Recovering Traceability links between Feature Models and Source Code of Product Variants

Hamzeh Eyal-Salman
LIRMM, University of Montpellier 2
161, rue Ada
34095 Montpellier cedex 5, France
(33) 04 67 41 85 85
Hamzeh.Eyalsalman@lirmm.fr

Abdelhak-Djamel Seriai
LIRMM, University of Montpellier 2
161, rue Ada
34095 Montpellier cedex 5, France
(33) 04 67 41 85 85
Abdelhak.Seriai@lirmm.fr

Ra'fat Al-msie'deen
LIRMM, University of Montpellier 2
161, rue Ada
34095 Montpellier cedex 5, France
(33) 04 67 41 85 85
Rafat.Al-msiedeen@lirmm.fr

ABSTRACT
Usually software product variants, developed by copy-paste-modify technique, are often a starting point for building Software Product Line. The distinguishing factor between traditional software engineering and software product line engineering is the variability. Traceability of variability in a software product line has been recognized as crucial factor for its success. This paper presents a method based on information retrieval namely, latent semantic indexing, to establish traceability links between object-oriented source code of product variants and its FM to support conversion from traditional software development into software product line development. Tracing and maintaining interrelationships between artifacts within a software system also are needed to automate products derivation process, facilitate program comprehension, make the process of maintaining the system less dependent on individual experts.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement–*Restructuring, reverse engineering, and reengineering*; H.3.3 **[Information Systems]:** Information Search and Retrieval- *Clustering , Information filtering.*

General Terms
Theory, Design, Documentation.

Keywords
Traceability links, feature models, source code, variability, software product line , latent semantic indexing.

1. INTRODUCTION
Product variants often evolve over the time from an initial product to a family of similar product variants that meet the need of a large group of consumers. The successful development of the initial product attracts new customers. For example, Wingsoft Financial Management System (WFMS) was developed for Fudan University and then evolved many times so that WFMS systems have been now used in over 100 universities in China [17].

Usually, developers use copy-paste-modify technique to build a new product variant from existing ones. Such ad hoc reuse technique causes a critical problem as the number of features and product variants grows because we must maintain each product variant separately from others and it also becomes difficult to find and trace features for reuse in new products.

As these problems accumulate, it becomes necessary to re-engineering product variants into a Software Product Line (SPL) for systematic reuse. SPL aims to decreasing development cost and time by developing a family of systems rather than one system at a time [16]. In the SPL, there are two models: feature model (FM) as representative of variability model and core asset model. FM has a pivot role because it represents a set of configurations where each valid configuration represents a specific product and it also is extensively used to automate the product derivation process [4]. Figure1 shows a simplified FM inspired by the mobile phone industry [3].

There are three issues that must be considered to reengineering product variants into SPL: FM, SPL artifacts (core assets model) and mapping between FM and SPL artifacts [12].

FM of product variants can be provided by system's developers and experts who accompanied and contributed product variants evolution. FM may also be reverse engineered from the documentations of products variants [1].

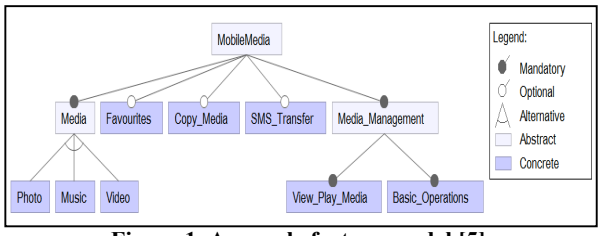


Figure 1. A sample feature model [5].

Regarding to SPL artifacts, the development team can utilize the available reusable elements such as: source code, design documents, test cases, etc. to building the required SPL core assets.

These parts (FM and core asset model) must be connected to exploit them during SPL life cycle [12]. The traceability links between source code of product variants and its FM are used to automate products derivation process in order to automatically configures all the assets for a product according to the features selection from the FM, exploit source code as an important artifact in SPL core asset, ensure consistency between extracted FM and source code, facilitate program comprehension process, make the process of maintaining the system less dependent on individual experts and recovery of various architectural elements.

This paper proposes a method based on information retrieval (IR) methods namely, latent semantic indexing , to establish and maintain traceability links between source code of product variants and textual descriptions of features as representative of FM. Feature names and descriptions can be extracted from documentation and code comments.

IR has proven useful in many disciplines such as the management of huge scientific and legal literature, image extraction and speech recognition. We believe that IR techniques can provide a way to establish the traceability links between source code and FM.

The remainder of the paper is organized as follows. Section 2 discusses background and related work. Section 3 shows the traceability link recovery process. Section details latent semantic indexing. Section 5 shows the experimental results. Finally, Section 6 presents conclusions and feature work.

2. BACKGROUND & RELATED WORK

Software traceability is the ability to describe and follow the life of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software lifecycle in both forward and backward directions (e.g., from requirements to the software architecture and the code components implementing them and vice-versa) [7].

Traceability relations refer to overlap, satisfiability, dependency, evolution, generalization/refinement, conflict and rationalization associations between various software artifacts [13]. In general, traceability relations can be classified as horizontal traceability or vertical traceability relations .The former type includes relations between different models, and the latter type includes relations between elements of the same model [10][15].

FM is a variability modeling technique widely used in SPLE to cover the variability in all SPL life cycle from requirements to test cases [3]. Variability defines what the allowed combinations of features (also called configurations) are. FM consists of feature diagram and cross tree constraint likes require and exclude constraints. Feature diagram is a tree like representation, the root of the tree refers to the complete system, tree nodes are features and tree edges represent dependency rules [14]. In the literature, there are many definitions of feature; in this paper we will consider the following definition [8]:

> "A distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained"

Many researchers attempted to establish traceability link via information retrieval (IR) approach [2][11]. IR-based approaches assume that all software artifacts are in textual format. Then, they compute textual similarity between two software artifacts using cosine similarity, e.g., a class and a requirement. The three IR methods which commonly used in traceability generation are probabilistic method, vector space method and latent semantic indexing.

Antoniol et al. [2] used probabilistic method (PM) and vector space model (VSM) to establish traceability links between source code and documentations. In each method, one type of particular artifacts treats as a query and another type of artifacts treats as a document. For example, source code treats as a query against requirements specification as a document and information is retrieved by literally matching terms in documents with a query.

Andrian et al. [11] used latent semantic indexing to establish traceability links between documentation and source code. In LSI, a query and a document can have cosine similarity even if they do not share any terms as long as their terms are semantically similar.

Ziadi et al. [20] proposed an approach to automate feature identification from the source code of similar products variants. This approach assumes that product variants use the same vocabulary to name packages, classes, attributes and methods; it treats the source code as a set of construction primitives and then applies an algorithm to identify features.

Ghanam et al. [6] presented an approach to link feature models to code artifacts using executable acceptance tests to ensure consistency between FM and code artifacts, and to trace the evolution of variability in the feature model.

3. OUR APPROACH TO RECOVER TRACEABILITY LINKS

This section describe our proposed approach to recovering traceability links between source code of product variants and its FM. Figure2 gives an overview about the traceability links process. The inputs of this process are FM, features description and object-oriented source code of product variants. The figure also shows three main phases:

1. Variability point extraction: In this phase, the variability points are reversed engineered from the source code where it can reflect four types of variations: package variation, class variation, method variation and attribute variation. To process source code, a parser is used to extract all information from the source code.
2. Mapping between variability points and FM: In this phase, we defined a corresponding model between variability points in the source code and variable features of FM (see figure 3). This model defines a feature as a block of variations. Each block acts a set of variations that appear together in the source code. The variability (variable features) can be implemented by four types of variations: packages variation, class variation, method variation and attribute variation. This paper will consider just class variation as a variability point in the source code. Class variation means a set of classes that make the difference among product variants in term of the provided functionality.
3. Applying a traceability method: In this phase, we will use latent semantic indexing to recovering the traceability links between source code and FM.

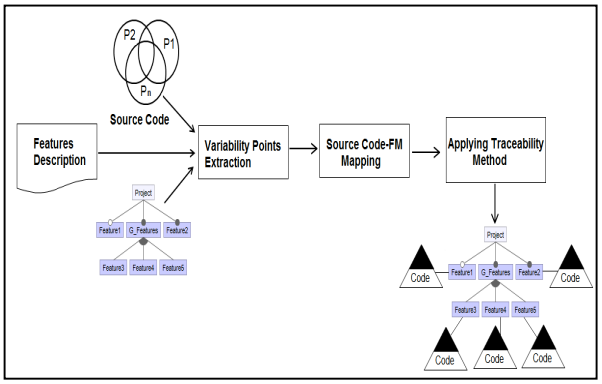
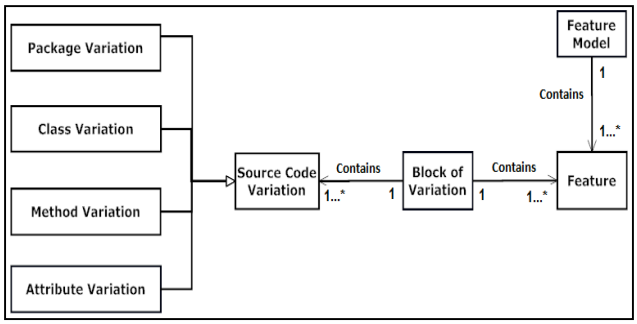


Figure 2.Traceabilityrecovering process overview.

Figure 3: Feature to source code mapping model.



4. LATENT SEMANTIC INDEXING (LSI)

LSI is a technique that projects queries and documents into a space with latent semantic dimensions. The basic assumption of LSI is that there exists some implicit relationships among the words of documents, that is to say, there are some latent semantic structures in free text. Semantic structure means an abstract semantic format which consists of semantic category and semantic relationship in natural languages [11].

LSI was developed to overcome problems that occur in the space vector model (VSM) namely, synonymy and polysemy, by replacing the original term–document matrix with an approximation. This is done using singular value decomposition (SVD), a technique originally used in signal processing to mitigate noise while preserving the original signal. Assuming that the original term–document matrix is noisy (the aforementioned synonymy and polysemy), the approximation is interpreted as a noise reduced – and thus better – model of the text corpus [9].

LSI will use feature descriptions as query to retrieve the classes related to the feature. In most object oriented languages class names are composed of concatenated terms like (EmailAddressFormatChecker) so that each term reflects partially the class functionality. We assume that programmers use meaningful names (i.e. names derived from the domain) to name classes.

Features descriptions and classes' names must be manipulated and normalized to become suitable as input of LSI. This preprocessing step include: all capital letters must be transformed into lower case letters, removing stop-words (such as articles, punctuation marks, numbers, etc.), all classes' names must be split into terms and performing word stemming.

LSI technique consists of the following steps [11]:

1. Constructing a term-document matrix whose [i, j]th element refers to the association between the ith term and jth document. This matrix is called VSM space. We will measure the weight of each term using Term Frequency (TF). TF refers to number of times term *i* occurs in the document *j*.
2. Decomposition VSM space LSI subspace by applying (SVD) to the term-document matrix. SVD is performed on the matrix to determine patterns in the relationships among the terms.
3. Computing the cosine similarity in LSI subspace by equation1.

$$Similarity(Q, D) = \frac{\sum_{i=1}^{m} d_i q_i}{\sqrt{\sum_{i=1}^{m} d_i^2 \sum_{i=1}^{m} q_i^2}} \quad \text{..... (Eq1)}$$

4. Filtering results according to a predetermined threshold, and then the traceability links between FM and source code are retrieved. In our work, the threshold is chosen in a heuristic way and its value is 0.5. This value means that classes that will be retrieved have a similarity with a feature description greater than or equal 0.5.

The effectiveness of IR methods is measured using IR metrics: recall, precision. For a given query, recall is the percentage of actual retrieved links over the total number of relevant links while precision is the percentage of correctly retrieved links to the total number of retrieved links (see equation 2 and 3) where (i) represents query set. [2].

Both measures have values between [0, 1]. If recall = 1, it means that all the correct links are recovered, however there could be recovered links that are not correct. If the

$$Recall = \frac{\sum_i \#(Relevant_i \wedge Retrieved_i)}{\sum_i \#Relevant_i}\% \quad \text{..... (Eq2)}$$

$$Precision = \frac{\sum_i \#(Relevant_i \wedge Retrieved_i)}{\sum_i \#Retrieved_i}\% \quad \text{..... (Eq3)}$$

precision=1, it means that all the recovered links are correct, however there could be correct links that were not recovered. Choosing a higher threshold for the link recovery will result in higher precision, while lowering the threshold will increase the recall. In general, the result of higher precision is a lower recall (and vice versa).

It is important to mention her that LSI will be applied two times. First, to recover traceability links between common feature and common classes while the second time to recover traceability links between variable features and variable classes. We can extract common classes by conducting a lexical matching among product variants' classes while common features can be extracted from FM. This task aims to recover traceability links with high precision by reducing number of classes.

5. EXPERIMENTAL RESULTS

In order to validate our approach as traceability recovering method between source code and FM, we will consider simple mobile media system to test this method. Figure 2, in the introduction section, represent a feature model for mobile media software. **Favourites**, **Copy_Media** and **SMS_Transfer** are optional features while **View_Play_media** and **Basic_Operations** are mandatory features. **Media** is alternative feature. Three configurations were chosen to realize three products including all mobile media features.

We assumed that each feature is described with certain words as shown in the table 1 below. For example, **SMS_Transfer** feature in the row 6 is described by (**send sms, receive sms)**.

Table1. Features description.

#	Feature name	Description
1	photo	capture photo, compression photo, scrambling photo, count photo
2	Music	Play, generate tones, organize music
3	Video	capture video, compression video, scrambling video
4	Favourites	set favourites, view favourites, save favourites
5	Copy_Media	copy media, store media
6	SMS_Transfer	send sms, receive sms
7	Basic_Operations	create media, delete media, edit media, label media, sorting media, move media, search media,save media
8	View_Play_Media	view media, play media

Also, we assumed that each feature is implemented with certain classes as shown in the table 2 below. For example,

SMS_Transfer feature in the row 6 is implemented by (**SendSMS** and **recieveSMS classes)**.

Table 2. Real implementations of features

#	Features	Classe Name	Class Id
1	Photo	CapturePhoto	1
		CompressionPhoto	2
		ScramblingPhoto	3
		CountPhoto	4
2	Music	GenerateTones	5
		OrganizeMusic	6
3	Video	CaptureVideo	7
		CompressionVideo	8
		ScramblingVideo	9
4	Favourites	SetFavourites	10
		ViewFavourites	11
		SaveFavourites	12
5	Copy_Media	CopyMedia	13
		StoreMedia	14
6	SMS_Transfer	SendSMS	15
		recieveSMS	16
7	Basic_Operation	CreateMedia	17
		DeleteMedia	18
		EditMedia	19
		LabelMedia	20
		SortingMedia	21
		MoveMedia	22
		SearchMedia	23
		SaveMedia	24
8	View_Play_Media	ViewMedia	25
		PlayMedia	26

Table 3 summarizes the results we obtained on recovering the traceability links between source code and FM using LSI. The first column represents the threshold value of cosine similarity, column 2 represents the number of correct links recovered, column 3 represents the number of incorrect links recovered, column 4 represents the number of correct links that were not recovered, column 5 represents the total number of recovered links (correct + incorrect), and the last two columns the precision and recall values. LSI gives high precision and recall values.

Table 3. LSI results.

Cosine threshold	Correct links retrieved	Incorrect links retrieved	Missed links recovered	Total links recovered	Precision	Recall
0.5	21	3	5	24	87.50%	80.77%

6. CONCLUSIONS AND FUTURE WORK

This paper presents a method based on information retrieval namely, LSI, to establish traceability links between object-oriented source code of product variants and its FM, this traceability links is used to support conversion from traditional software development into SPL.

The results obtained in the simple reported case study proved that, in general, LSI can be used to recovering traceability links between FM and object oriented source code of product variants with high recall (80.77%) and precision (87.50%).

As future work we will consider other types of source code variations (package variation, method variation and attribute variation) and use all other information provided by the FM (such as cross tree constraints, alternative features and ect.) to recover more reliable traceability links.

7. REFERENCES

[1]. Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P. and Lahire, P. 2012. On extracting feature models from product descriptions. InProceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '12). ACM, New York, NY, USA, 45-54.

[2]. Antoniol, G., Canfora, G., Casazza, G., Lucia, A. and Merlo, E. 2002. Recovering Traceability Links between Code and Documentation. IEEE Trans. Softw. Eng. 28, 10 (October 2002), 970-983.

[3]. Benavides, D., Segura, S. and Ruiz-Cort, A. 2010. Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35, 6 (September 2010), 615-636.

[4]. Clements, P. and Northrop, L. 2001. Software product lines: practices patterns. Addison-Wesley Longman Publishing Co., Boston, MA, USA,

[5]. Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F. and Dantas, F. 2008. Evolving software product lines with aspects: an empirical study on design stability. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 261-270.

[6]. Ghanam, Y. and Maurer, F. 2010. Linking feature models to code artifacts using executable acceptance tests. In Proceedings of the 14th international conference on Software product lines: going beyond (SPLC'10), Jan Bosch and Jaejoon Lee (Eds.). Springer-Verlag, Berlin, Heidelberg, 211-225.

[7]. Gotel, O. and Finkelstein, C.1994. An analysis of the requirements traceability problem. In Proceedings of 1st International Conference on Requirements Engineering (Colorado Springs, CO). IEEE Computer Society Press, Los Alamitos, CA, 94–101.

[8]. Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). FORM: a feature oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 5(1):143–168.

[9]. Kuhn,A., Ducasse, S. and Girba,T. 2007. Semantic clustering: Identifying topics in source code. Inf. Softw. Technol. 49, 3 (March 2007), 230-243.

[10]. Lindvall, M. and Sandahl, K. 1996. Practical implications of traceability. Softw. Pract. Exper. 26, 10 (October 1996), 1161-1180.

[11]. Marcus, A. and Maletic, J. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. In Proceedings of the 25th International Conference on Software Engineering (ICSE '03). IEEE Computer Society, Washington, DC, USA, 125-135.

[12]. Pohl,K., Böckle, G. and Linden, F. 2005. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[13]. Ramesh, B. and Jarke, M. 2001. Toward Reference Models for Requirements Traceability. IEEE Trans. Softw. Eng. 27, 1 (January 2001), 58-93.

[14]. Schobbens, P.Y, Heymans, P. and Trigaux, J.C. 2006. Feature Diagrams: A Survey and a Formal Semantics. In Proceedings of the 14th IEEE International Requirements Engineering Conference (RE '06). IEEE Computer Society, Washington, DC, USA, 136-145.

[15]. Spanoudakis, G. and Zisman, A., Software Traceability: A Roadmap, in Handbook of Software Engineering and Knowledge Engineering, Chang, S. K., Ed. World Scientific Publishing Co, 2005, pp. 395-428.

[16]. Weiss, D. and Lai, C. 1999. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[17]. Ye, P., Peng, X., Xue, Y. and Jarzabek, S.2009: A Case Study of Variation Mechanism in an Industrial Product Line. ICSR. 126-136.

[18]. Ziadi, T., Frias, L., Silva, M. and Ziane, M. 2012. Feature Identification from the Source Code of Product Variants. 16th European Conference on Software Maintenance and Reengineering. 417-422.