Mining Features from the Object-Oriented Source Code of Software Variants by Combining Lexical and Structural Similarity

R. AL-msie'deen, A.-D. Seriai, M. Huchard
LIRMM / CNRS & Montpellier 2 University, Montpellier, France
Al-msiedee, Abdelhak.Seriai, huchard@lirmm.fr

C. Urtado and S. Vauttier
LGI2P / Ecole des Mines d'Alès, Nîmes, France
Christelle.Urtado, Sylvain.Vauttier@mines-ales.fr

Abstract

Migrating software product variants which are deemed similar into a product line is a challenging task with main impact in software reengineering. To exploit existing software variants to build a software product line (SPL), the first step is to mine the feature model of this SPL which involves extracting common and optional features. Thus, we propose, in this paper, a new approach to mine features from the object-oriented source code of software variants by using lexical and structural similarity. To validate our approach, we applied it on ArgoUML, Health Watcher and Mobile Media software. The results of this evaluation showed that most of the features were identified[1].

***Keywords**: feature mining, software product variants, structural similarity, code dependencies, Formal Concept Analysis, Latent Semantic Indexing, Software Product Line.*

1 Introduction

A software product line (SPL) is a family of related program variants that share a common code base [5]. In the commonly encountered software development cycles, software product variants often evolve from an initial product developed for and successfully used by the first customers. When software variants become numerous, switching to a rigorous software product line engineering (SPLE) process is a solution to tame the increase in complexity of all engineering tasks. To switch to SPLE starting from a collection of existing variants, the first step is to mine a feature model that describes the SPL. This further implies to identify the software family's common and variable features. Manual reverse engineering of the feature model for the existing software variants is time-consuming, error-prone, and requires substantial effort [14]. Assistance tools may be of great interest in this task. In our previous work [2] we have presented an approach for feature location in a collection of software product variants based on Formal Concept Analysis (FCA). Our approach identifies functional features through the identification of their respective implementations which consist of object-oriented building elements (OBEs) of the source code such as *packages*, *classes*, *attributes*, *methods* or *method body elements* (*local variable*, *attribute access*, *method invocation*). Our hypothesis is that optional (*resp.* common) features appear in some but not all (*resp.* all) variants, thus they are implemented by OBEs that appear in some but not in all (*resp.* all) variants. Then our proposal consisted in dividing the OBE set in some specific subsets: The *Common Block* of OBEs (CB) and a set of blocks composed of variables OBE (*Blocks of Variations*, denoted as BVs). In a second work [1] we extended our first approach to gain more precision on the sets of OBEs candidates to be the implementations of features. We split the *common block* and the *blocks of variation* using lexical similarity between OBEs, through the Latent Semantic Indexing technique (LSI). In this paper, we investigate a new approach, inspired by our previous results, which consists in enhancing the use of FCA and LSI, with the use of structural dependencies between OBEs. We consider only variants whose variability is expressed by the existence or not of some packages and classes (*i.e.*, variability is expressed through packages and classes). Based on our previous experience on the case studies, we found that a feature was at most implemented at package or class levels (*e.g.*, ArgoUML-SPL and Mobile Media variants are expressed at package and class level respectively). Our approach is detailed in the remainder of this paper as follows.

[1]This work has been funded by grant ANR 2010 BLAN 021902

Section 2 briefly presents the background needed to understand the proposal. Section 3 shows an overview of our approach. Section 4 presents the feature mining process. Section 5 describes the experiments that were conducted to validate our proposal. Section 6 discusses the related work, while section 7 concludes and provides perspectives for this work.

2 Background

This section quickly introduces Formal Concept Analysis (FCA), Latent Semantic Indexing (LSI) and the structural dependencies between object-oriented building elements which we consider relevant for our approach.

2.1 Formal Concept Analysis (FCA)

Galois lattices and concept lattices [7] are core structures of a data analysis framework (Formal Concept Analysis) for extracting an ordered set of concepts from a dataset, called a formal context, composed of objects described by attributes. In our approach, we consider the AOC-poset (for Attribute-Object-Concept poset), which is the sub-order of the concept lattice restricted to object-concepts and attribute-concepts. AOC-posets scale much better than lattices. The interested reader can find more information about our use of FCA in [2].

2.2 Latent Semantic Indexing

LSI is an advanced Information Retrieval (IR) method. The heart of LSI is Singular Value Decomposition (SVD) technique. This technique is used to mitigate noise introduced by stop words (*e.g.*, ”the”, ”an”, ”above”) and to overcome two classical problems arising in natural language processing: synonymy and polysemy [9]. We chose LSI (see [1]) because it already had positive results in addressing maintenance tasks such as concept location [12] and recovery of traceability links between source code and documentation [10]. LSI assumes that all software artifacts are in textual format. Then, it computes the lexical similarity between two software artifacts based on the cosine similarity matrix (*cf.* Section 4.2.2). The interested reader can find more information about our use of LSI in [1].

2.3 Structural similarity

Using structural dependency information contained in source code has been proposed in [10] to increase the precision and recall of an IR method. When the candidate links were correct, then the dependency information could help locate additional correct links. Our process is based on the identification of object oriented building elements (OBEs) and their relationships which include inheritance, composition, invocation relationship, etc. We will use a coupling metric which measures the degree to which classes are linked to one another. Such coupling is an indication of the connections between elements of the object-oriented design [8]. It also measures the degree of interdependence and interaction between modules. Another definition is ”Coupling is a measure of the association, whether by inheritance or otherwise, between classes in a software product” [4]. Though coupling is a notion from structured design, it is still applicable to OO design at the levels of modules, classes and objects. We are concerned only with coupling between classes and we will consider these main dependencies [8] [4]:

1. **Inheritance coupling**: When a general class (superclass) is connected to its specialized classes (subclasses).
2. **Method invocation coupling**: When methods of one class use methods of another class.
3. **Composition coupling**: When an instance of one class is referred to in another class.
4. **Attribute access coupling**: When methods of one class use attributes of another class.
5. **Combined coupling**: It is the union of the other couplings (*i.e.*, two or more couplings) between two classes.

3 Approach Overview

This section provides the main concepts and hypotheses used in our approach for mining features from source code. It also shortly describes the example that illustrates the remaining of the paper.

3.1 Key ideas

In this paper we focus on the mining of functional features. We consider software systems in which functional features are implemented at the programming language level (*i.e.*, source code). We also restrict to OO software. Thus, features are implemented using object-oriented building elements (OBEs) and we restrict our study to packages and classes. We consider that a feature corresponds to one and only one set of OBEs. We also consider that feature implementations may overlap: a given OBE can be shared between several features’ implementation. In this paper we rely on structural similarity between OBEs to refine the splitting of blocks, which was done in our previous work only through lexical similarity. For two OBEs the structural similarity is based on coupling (*cf.* Section 2.3). Whether two OBEs (*i.e.*, classes) are structurally similar depends on the degree of coupling between these OBEs. Thus structural similarity between each pair of OBEs is computed based on the coupling measures.

3.2 Features versus Object-oriented Building Elements: the Mapping Model

Mining a feature from the source code of variants consists in identifying a candidate group of OBEs that constitutes its implementation. This group of OBEs must either be present in all variants (case of a common feature) or in some but not all variants (case of an optional feature). As the number of OBEs is large, mining features requires to reduce this search space. Our proposal consists in dividing the OBE set in specific subsets to obtain candidates for the *common feature set* – also called *common block* (CB) – and several *optional feature sets* (Block of Variations, denoted as BVs). Optional (*resp.* common) features appear in some but not all (*resp.* all) variants, they are implemented by OBEs that appear in some but not in all (*resp.* all) variants. Based on the lexical and structural similarity between the OBEs we identify atomic blocks of variation (ABV) (*i.e.*, optional feature) amongst each BV. A BV is thus composed of several ABVs. The same process applies for common features: we identify common atomic blocks (CAB) from CB based on the lexical and structural similarity between the OBEs. A CB is thus composed of several CABs. All concepts we defined for mining features are illustrated in the *OBE to feature* mapping model of Figure 1.

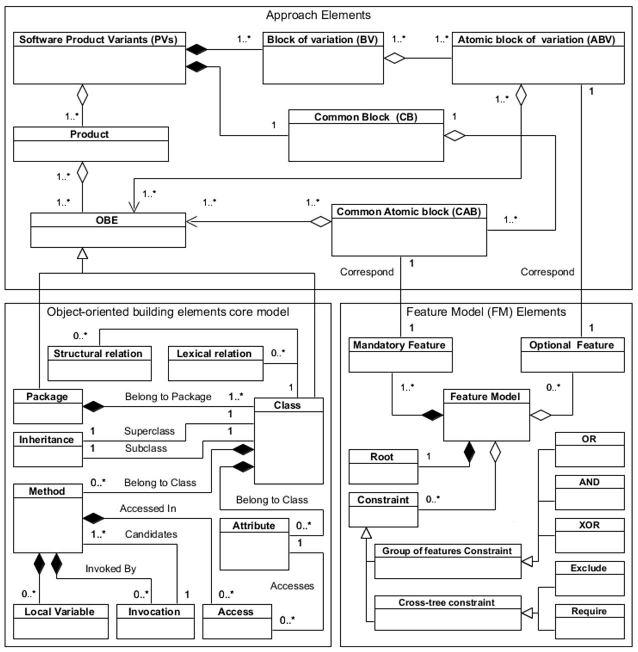

Figure 1. *OBE to Feature* **mapping model**

3.3 An Illustrative Example

As an illustrative example, we consider five drawing shapes software variants. This software allows a user to draw seven different kinds of shapes. Drawing shapes software variants represent a small case study (*e.g.* version 5 consists of 8 packages, 25 classes and 600 lines of code)[2].

4 The Feature Mining process

The mapping model between OBEs and features defines associations between these features and blocks of OBEs. The process takes the variants' source code as its input. The first step of this process aims at identifying BVs and the CB based on FCA (*cf.* Section 4.1). In the second step, we rely on structural similarity to determine the dependencies between OBEs (*cf.* Section 4.2.1). In the third step, we rely on LSI to determine the lexical similarity between OBEs (*cf.* Section 4.2.2). In the fourth step, we rely on lexical and structural similarity (*cf.* Section 4.2.3) to determine all possible similarity links between OBEs. Finally these similarity links are used to identify candidates atomic blocks based on OBE clusters (*cf.* Section 4.2.4). Figure 2 shows our feature mining process.

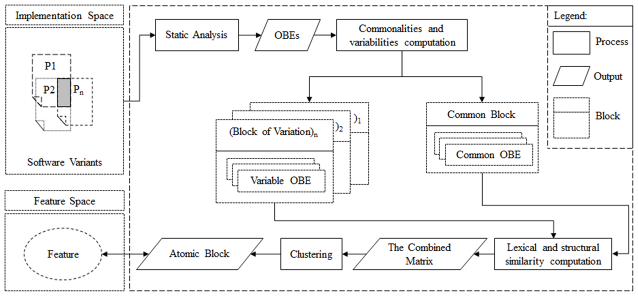

Figure 2. Feature mining process

4.1 Identifying the Common Block and Blocks of Variation

The technique used to identify the CB and BVs relies on FCA. First, a formal context where objects are product variants and attributes are OBEs is defined (*cf.* Table 1). The corresponding AOC-poset is then calculated. The intent of each concept represents OBEs common to two or more products. As concepts of AOC-posets are ordered, the intent of the most general (*i.e.*, top) concept gathers OBEs that are common to all products. They constitute the CB. The intents of all remaining concepts are BVs. They gather sets of OBEs common to a subset of products and correspond to the implementation of one or more features. The

[2]https://code.google.com/p/svariants/

extent of each of these concepts is the set of products having these OBEs in common (*cf.* Figure 3).

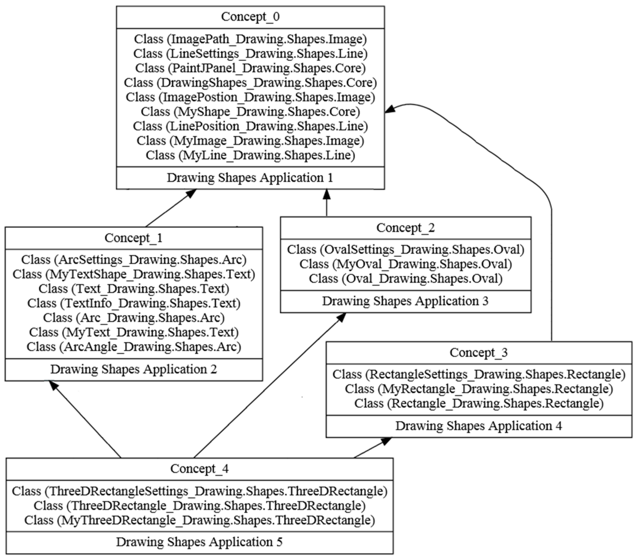
Concept_0
Class (ImagePath_Drawing.Shapes.Image)
Class (LineSettings_Drawing.Shapes.Line)
Class (PaintJPanel_Drawing.Shapes.Core)
Class (DrawingShapes_Drawing.Shapes.Core)
Class (ImagePostion_Drawing.Shapes.Image)
Class (MyShape_Drawing.Shapes.Core)
Class (LinePosition_Drawing.Shapes.Line)
Class (MyImage_Drawing.Shapes.Image)
Class (MyLine_Drawing.Shapes.Line)
Drawing Shapes Application 1
Concept_1
Class (ArcSettings_Drawing.Shapes.Arc)
Class (MyTextShape_Drawing.Shapes.Text)
Class (Text_Drawing.Shapes.Text)
Class (TextInfo_Drawing.Shapes.Text)
Class (Arc_Drawing.Shapes.Arc)
Class (MyText_Drawing.Shapes.Text)
Class (ArcAngle_Drawing.Shapes.Arc)
Drawing Shapes Application 2
Concept_2
Class (OvalSettings_Drawing.Shapes.Oval)
Class (MyOval_Drawing.Shapes.Oval)
Class (Oval_Drawing.Shapes.Oval)
Drawing Shapes Application 3
Concept_3
Class (RectangleSettings_Drawing.Shapes.Rectangle)
Class (MyRectangle_Drawing.Shapes.Rectangle)
Class (Rectangle_Drawing.Shapes.Rectangle)
Drawing Shapes Application 4
Concept_4
Class (ThreeDRectangleSettings_Drawing.Shapes.ThreeDRectangle)
Class (ThreeDRectangle_Drawing.Shapes.ThreeDRectangle)
Class (MyThreeDRectangle_Drawing.Shapes.ThreeDRectangle)
Drawing Shapes Application 5

Figure 3. The AOC-poset for the Formal Context of Table 1

4.2 Mining Features from CB and BVs

The CB and BVs might implement several features. Identifying the OBEs that characterize a feature's implementation thus consists in splitting the CB and the BVs in smaller sets that we call atomic blocks. Atomic blocks are identified based on the calculation of the similarity between the OBEs of a block. In this paper we consider lexical and structural similarity. Atomic blocks are clusters of the most similar OBEs. These clusters are built with FCA as detailed in the following.

4.2.1 Measuring OBEs' Similarity Based on Structural Dependency

Structural similarity is used to capture and represent the dependencies between classes of a block. We use a Dependency Structure Matrix (DSM), which is a square matrix in which the classes being analyzed correspond to the rows and columns. An entry in the matrix indicates that the class on the corresponding column depends on the class on the corresponding row. Dependencies on the diagonal, from upper left to lower right, are not of interest because they would only indicate that an item depends on itself. In DSM, a character '$\times$' means that a dependency exists (*cf.* Table 2). We used a class DSM to represent the dependency among classes in the common block and inside each block of variation. We use the structural relations that are introduced in Section 2.3. For example, in the dependency structure matrix (*cf.* Table 2), the OBE "Class (ArcSettings_Drawing.Shapes.Arc)" is linked to the OBE "Class (ArcAngle_Drawing.Shapes.Arc)" because there is a structural link between these two classes (*i.e.*, inheritance coupling). However, the OBE "Class (Text_Drawing.Shapes.Text)" and the OBE "Class (ArcAngle_Drawing.Shapes.Arc)" are not linked because there is no structural link between these two classes.

Table 2. DSM of Concept_1

	Class (ArcSettings_Drawing.Shapes.Arc)	Class (Arc_Drawing.Shapes.Arc)	Class (ArcAngle_Drawing.Shapes.Arc)	Class (MyTextShape_Drawing.Shapes.Text)	Class (Text_Drawing.Shapes.Text)	Class (TextInfo_Drawing.Shapes.Text)	Class (MyText_Drawing.Shapes.Text)
Class (ArcSettings_Drawing.Shapes.Arc)		×	×				
Class (Arc_Drawing.Shapes.Arc)	×		×				
Class (ArcAngle_Drawing.Shapes.Arc)	×	×					
Class (MyTextShape_Drawing.Shapes.Text)					×		
Class (Text_Drawing.Shapes.Text)				×		×	×
Class (TextInfo_Drawing.Shapes.Text)					×		
Class (MyText_Drawing.Shapes.Text)					×		

4.2.2 Measuring OBEs' Similarity Based on LSI

In order to apply LSI, we build a corpus that represents a collection of documents and queries. In our case, each class in CB and BVs represents both a document and a query. To be processed, the document and query must be normalized (*e.g.*, all capitals turned into lower case letters, articles, punctuation marks or numbers removed). The normalized document generated by analyzing the source code of a class is split into terms and, at last, word stemming is performed. To create a document for each class, we must consider all useful information that describes the class (*i.e.*, package name, class name, attributes names, methods names and method body elements names, *e.g.*, parameter name, local variable name, method invocation name, attribute access name). The most important parameter of LSI is the number of chosen term-topics. A term-topic is a collection of terms that co-occur frequently in the documents of the corpus. We need enough term-topics to capture real term relations. In our work we cannot use a fixed number of topics for LSI because we have blocks of variation (*i.e.*, partitions) with different sizes. The number of term-topics (# term-topics) is equal to "$K * N$", where K is a variable, its value depends on the size of each BV and N is the number of columns of the term-document matrix that is generated by LSI [1]. To compute similarity between each pair of OBEs in the CB and BVs and produce Lexical Similarity Matrix (LSM), we

Table 1. A formal context describing drawing shapes software variants

	Class (PaintJPanel_Drawing.Shapes.Core)	Class (DrawingShapes_Drawing.Shapes.Core)	Class (MyShape_Drawing.Shapes.Core)	Class (LineSettings_Drawing.Shapes.Line)	Class (LinePosition_Drawing.Shapes.Line)	Class (MyLine_Drawing.Shapes.Line)	Class (ImagePath_Drawing.Shapes.Image)	Class (MyImage_Drawing.Shapes.Image)	Class (ImagePostion_Drawing.Shapes.Image)	Class (ArcSettings_Drawing.Shapes.Arc)	Class (Arc_Drawing.Shapes.Arc)	Class (ArcAngle_Drawing.Shapes.Arc)	Class (MyTextShape_Drawing.Shapes.Text)	Class (Text_Drawing.Shapes.Text)	Class (TextInfo_Drawing.Shapes.Text)	Class (MyText_Drawing.Shapes.Text)	Class (OvalSettings_Drawing.Shapes.Oval)	Class (Oval_Drawing.Shapes.Oval)	Class (MyOval_Drawing.Shapes.Oval)	Class (RectangleSettings_Drawing.Shapes.Rectangle)	Class (MyRectangle_Drawing.Shapes.Rectangle)	Class (Rectangle_Drawing.Shapes.Rectangle)	Class (ThreeDRectangleSettings_Drawing.Shapes.ThreeDRectangle)	Class (ThreeDRectangle_Drawing.Shapes.ThreeDRectangle)	Class (MyThreeDRectangle_Drawing.Shapes.ThreeDRectangle)
Drawing Shapes Software 1	×	×	×	×	×	×	×	×	×																
Drawing Shapes Software 2	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×									
Drawing Shapes Software 3	×	×	×	×	×	×	×	×	×								×	×	×						
Drawing Shapes Software 4	×	×	×	×	×	×	×	×	×											×	×	×			
Drawing Shapes Software 5	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

proceed in two steps: building the cosine similarity matrix, transform cosine similarity matrix into the Lexical Similarity Matrix. The interested reader can find more information about these two steps in [1].

Cosine similarity matrix: Similarity between OBEs is described by a cosine similarity matrix [9] which columns and rows both represent vectors of OBEs: documents as columns and queries as rows. In our work, we consider the most widely used threshold for cosine similarity to be 0.70 [9]. Cosine similarity matrix is a numerical matrix encoding similarity. As an example, in the cosine similarity matrix, the OBE "Class (ArcSettings_Drawing.Shapes.Arc)" is similar to the OBE "Class (ArcAngle_Drawing.Shapes.Arc)" because their similarity value is 0.97, which is greater than the threshold. However, the OBE "Class (Text_Drawing.Shapes.Text)" and the OBE "Class (ArcSettings_Drawing.Shapes.Arc)" are not linked because their similarity is 0.12, thus less than the threshold. In this example, the *K* value for this BV (*i.e.*, Concept_1) is equal to 0.29 (*i.e.*, the number of topics in this block is equal to 2).

Transform cosine similarity matrix into the LSM: LSM is a square matrix (*cf.* Table 3) where each entry $c_{i,j}$ represents a lexical similarity between class i and class j higher than a chosen threshold (here 0.70). The diagonal entries ($c_{i,i}$) always have value '$\times$' to indicate that a class is similar to itself. We used class LSM to represent the lexical similarity between classes in the CB and for each of the BVs. Table 3 shows the formal context (*i.e.*, LSM) obtained by transforming the similarity matrix corresponding to the BV of Concept 1 from Figure 3.

Table 3. LSM of Concept_1

	Class (ArcSettings_Drawing.Shapes.Arc)	Class (Arc_Drawing.Shapes.Arc)	Class (ArcAngle_Drawing.Shapes.Arc)	Class (MyTextShape_Drawing.Shapes.Text)	Class (Text_Drawing.Shapes.Text)	Class (TextInfo_Drawing.Shapes.Text)	Class (MyText_Drawing.Shapes.Text)
Class (ArcSettings_Drawing.Shapes.Arc)	×	×	×				
Class (Arc_Drawing.Shapes.Arc)	×	×	×				
Class (ArcAngle_Drawing.Shapes.Arc)	×	×	×				
Class (MyTextShape_Drawing.Shapes.Text)				×	×	×	×
Class (Text_Drawing.Shapes.Text)				×	×	×	×
Class (TextInfo_Drawing.Shapes.Text)				×	×	×	×
Class (MyText_Drawing.Shapes.Text)				×	×	×	×

4.2.3 Measuring OBEs' Similarity Based on Lexical and Structural Similarity

To combine both lexical and structural similarity between OBEs in the common block or blocks of variation we introduce what we call a combined matrix. A combined matrix (CM) is a square matrix which integrates the previous two matrices. In other word, this matrix represents both DSM and LSM between a set of classes. The CM is an adjacency matrix where a cell represents a link between two classes based on the structural or lexical similarity ((*cf.* Table 4)). All possible links between OBEs are considered in this matrix. The combined matrix of Table 4 also constitutes the formal context which is used as input for applying FCA in the next step.

Table 4. CM of Concept_1

	Class (ArcSettings_Drawing.Shapes.Arc)	Class (Arc_Drawing.Shapes.Arc)	Class (ArcAngle_Drawing.Shapes.Arc)	Class (MyTextShape_Drawing.Shapes.Text)	Class (Text_Drawing.Shapes.Text)	Class (TextInfo_Drawing.Shapes.Text)	Class (MyText_Drawing.Shapes.Text)
Class (ArcSettings_Drawing.Shapes.Arc)	×	×	×				
Class (Arc_Drawing.Shapes.Arc)	×	×	×				
Class (ArcAngle_Drawing.Shapes.Arc)	×	×	×				
Class (MyTextShape_Drawing.Shapes.Text)				×	×	×	×
Class (Text_Drawing.Shapes.Text)				×	×	×	×
Class (TextInfo_Drawing.Shapes.Text)				×	×	×	×
Class (MyText_Drawing.Shapes.Text)				×	×	×	×

4.2.4 Identifying Features Using FCA

We then use FCA to identify, from each BV and from the CB, OBE sets the elements of which are similar based on lexical and structural similarity. The interest of FCA for this task is to help extracting concepts which represent mutually similar OBEs. For the drawing shapes example, the AOC-poset of Figure 4 shows two atomic blocks of variation (that correspond to two distinct features) mined from a single block of variation (Concept 1 from Figure 3). The same feature mining process is used for the CB and for each of the BVs.

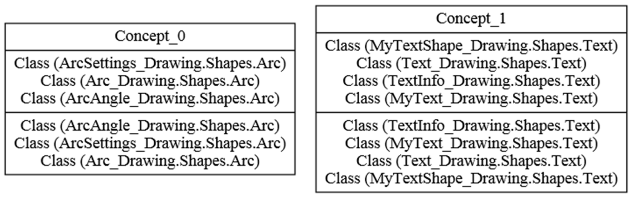
Concept_0
Class (ArcSettings_Drawing.Shapes.Arc)
Class (Arc_Drawing.Shapes.Arc)
Class (ArcAngle_Drawing.Shapes.Arc)
Class (ArcAngle_Drawing.Shapes.Arc)
Class (ArcSettings_Drawing.Shapes.Arc)
Class (Arc_Drawing.Shapes.Arc)
Concept_1
Class (MyTextShape_Drawing.Shapes.Text)
Class (Text_Drawing.Shapes.Text)
Class (TextInfo_Drawing.Shapes.Text)
Class (MyText_Drawing.Shapes.Text)
Class (TextInfo_Drawing.Shapes.Text)
Class (MyText_Drawing.Shapes.Text)
Class (Text_Drawing.Shapes.Text)
Class (MyTextShape_Drawing.Shapes.Text)

Figure 4. Atomic Blocks Mined from *Concept_1*

5 Experimentation

This section presents the case studies in which we apply our approach. We present the evaluation metrics in this section. We also describe our prototype implementation and, at last, we present the feature mining results and threats to validity of our approach.

Case studies: To validate our approach, we ran experiments on three Java open-source softwares: Mobile media[3] (small), Health watcher[4] (medium) and ArgoUML-SPL[5] (large). We used 5 variants for Mobile media, 10 variants for Health watcher and 10 variants for ArgoUML-SPL. Mobile media, Health watcher and ArgoUML variants are well documented. Their variants feature model is available for comparison to our results and validation of our proposal. Mobile media is a Java-based open source application which manipulates photo, music, and video on mobile devices, such as mobile phones. Health Watcher is a Java-based open source web application that manages health related records and complaints. ArgoUML is a Java-based open source tool widely used for designing systems in UML. Mobile media, Health watcher and ArgoUML variants are presented in Table 5 characterized by metrics LOC (Lines of Code), NOP (Number of Packages), NOCl (Number of Classes) and NOCo (Number of Couplings).

Table 5. Mobile media, Health watcher and ArgoUML software product variants

Product #	Mobile Media Product Description	LOC	NOP	NOCl	NOCo
P1	Mobile photo - Base	760	10	16	30
P2	Exception handling included	1,050	15	25	38
P3	New feature added to send/receive photo	1,823	17	38	192
P4	New feature added to manage music	2,214	17	47	310
P5	New feature added to manage videos	2,645	17	51	420
Product #	Health Watcher Product Description	LOC	NOP	NOCl	NOCo
P1	Base - no extensions applied	5,288	22	88	776
P2	Command pattern applied	5,646	23	92	932
P3	State pattern applied	6,112	24	104	1076
P4	Observer pattern applied	6,222	26	106	1082
P5	Adapter pattern applied	6,379	26	108	1112
P6	Abstract Factory pattern applied	6,417	27	112	1122
P7	Adapter pattern applied	6,441	27	116	1202
P8	Abstract Factory pattern applied	6,468	28	120	1214
P9	New functionality added	7,709	28	132	1992
P10	Exception handling applied	7,591	29	135	1956
Product #	ArgoUML Product Description	LOC	NOP	NOCl	NOCo
P1	All Optional Features disabled	82,924	55	1,243	6626
P2	All Optional Features enabled	120,348	81	1,666	17690
P3	Only Logging disabled	118,189	81	1,666	17388
P4	Only Cognitive disabled	104,029	73	1,451	8574
P5	Only Sequence diagram disabled	114,969	77	1,608	17110
P6	Only Use case diagram disabled	117,636	78	1,625	17368
P7	Only Deployment diagram disabled	117,201	79	1,633	15338
P8	Only Collaboration diagram disabled	118,769	79	1,647	17554
P9	Only State diagram disabled	116,431	81	1,631	17098
P10	Only Activity diagram disabled	118,066	79	1,648	17584

Evaluation Measures: In order to evaluate our approach and based on our knowledge about software variants and their features (*i.e.*, OBEs for each feature) we have used three measures: *precision*, *recall* and *F-Measure* [3]. Recall is the percentage of correctly retrieved OBEs to the total number of relevant OBEs, while precision is the percentage of correctly retrieved OBEs to the total number of retrieved OBEs. F-Measure defines a tradeoff between precision and recall so that it gives a high value only in case where both recall and precision are high. All measures have values in [0, 1]. If recall equals 1, all relevant OBEs are retrieved. However, some retrieved OBEs might not be relevant. If precision equals 1, all retrieved OBEs are relevant. However, relevant OBEs might not be retrieved. If F-Measure equals 1, all relevant OBEs are retrieved.

[3] homepages.dcc.ufmg.br/~figueiredo/spl/icse08/
[4] http://ptolemy.cs.iastate.edu/design-study/
[5] http://argouml-spl.tigris.org/

However, some retrieved OBEs might not be relevant.

Implementation: To analyze product variants source code and extract OBEs we used the Eclipse Java Development Tools (JDT) which is based on Eclipse AST (Abstract Syntax Tree). JDOM library is used to present OBEs for each software variant in XML format. For applying FCA we used the Eclipse eRCA platform[6]; eRCA is a framework that eases the use of Formal and Relational Concept Analysis. For the purpose of our approach, we developed our LSI tool[7]. The FeatureIDE[8] plugin is integrated to represent the feature models of software variants[9].

Result: Table 6 summarizes the obtained results. For readability's sake, we manually associated feature names to atomic blocks, based on the study of the content of each block and on our knowledge on software. Of course, this does not impact the quality of our results. Results show that precision appears to be high for all optional features. This means that all mined OBEs grouped as features are relevant. This result is due to search space reduction. In most cases, each BV corresponds to one and only feature. For mandatory features, precision is also quite high thanks to our clustering technique that identifies ABVs based on lexical and structural similarity. However, precision is smaller than the one obtained for optional features. This deterioration can be explained by the fact that we do not perform search space reduction for the CB. Considering the recall metric, its average value is 100% for Mobile Media, Health Watcher and ArgoUML. This means that all OBEs that compose features are mined. Considering the F-Measure metric, our approach has values that range from 70% to 100%. This means that most OBEs that compose features are mined and provides initial evidence with regard to the efficiency of our approach. All common and optional features are mined for all case studies except *logging* feature in ArgoUML-SPL and *create album/photo* and *delete album/photo* features in Mobile Media software variants; the reason behind this limitation is that the logging, create album/photo and delete album/photo features are implemented by method body and our approach only considers the software variants whose variability is represented mainly in the package or class level. The results of this evaluation showed that most of the features were identified and proves the scalability of our feature mining approach. In our approach we use a variable *K* with different ratios to determine the number of topics in each block. The column (K) in Table 6 shows the *K* value for each feature. Figure 5 shows the mined feature model for ArgoUML-SPL. The mined feature model consists of optional and mandatory features with only one level of hierarchy and without cross-tree constraints and groups of features constraints.

Table 6. Features mined from Mobile media, Health watcher and ArgoUML softwares

Case Study	Feature					Evaluation Metrics		
	Mandatory	Optional	NOCl	NOCo	K	Precision	Recall	F-Measure
ArgoUML-SPL Features								
Class	x		55	170	0.03	72%	100%	82%
Diagram		x	18	16	0.06	100%	100%	100%
Sequence		x	57	140	0.02	100%	100%	100%
Deployment		x	20	8	0.05	100%	100%	100%
Collaboration		x	19	2	0.06	100%	100%	100%
Cognitive		x	207	6556	0.01	100%	99%	99%
Use case		x	39	14	0.03	100%	100%	100%
State		x	35	48	0.03	100%	100%	100%
Activity		x	18	8	0.06	100%	100%	100%
Health Watcher Features								
Special Complaint	x		10	18	0.20	83%	100%	90%
Food Complaint	x		10	20	0.20	83%	100%	90%
Animal Complaint	x		10	20	0.20	83%	100%	90%
Update Health Unit	x		5	10	0.20	60%	100%	75%
Distribution	x		6	8	0.20	100%	100%	100%
Update employee	x		6	16	0.20	100%	100%	100%
Infrastructure Management		x	22	506	0.09	100%	95%	97%
Complaint Management		x	12	40	0.09	100%	100%	100%
Java RMI		x	3	2	0.34	100%	100%	100%
Database		x	4	4	0.25	100%	100%	100%
Support services for users		x	4	0	0.25	100%	100%	100%
Computer infrastructure		x	4	6	0.25	100%	100%	100%
Update Complaint		x	12	98	0.09	100%	100%	100%
Java Servlets		x	19	342	0.06	100%	100%	100%
Exception Handling		x	4	3	0.40	100%	80%	88%
Mobile Media Features								
Album Managment	x		6	6	0.25	85%	100%	92%
Photo Management	x		5	4	0.25	71%	100%	83%
Exception		x	8	6	0.13	100%	100%	100%
Photo List Screen		x	5	6	0.20	100%	100%	100%
Video Management		x	6	8	0.17	100%	100%	100%
SMS Transfer		x	12	50	0.09	100%	100%	100%
Music Management		x	17	92	0.06	100%	100%	100%

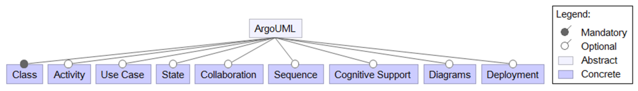

Figure 5. ArgoUML feature model

In our previous work [1], we applied experimentation on Mobile Media and ArgoUML software variants based on the lexical similarity only. Comparing with our new results when we combine both lexical and structural similarity we find that all evaluation measures now have values quite higher than lexical similarity alone (*cf.* Table 7). This is especially true for recall measure; with our new proposal, we get all OBEs that compose the feature and recall measure value is 100% for all features (except cognitive support where its value is 99%).

Threats to validity: One threat to the validity of our approach is that we only investigate product variants in which the variability is represented in the packages or classes without considering software variants where the variability is represented in the method level or in the method body level.

[6]`http://code.google.com/p/erca/`
[7]`Available at http://code.google.com/p/lirmmlsi`
[8]`http://www.fosd.de/featureide/`
[9]`Available at https://code.google.com/p/refm/`

Table 7. Comparing the two approaches

	Lexical similarity		Lexical and structural similarity	
Measure / Average	Mobile Media	ArgoUML	Mobile Media	ArgoUML
Precision	87%	97%	96%	97%
Recall	66%	67%	100%	100%
F-Measure	75%	79%	98%	98%

In this paper, we use different software variants. Fortunately, in the three softwares, there is a common vocabulary which helped us in having good results from the lexical similarity. As another limitation of our approach, developers might not use the same vocabularies to name OBEs across software variants. This means that lexical similarity may be not reliable (or should be improved with other techniques) in all cases to identify common and variable features.

6 Related work

In our previous work [2] we present an approach for feature mining in a collection of software product variants based on FCA by distinguishing between the *common block* and *blocks of variation*. We extended our previous work [2] by splitting blocks of source code elements based on the lexical similarity [1]. In this paper we rely on the lexical and structural similarity to mine features. The results showed that the combined approach is better than the lexical similarity alone. Rubin *et al.* [11] present an approach to locate optional features from two product variants' source code. They do not consider common features and limit their proposal to two variants. The approach proposed by Ziadi *et al.* [14] is the closest one. They identify all common features as a single mandatory feature. However, they do not distinguish between optional features that appear together in a set of variants. McMillan *et al.* [10] propose an approach to identify traceability links between source code and documentation by combining both textual and structural analysis. Yang *et al.* [13] analyzed open source applications for multiple existing domain applications with similar functionalities. They propose an approach to recover domain feature models using FCA, concept pruning/merging, structure reconstruction and variability analysis. Duszynski *et al.* [6] describe a framework for the analysis and visualization of similarities (i.e., commonalities) and variations (i.e., variabilities) across related software variants based on clone detection.

7 Conclusion and perspectives

In this paper, we proposed an approach for mining features from object-oriented source code of software product variants based on lexical and structural similarity. We have implemented our approach and evaluated its produced results on three case studies. Results showed that most of the features were identified. In this paper, we manually associated feature names to atomic blocks, based on the study of the content of each block. As a future work we plan to automatically propose feature names for the atomic blocks. We also plan to use the mined common and variable features and the lattices to automate the building of the studied software family's feature model with its constraints.

References

[1] R. AL-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman. Mining Features from the Object-Oriented Source Code of a Collection of Software Variants Using Formal Concept Analysis and Latent Semantic Indexing. In *25th SEKE Conference*, 2013.

[2] R. AL-Msie'deen, A. D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman. Feature location in a collection of software product variants using Formal Concept Analysis. In *13th ICSR*, pages 302–307. Springer, 2013.

[3] T. F. Bissyand, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère. Empirical Evaluation of Bug Linking. In *CSMR 17th Conference*, pages 89–98. IEEE, 2013.

[4] S. Budhkar and A. Gopal. Component-based architecture recovery from object oriented systems using existing dependencies among classes. *International Journal of Computational Intelligence Techniques*, 3(1):56–59, 2012.

[5] P. C. Clements and L. M. Northrop. *Software product lines: practices and patterns*. Addison-Wesley, 2001.

[6] S. Duszynski, J. Knodel, and M. Becker. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *18th WCRE*, pages 303–307. IEEE, 2011.

[7] B. Ganter and R. Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer, 1999.

[8] A.-E. E. Hamdouni, A. D. Seriai, and M. Huchard. Component-based Architecture Recovery from Object Oriented Systems via Relational Concept Analysis. In *CLA '10 Conference*, pages 259–270, 2010.

[9] A. Marcus and J. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE Conference*, pages 125–135. IEEE, 2003.

[10] C. McMillan, D. Poshyvanyk, and M. Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *TEFSE '09 Workshop*, pages 41–48. IEEE, 2009.

[11] J. Rubin and M. Chechik. Locating distinguishing features using diff sets. In *27th ASE Conference*, ASE 2012, pages 242–245. ACM, 2012.

[12] S. Wang, D. Lo, Z. Xing, and L. Jiang. Concern Localization using Information Retrieval: An Empirical Study on Linux Kernel. In *WCRE '11 Conference*, pages 92–96. IEEE, 2011.

[13] Y. Yang, X. Peng, and W. Zhao. Domain Feature Model Recovery from Multiple Applications Using Data Access Semantics and Formal Concept Analysis. In *WCRE '09 Conference*, pages 215–224. IEEE, 2009.

[14] T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane. Feature identification from the source code of product variants. In *CSMR'2012*, pages 417–422, 2012.