# An approach to recover feature models from object-oriented source code

Ra'Fat Ahmad Al-Msie'Deen, Abdelhak-Djamel Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, Hamzeh Eyal-Salman

# An approach to recover feature models from object-oriented source code

**R. AL-Msie'deen[*], A. Djamel Seriai[*], M. Huchard[*], C. Urtado[**], S. Vauttier[**], and H. S. Eyal Salman[*]**

[*] *LIRMM (CNRS and Univ. Montpellier 2), Montpellier, France,*
[**] *LGI2P / Ecole des Mines d'Alès, Nîmes, France*

ABSTRACT. *Software Product Line (SPL) is a development paradigm that targets the creation of software system variants that belong to the same domain. Usually software system variants, developed with clone-and-own approach, form a starting point for building SPL. To migrate software systems which are deemed similar to a product line, it is necessary to detect the common features and variations between a set of software system variants. Reverse engineering the feature model (FM) of an existing system is a challenging activity. FM describes the common and variable characteristics of a product line. In recent years, a lot of work has addressed the extraction of FM from different artefacts. Little work addressed extraction of FM from source code. This paper proposes a general approach to extract initial FM from the object-oriented (OO) source code of a set of software system variants in order to support the migration process from conventional software development to software product line engineering (SPLE). We present an approach to extract features of FM from the analysis of object-oriented source code for a set of software product variants. This approach is based firstly on the definition of the mapping model between object-oriented elements (OOE) and those of FM. Secondly; it uses an identification process exploiting on the one hand Formal Concept Analysis (FCA) as a method for clustering OOE corresponding to the implementation of features and on the other hand Latent Semantic Indexing (LSI) to define a similarity measure on which is based this clustering.*

KEYWORDS: *Software product line engineering; feature identification; feature model; source code variation; OO source code reverse engineering; software system variants; Formal Concept Analysis; Latent Semantic Indexing.*

1re soumission à *JLP 2012*, le April 5, 2013.

## 1. Introduction

Several definitions of SPLs can be found in the literature; according to Clements and Northrop [CLE 01] a SPL is "a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way". A SPL is usually characterized by two sets of features: the features that are shared by all products in the family, called the *SPL's commonalities*, and, the features that are shared by some, but not all, products in the family, called the *SPL's variability*. These two sets define the mandatory and optional parts of the SPL.

In order to provide a more subtle description of the possible combinations of optional features (e.g., some optional feature might exclude another and require a third one), SPLs are usually described with a defacto standard formalism called *feature model* [ACH 12]. A feature model characterizes the whole software family. It defines all the valid feature sets or configurations. Each valid configuration represents a specific product (either it be an existing product or a valid product-to-be). Feature modelling is a method for describing commonalities and variabilities in software product line. Feature model was first introduced in the Feature-Oriented Domain Analysis (FODA) method by [KAN 90].

*Software product line engineering* (SPLE) is the process to both model the software family (also called domain engineering) and develop a software that sticks to the software family definition (also called application engineering) [CLE 01]. When investigating the actually practiced development methods, it appears that the need for a disciplined SPLE process appears after the development of several product variants [DUS 11]. These variants are developed using *ad hoc* techniques such as *copy*, *paste*, *modify* without explicit plan or strategy for reuse. Once released, if the products meet their market, similar products are to be developed [JOH 09] and it becomes too complex to develop and maintain them and too costly without switching to SPLE.

Manual analysis of the existing products and manual feature model reverse engineering for the existing software family is time-consuming and error-prone, and requires substantial effort. Automating the reverse engineering of a feature model from source code would be of great help. Expected benefits are to improve product maintenance, ease system migration, and discover new valid configurations that may lead to the production of new software products [CHI 90]. We use FCA and IR to get these objectives.

We use Formal Concept Analysis to extract commonalities and variabilities for a set of product variants. FCA is a mathematical method that provides a way to identify "meaningful groupings of objects that have common attributes"[LOE 07]. Information retrieval (IR) has proven useful in many disciplines such as software maintenance and evolution, image extraction, speech recognition and horizontal search engines like Google. Furthermore feature location is one of the most common applications of IR in software engineering [DAV 11]. IR methods sort the documents against queries by extracting information about the occurrences of terms within them. The extracted

information is used to find similarity between queries and documents. LSI assumed that there are some implicit relationships among the words of documents that always appear together even if they do not share any terms; that is to say, there are some latent semantic structures in free text [DAV 11].

In recent years, a lot of work on reverse engineering has addressed the extraction of feature models from different artefacts[ZIA 12, RYS 11]. Few works have addressed the problem of identification of FM from the source code of product variants [ZIA 12] (see Section 7).

The main goal of our work is to recover the features from OO source code based on FCA to extract commonalities and variations from product variants and integrate Latent Semantic Indexing (LSI) with FCA to recover the features.

The remainder of this paper is organized as follows. Section 2 shows an overview of the approach and the variation-feature mapping model, and presents our proposed feature model extraction process from object-oriented building elements (OBE). Section 3 discusses source code variability identification from OO source code for a set of product variants. Section 4 explains the extraction process for commonalities and variations using FCA. Section 5 explains the extraction process for the atomic block of variations (feature) from block of variations using LSI and FCA. Section 6 discusses the implementation. Section 7 discusses related work that addressed reverse engineering feature models from different artefacts. Finally, we conclude and draw perspectives for this work in Section 8.

## 2. Approach Overview

In this section, we will explain the main concepts of our approach and how these concepts can be used to apply our approach.

### 2.1. *Features versus object-oriented elements: the mapping model*

The general objective of our approach is to extract FM which model common and variable features of software product variants. We present in this paper the part concerning features identifications. We rely on the following definition of the feature: "a feature is a prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems" [KAN 90]. We consider that a feature represents an aspect valuable to the customer. It is represented by a single term. We adhere to the classification given by [KAN 90] which distinguishes three categories of features: Functional, operational, and presentation features. To identify features we rely on the mapping model between these features and object-oriented building elements (OBE) (see Fig. 1).

As there are several ways to implement the features [BEU 04], we assume that the features are implemented at the programming language level. Thus, the elements of the source code can reflect these features. For object-oriented source code, the manda-

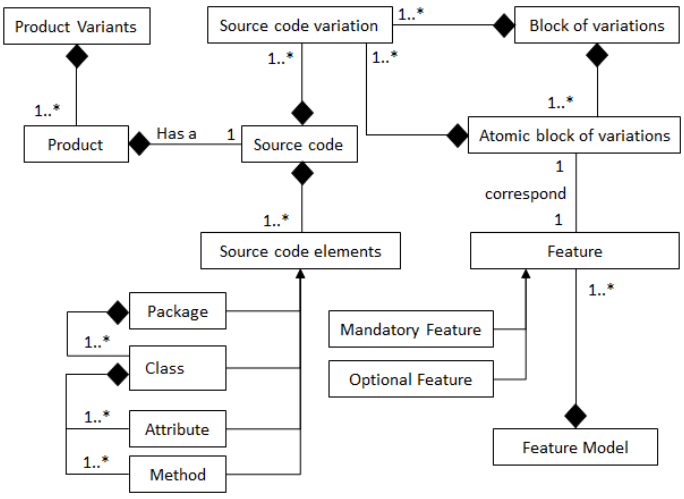


**Figure 1.** *Variation-feature mapping model*

tory features are realized by *object-oriented building elements* (OBE) (i.e. packages, classes, etc.) that are shared by all product variants. However, the optional features are realized by variable elements that appear only in some variants. We consider that a feature corresponds to one and only one set (group) of OBE. This means that a feature always has the same implementation in all products where it is present.

An optional feature is implemented by *variable object-oriented building elements* (VOBE) in all products where it is present. So we define a *block of variations* (BV) as a set of (VOBE) which are always associated (i.e., which are always identified together in all the products in which they appear).

It is clear that a VOBE cannot occur in a product unless accompanied by all the VOBE that are part of the implementation of the corresponding feature. This is also the case for VOBE that belong to interdependent features (linked via "*and*" or "*require*" constraints). Therefore, a VOBE implements an optional feature that necessarily belongs to one and only one BV. Following our approach a BV can gather VOBE that represent one or more features linked by "and" or "require" constraints. The BV are found thanks to FCA.

The subsets of VOBE that belong to a BV and represent one and only one feature are called *atomic blocks of variations* (ABV). A BV is composed of set of ABVs. To determine its sub-parts, we rely on the clustering of the closest VOBEs considering the similarity measures that are related to LSI method.

The mandatory features are implemented by *common object-oriented building elements* (COBE) in all product variants. In the same way as for the identification of ABV,

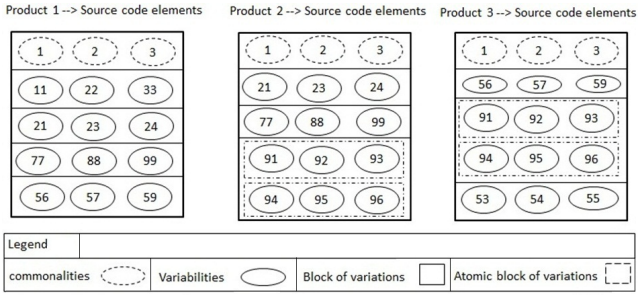


**Figure 2.** *Illustrative example.*

we rely on the clustering of the closest OBE in the *common block* (CB) to determine the parts of this partition. Each part will be considered as a mandatory feature.

Fig. 2 shows an example of a set of product variants (3 products). This is an abstract example to show the main concepts that exist in the variation-feature mapping model.

### 2.2. *Feature model extraction process*

The approach that we propose is illustrated in Fig. 3. Feature model extraction process consists of the following steps:

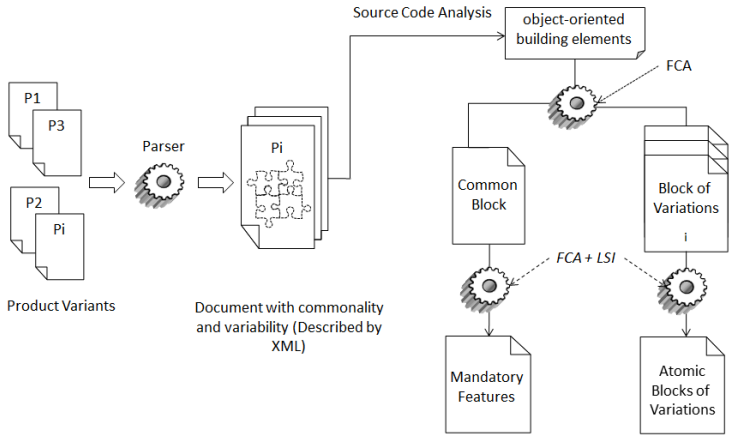


**Figure 3.** *Feature model extraction process*

*- OO Source code is analyzed to extract object-oriented building elements (packages, classes, methods, attributes) for all product variants.*
*- Commonalities and variations are extracted for all product variants using FCA.*

*Blocks of variations are given by using FCA.*
*- Blocks of variations are divided into atomic blocks of variations. Each atomic block of variations corresponds to one and only one feature. In this step, we use LSI and FCA to identify features based on the textual similarity.*

## 3. Identification of source code variability: OO source code analysis

Any OO source code can hold four levels of variations: package variation, class variation, attribute variation, method variation (see Fig. 4). Package variation shows variation on two levels: package set variation (set of packages that appear in some products but not all products), and package content variation (means all product variants have the same packages but with different contents). Variation at the class level can appear in the class signature, attribute set and method set. Class signature variation means that two or more classes have the same name in many packages but declare different relations, or different access levels. Method and attribute set variation captures the differences between product variants in term of provided functionalities. Attribute variation can be found in attribute declarations such as: access level, data type, etc. Method variation can appear in the method signature (access level, returned data type, parameter list, parameter list order and method exception) and in the body of the method (local variable, method invocation, access).

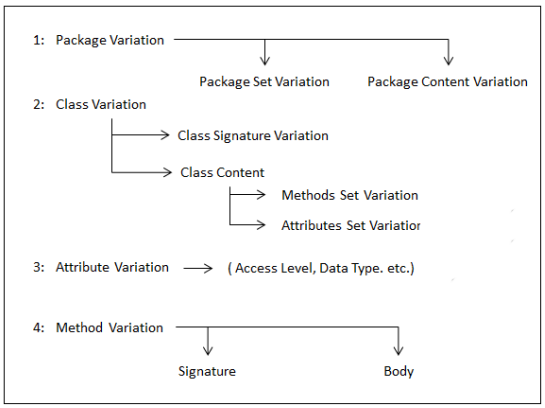


**Figure 4.** *Object-oriented source code variations.*

## 4. Commonality and variation identification using FCA

We use FCA to extract commonalities and variabilities for a set of product variants. In the concept lattice the upper concept represents all source code elements that are shared by all products (common block), while other concepts reflect variability among

this family (variabilities) and contain all source code variations shared by some products but not all products (block of variations). In the formal context, products constitute the rows while source code elements (packages, classes, methods, attributes) constitute the columns (see Table 1).

**Table 1.** *Part of the formal context describing text editing systems by source code elements*

| | Package (Name:um.lirmm.texteditor.changedisplaysettings) | Class (Name:Create, owner:um.lirmm.texteditor.filemanagement.file) | Class (Name:PDF, owner:um.lirmm.texteditor.filemanagement.print) | Class (Name:ViewHelp, owner:um.lirmm.texteditor.viewhelp) | Class (Name:Save, owner:um.lirmm.texteditor.filemanagement.savefile) | Class (Name:SelectAll, owner:um.lirmm.texteditor.filemanagement.edit) | Method (Name:SelectAll, owner:SelectAll) | Class (Name:Resize, owner:um.lirmm.texteditor.changedisplaysettings.resize) | Class (Name:Clear, owner:um.lirmm.texteditor.filemanagement.clear) | Class (Name:ReadOnly, owner:um.lirmm.texteditor.filemanagement.readonly) | Class (Name:UnSplit, owner:um.lirmm.texteditor.changedisplaysettings.unsplitall) | Class (Name:Horizontal, owner:um.lirmm.texteditor.changedisplaysettings.split()) | Class (Name:Vertical, owner:um.lirmm.texteditor.changedisplaysettings.split) | Method (Name:setVertical, owner:Vertical) | Local Variable (Name:Vertical, owner:setVertical) | Method Invocation (Name:print, Accessed-in:[Vertical], owner:setVertical) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TextEditingSystem1** | × | × | × | × | × | × | × | × | | × | × | × | × | × | × | × |
| **TextEditingSystem2** | × | × | × | × | × | × | × | × | × | | × | × | × | × | × | × |
| **TextEditingSystem3** | × | × | × | × | × | × | × | | × | | × | × | × | × | × | × |
| **TextEditingSystem4** | × | × | × | × | × | | | | | | | | | | | |
| **TextEditingSystem5** | × | × | × | × | × | | | × | | × | | | | | | |
| **TextEditingSystem6** | × | × | × | × | × | | | | | × | × | × | × | × | × | × |
| **TextEditingSystem7** | × | × | × | × | × | × | × | × | × | | | | | | | |
| **TextEditingSystem8** | × | × | × | × | × | | | | | × | × | × | × | × | × | × |

The concept lattice is presented in Fig. 5. The common block contains all the source code elements that implement mandatory features. The source code elements that are shared by more than one product are called a *block of variations*. A Block of variations contains source code elements that appeared every times together to implement a set of features for some product.

## 5. Atomic block of variations (feature) identification using LSI and FCA

To identify the atomic block of variations that represent a single feature from a block of variations, we consider LSI and FCA to recover all atomic block of variations. In our case, each line in the block of variations represents a single document and at the same time represents a query.

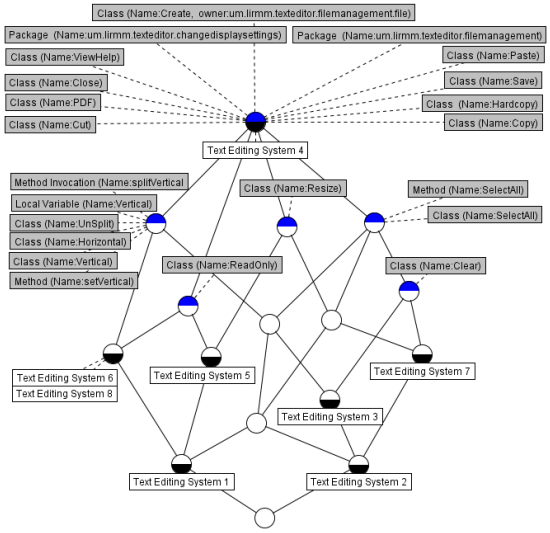


**Figure 5.** *The concept lattice for the formal context of Table 1*

Most of the existing tools about FCA are referenced from the web page of Uta Priss [1]. For this paper, we used the eclipse eRCA platform [2] and Concept Explorer [3].

All information in the block of variations must be manipulated and normalized to become suitable as input of LSI. This preprocessing step includes: all capital letters must be transformed into lower case letters, removing stop-words (such as articles, punctuation marks, numbers, etc.), all lines must be split into terms and performing word stemming.

Similarity between lines is described by a similarity matrix. In the similarity matrix columns represent lines vectors and rows represent lines vectors also. LSI uses each line in the block of variations as a query to retrieve all lines that have similarity with it, according to cosine similarity. In our work, we consider the most widely used threshold for cosine similarity that is equals to 0.70 [DAV 11]. We use the similarity matrix (see Table 2) (LSI result) as input for the FCA to group the similar elements together based on the lexical similarity; after that, we ignore any document that has similarity with itself only (see Table 4). Table 3 shows the formal context of the similarity matrix (threshold for cosine similarity equals to 0.70). So we take the interchanged context as input for FCA; FCA identifies the meaningful groupings of objects that have common attributes. In our case, the concept lattice (see Fig. 6) shows two atomic block

1. http://www.upriss.org.uk/fca/fca.html
2. http://code.google.com/p/erca/
3. http://conexp.sourceforge.net/

of variations. Each atomic block represents one and only one feature. Note that each document ($Doc\_i$) represents a line from a block of variations.

**Table 2.** *The similarity matrix (SimMat).*

| | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Doc 6 | Doc 7 | Doc 8 |
|---|---|---|---|---|---|---|---|---|
| **Doc 1** | 1 | 0.70 | 0 | 0 | 0 | 0.70 | 0 | 0 |
| **Doc 2** | 0.70 | 1 | 0 | 0 | 0 | 0.70 | 0 | 0 |
| **Doc 3** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **Doc 4** | 0 | 0 | 0 | 1 | 0.70 | 0 | 0 | 0 |
| **Doc 5** | 0 | 0 | 0 | 0.70 | 1 | 0 | 0 | 0 |
| **Doc 6** | 0.70 | 0.70 | 0 | 0 | 0 | 1 | 0 | 0 |
| **Doc 7** | 0.70 | 0.70 | 0 | 0 | 0 | 0.70 | 1 | 0 |
| **Doc 8** | 0.70 | 0.70 | 0 | 0 | 0 | 0.70 | 0 | 1 |

**Table 3.** *The context (SimMat) for $\theta$= 0.70.*

| | Doc 1 | Doc 2 | Doc 3 | Doc 4 | Doc 5 | Doc 6 | Doc 7 | Doc 8 |
|---|---|---|---|---|---|---|---|---|
| **Doc 1** | x | x | | | | x | | |
| **Doc 2** | x | x | | | | x | | |
| **Doc 3** | | | x | | | | | |
| **Doc 4** | | | | x | x | | | |
| **Doc 5** | | | | x | x | | | |
| **Doc 6** | x | x | | | | x | | |
| **Doc 7** | x | x | | | | x | x | |
| **Doc 8** | x | x | | | | x | | x |

**Table 4.** *The interchanged (SimMat) context.*

| | Doc 1 | Doc 2 | Doc 6 | Doc 4 | Doc 5 |
|---|---|---|---|---|---|
| **Doc 1** | x | x | x | | |
| **Doc 2** | x | x | x | | |
| **Doc 4** | | | | x | x |
| **Doc 5** | | | | x | x |
| **Doc 6** | x | x | x | | |
| **Doc 7** | x | x | x | | |
| **Doc 8** | x | x | x | | |

## 6. Approach implementation

To validate our approach, we used a text editing software product line [4] as a case study. A text editor is a computer program that lets a user enter, change, store, and usually print text, and provide the basic operations that satisfy the end user. This family has eight product variants. Each product implements a simple text editing application. Features are collected in what it is called a FM to specify the variations between these products. The feature model of the text editing system is shown in Fig. 7; the features with white circles on top are optional features while all features with black circles on top are mandatory features. Fig. 5 shows small part of the concept lattice for these products. We extract the common block that contains all common (mandatory) features, and a set of blocks of variations that contain source code elements for optional

4. `http://www.lirmm.fr/TextEditingSystemSPL`

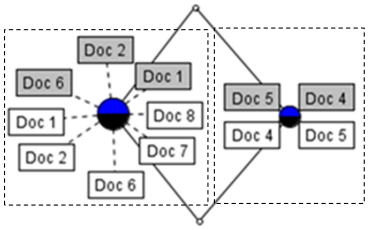


**Figure 6.** *The concept lattice for the formal context of Table 4.*

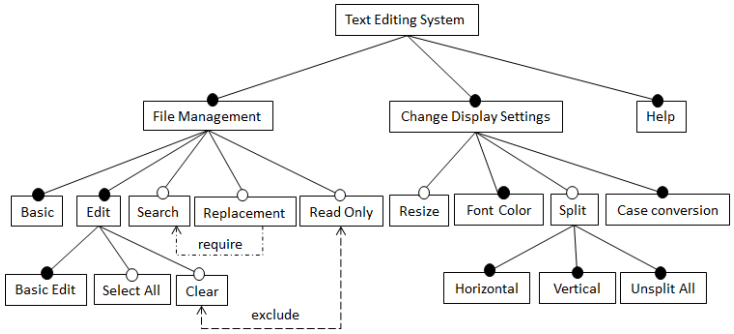


**Figure 7.** *Text editing system FM.*

features. Each block of variations has at least one atomic block of variations that represents a single optional feature. In the text editing system, the concept lattice shows that unsplit, split horizontal, and split vertical features all times appear together in the same block of variations. After applying LSI with FCA on this block we recover three atomic blocks of variations (each one represent a single feature) based on the textual similarity. Concept lattice (see Fig. 8) shows the recovered features from this block.

## 7. Related work

Ziadi et al. [ZIA 12] propose an automatic approach for feature identification from source code for a set of product variants. Their approach only investigates products in which the variability is represented in the name of classes, methods and attributes, without considering a product lines in which the variability is mainly represented in the body of methods. The recovered feature model contains only one mandatory feature, and optional features. The extracted feature model has only one level of hierarchy, without distinction between the mandatory features, without any feature group and group constraints, and without cross tree constraints. We use FCA to extract commonalities and variations from product variants and distinguish between the mandatory features by using LSI and FCA based on the lexical similarity, and extracts all optional features and constraints such as: "and" and "require".

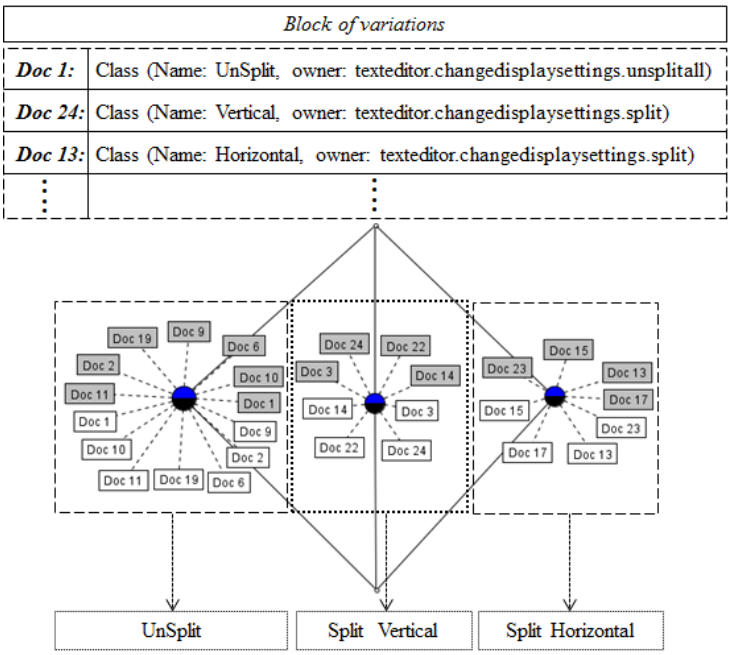


**Figure 8.** *Concept lattice shows three atomic blocks of variations extracted from one block of variations.*

Ryssel et al. [RYS 11] propose an approach to extract feature diagrams using FCA from an incidence matrix that contains matching relation as input. It shows the parts of a set of function- block oriented models that describe different controllers of a DC motor. They introduce an automatic approach to recognize variants in a set of models and identify the variation points and their dependencies within variants. In our approach the incidence matrix contains source code elements for a set of product variants, and we use FCA to extract commonalities and variations for these product variants.

Our approach focuses on recovering an initial feature model from a set of product variants to support the migration process from conventional software development to software product line engineering.

## 8. Conclusion

In this paper, we proposed an approach based on FCA and LSI to extract a feature model from the object-oriented source code of software system variants. FCA can be used to extract variations blocks. Then LSI is used with FCA to recover atomic blocks of variations that represent a single feature, using the textual similarity.

As future work, we will apply a clustering algorithm on the commonality and variability blocks to determine more precisely each feature implementation based on both textual and semantic similarity. For the semantic similarity, we rely on all available information and links that exists between variable object-oriented building elements such as: inheritance (which class inherits from which class), invocations (which method invokes which method). Also we will try to organize the extracted features as a feature model including all cross-tree constraints, using information contained in the concept lattice.

## 9. References

[ACH 12] ACHER M., HEYMANS P., MICHEL R., "Next-generation model-based variability management: languages and tools", *Proceedings of the 16th International Software Product Line Conference*, vol. 2 of *SPLC '12*, New York, NY, USA, 2012, ACM, p. 276–277.

[BEU 04] BEUCHE D., PAPAJEWSKI H., SCHRÖDER-PREIKSCHAT W., "Variability management with feature models", *Sci. Comput. Program.*, vol. 53, num. 3, 2004, p. 333–352, Elsevier North-Holland, Inc.

[CHI 90] CHIKOFSKY E. J., CROSS II J. H., "Reverse Engineering and Design Recovery: A Taxonomy", *IEEE Software*, vol. 7, num. 1, 1990, p. 13–17, IEEE.

[CLE 01] CLEMENTS P. C., NORTHROP L. M., *Software product lines: practices and patterns*, Addison-Wesley, 2001.

[DAV 11] DAVID B., LAWRIE D., "Information Retrieval Applications in Software Maintenance and Evolution", In Encyclopedia of Software Engineering, 2011, p. 454-463.

[DUS 11] DUSZYNSKI S., KNODEL J., BECKER M., "Analyzing the Source Code of Multiple Software Variants for Reuse Potential", p. 303-307, IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[JOH 09] JOHN I., EISENBARTH M., "A decade of scoping: a survey", *Proceedings of the 13th International Software Product Line Conference*, Pittsburgh, PA, USA, 2009, Carnegie Mellon University, p. 31–40.

[KAN 90] KANG K. C., COHEN S. G., HESS J. A., NOVAK W. E., PETERSON A. S., "Feature-Oriented Domain Analysis (FODA) Feasibility Study", November 1990.

[LOE 07] LOESCH F., PLOEDEREDER E., "Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations", KRIKHAAR R. L. VERHOEF C. L. G. A. D., Ed., *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, Netherlands, March 2007, IEEE, p. 159–170.

[RYS 11] RYSSEL U., PLOENNIGS J., KABITZSCH K., "Extraction of feature models from formal contexts", *Proceedings of the 15th International Software Product Line Conference, Volume 2*, Munich, Germany, 2011, ACM, p. 4:1–4:8.

[ZIA 12] ZIADI T., FRIAS L., DA SILVA M. A. A., ZIANE M., "Feature Identification from the Source Code of Product Variants", MENS T. CLEVE A. F. R., Ed., *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, Los Alamitos, CA, USA, 2012, IEEE, p. 417–422.