$$DISP(CCC, \mathcal{K}) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} disp(C_i, \mathcal{K}). \quad (2)$$

$disp(C, \mathcal{K})$ is the dispersion of a crosscutting concern C and is defined as:

$$disp(C, \mathcal{K}) = \frac{1}{|D_C|}, \quad (3)$$

where

$$D_C = \{k | k \in \mathcal{K} \text{ and } k \cap C \neq \emptyset\}. \quad (4)$$

D_C is the set of clusters that contain elements which are also in C.

$DISP$ measure takes values in [0, 1] and defines the dispersion degree of crosscutting concerns in clusters, considering, for each crosscutting concern, the number of clusters that contain elements belonging to the concern. Larger values for $DISP$ indicate better partitions with respect to set of the crosscutting concerns to be discovered, meaning that $DISP$ has to be maximized.

Definition 2 ***DIVersity of a partition - DIV.*** [18]
The diversity of a partition $\mathcal{K}$ with respect to the set CCC, denoted by $DIV(CCC, \mathcal{K})$, is defined as

$$DIV(CCC, \mathcal{K}) = \frac{1}{|\mathcal{K}|} \sum_{i=1}^{|\mathcal{K}|} div(CCC, K_i). \quad (5)$$

$div(CCC, k)$ is the diversity of a cluster $k \in \mathcal{K}$ and is defined as:

$$div(CCC, k) = \frac{1}{|V_k| + \tau(k)} \quad (6)$$

where

$$V_k = \{C | C \in CCC \text{ and } k \cap C \neq \emptyset\} \quad (7)$$

is the set of crosscutting concerns that have elements in k, and

$$\tau(k) = \begin{cases} 1 & \text{if } k \cap NCCC \neq \emptyset \\ 0 & \text{if } k \cap NCCC = \emptyset \end{cases} \quad (8)$$

.

$\tau(k)$ is 1 if the cluster k contains elements that do not implement any crosscutting concern, and 0 otherwise.

DIV measure takes values in [0, 1] and defines the degree to which each cluster contains elements from different crosscutting concerns or elements from other concerns. Larger values for DIV indicate better partitions with respect to set of the crosscutting concerns to be discovered, meaning that DIV has to be maximized.

Considering the fact that the user analyzes all the clusters from the software system, our approach does not obtain false negatives. The false positives are considered in the *DIV* measure.

4.1 A Simple Java Code Example

In this subsection we present a small example that shows how methods are grouped in clusters by *HACO* algorithm using distance d (Equation (1)). We have chosen the example below in order to provide the reader with an easy to follow example of crosscutting concerns identification.

Let us consider the Java code example given in Table 1.

For the code illustrated in Table 1, the set of crosscutting concerns is $CCC = \{C_1, C_2\}$, where $C_1 = \{L.m1, L.m2\}$ and $C_2 = \{A.mB, A.mC\}$. The methods $L.m1$ and $L.m2$ are called from two different contexts (methods $A.mA$ and $B.mC$) and they are mixed with other code. This is an example of code scattering (they are called from two different contexts) and code tangling (they are mixed with some other code). The same happens with methods $A.mB$ and $A.mC$. These methods are called only from one context, outside their class, but this example is very small, compared to a real application.

After applying *HACO* algorithm using distance d, the obtained clusters are shown in Table 2. Analyzing the results obtained for the Java code example described in Table 1, we conclude the folowing:

- *HACO* algorithm identifies the existing crosscutting concerns C_1 and C_2.
- The methods from each crosscutting concern, C_1 (i.e., $L.m1$ and $L.m2$) and C_2 (i.e, $A.mB$ and $A.mC$), are grouped together in separate clusters.
- The value of $DISP$ and DIV measures for the obtained partition are 1.

4.2 JHotDraw case study

In this subsection we consider the open source JHotDraw version 5.4b1 [5] case study for evaluating *HACO* algorithm. It is a Java GUI framework for technical and structured graphics, developed by Erich Gamma and Thomas Eggenschwiler, as a design exercise for using design patterns. It consists of **396** classes and **3359** methods.

The set of crosscutting concerns used for the evaluation is: *Adapter*, *Command*, *Composite*, *Consistent behavior*, *Contract enforcement*, *Decorator*, *Exception handling*, *Observer*, *Persistence*, and *Undo*. The set of crosscutting concerns and their implementing methods was constructed using the results reported by Marin *et al*. and publicly available at [15].

We have applied *HACO* algorithm for JHotDraw case study and we have obtain the following values: 0.457 for $DISP$ measure and 1 for DIV.

4.3 A real software system

In this section we present a real software system as a case study for evaluating *HACO* algorithm. We aim to observe how our technique performs for a real system. It is DICOM (*Digital Imaging and Communications in Medicine*) [4] and HL7 (*Health Level 7*) [10] compliant PACS (*Picture Archiving and Communications System*) system, facilitating the medical images management, offering quick access to radiological images, and making the diagnosing process easier.

The analyzed application is a large distributed system, consisting of several subsystems in form of standalone and web-based applications. We have applied *HACO* algorithm on one of the subsystems from this application.

The analyzed subsystem is a stand-alone Java application used by physicians in order to interpret radiological images. The application fetches clinical images from an image server (using DICOM protocol),

Table 1: Java Code example.

```
public class A {
 private int t;
 public A(){
  mB();
  t=new Random().nextInt();
  mC(); }
 public void mA(L l){
  l.m1();
  mB();
  l.m2(); }
 public void mB(){ }
 public void mC(){ }
}
public class L {
 public L(){}
 public void m1(){}
 public void m2(){}
}
public class B {
 private L l=new L();
 private A a=new A();
 public B(){
  a.mA();
  a.mB(); }
 public int mC(int nr){
  int sum=0;
  l.m1();
  for(int i=1; i<nr; i++) sum+=i;
  l.m2();
  return sum; }
 public double mD(int nr){
  a.mB();
  double rez=Math.sqrt(nr);
  a.mC();
  return rez; }
}
```

Table 2: The clusters obtained by *HACO* using distance d.

Cluster	Methods
C1	{ B.mD, B.mC, B.B }
C2	{ L.m1, L.m2 }
C3	{ A.mB, A.mC }
C4	{ A.A, A.mA, L.L }

displays them, and offers various tools to manage radiological images.

The analyzed application consists of **1015** classes and **8639** methods.

We mention that the set of crosscutting concerns was not apriori known for the considered software system. That is why, after an analysis of the system, we have identified the following set of crosscutting concerns and the methods that implement them: *Logging*, *Exception handling*, *Command*, *Persistence*, *Undo*, and *Consistent behavior*. Given the size of the analyzed system, some crosscutting concerns may have been missed.

After applying *HACO* algorithm, we have obtained the following results: 0.41 for $DISP$ measure and 0.92 for DIV.

Analyzing the obtained results, we have concluded that the obtained results provide a good starting point for extracting aspects from a software system. We have also identified possible improvements of our approach:

- Given the large number of methods from the system, preprocessing and postprocessing steps cand be added to our approach in order to facilitate easy interpretation of the obtained results and to reduce the computational complexity of the proposed technique.

- The accuracy of the obtained results are influenced by technical issues like: the use of anonymous inner classes, introspection, the use of dynamic proxies. These kind of technical aspects frequently appear in real life projects. In order to correctly deal with these aspects, we have to improve the way the data are collected from the software system in order to compute the distances between the methods (Equation 1).

4.4 Comparison with existing approaches

We have compared *HACO* algorithm with the approach from [24]. After applying *kAM* algorithm [24] for JHotDraw case study the value obtained for $DISP$ is 0.4005 and the value obtained for DIV is 0.9. Comparatively, considering the $DISP$ and DIV measures, *HACO* algorithm has obtained better results than *kAM* algorithm (0.457 for $DISP$ and 1 for DIV). This means that in the partition obtained by *HACO* algorithm the methods from the crosscutting concerns were better grouped than in the partition obtained by *kAM* algorithm.

We did not provide a comparison of our approach with the two other existing clustering based aspect mining approaches for the following reasons:

- Shepherd and Pollock have proposed in [26] an aspect mining tool based on clustering that does not automatically identify the crosscutting concerns. The user of the tool has to manually analyze the obtained clusters in order to discover crosscutting concerns.
- The technique proposed by He and Bai [8] cannot be reproduced, as they do not report neither the clustering algorithm used, nor the distance metric between the objects to be clustered. Also, the results obtained for the case study used by the authors for evaluation are not available.

The non-clustering aspect mining techniques cannot be compared with our approach because of the following reasons:

- some techniques are dynamic and they depend on the data used during executions [1, 27];
- for the static techniques [16] only parts of the results are publicly available;
- there is no case study used by all these techniques.

5 Conclusions and Future Work

We have approached in this paper the problem of identifying crosscutting concerns in existing software systems. We have introduced a hierarchical agglomerative clustering algorithm (*HACO*) that can be used for identifying aspects in object-oriented software systems.

In order to evaluate the obtained results, we have considered JHotDraw case study and a real software system. We have provided a comparison of our approach with similar existing approaches and we have identified possible improvements of our approach.

Further work can be done in the following directions:

- To use other machine learning techniques [17] in order to identify crosscutting concerns in existing software systems.
- To improve *HACO* algorithm by identifying the most appropriate heuristic to be used as stopping criterion in the hierarchical clustering process.
- To improve the distance semi-metric used for discriminating the methods in the clustering process.
- To apply *HACO* algorithm on other real software systems.
- To use other unsupervised learning techniques (self organizing maps [13], Hebbian learning [20]) for crosscutting concerns identifcation.

ACKNOWLEDGEMENT

This work was supported by the research project ID_2286, No. 477/2008, sponsored by the Romanian National University Research Council (CNCSIS).

References

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

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

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

[4] Digital Imaging and Communications in Medicine. http://medical.nema.org/.

[5] Gamma, E. JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[6] Ganter, B. and Wille, R. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. Translator-C. Franzke.

[7] Han, J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

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

[9] Henderson-Sellers, B. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[10] Health Level 7. www.hl7.org/.

[11] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

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

[13] Kohonen, T. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[14] Krinke, J. 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.

[15] Marin, M. Fan-in jhotdraw v5.4b1 results. http://swerl.tudelft.nl/bin/view/AMR/FanInAnalysisResults#JHotDraw_v_54b1.

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

[17] Mitchell, T. M. *Machine Learning*. McGraw-Hill, New York, 1997.

[18] Moldovan, G. and Serban, G. Clustering based aspect mining formalized. *WSEAS Transactions on Computers*, 6(2):199–206, 2007.

[19] Moldovan, G. S. and Serban, G. 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.

[20] O'Reilly, R. C. Generalization in interactive networks: The benefits of inhibitory competition and hebbian learning. *Neural Computation*, 13(6):1199–1241, 2001.

[21] Parnas, D. L. On the criteria to be used in decomposing systems into modules. *Communications of ACM*, 15(12):1053–1058, 1972.

[22] Sampaio, A., Loughran, N., Rashid, A., and Rayson, P. Mining Aspects in Requirements. In *Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005)*, Chicago, Illinois, USA, 2005.

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

[24] Serban, G. and Moldovan, G. S. 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.

[25] Shepherd, D., Gibson, E., and Pollock, L. Design and Evaluation of an Automated Aspect Mining Tool. In *2004 International Conference on Software Engineering and Practice*, pages 601–607. IEEE, June 2004.

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

[27] Tonella, P. and Ceccato, M. Aspect Mining through the Formal Concept Analysis of Execution Traces. In *Proc. of the 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 112–121, Washington, DC, USA, 2004. IEEE Computer Society.

[28] Tourwé, T. and Mens, K. Mining Aspectual Views using Formal Concept Analysis. In *Proc. IEEE International Workshop on Source Code Analysis and Manipulation*, 2004.