Name Suggestions during Feature Identification: The VariClouds Approach

Jabier Martinez
SnT, University of Luxembourg
& LIP6, Sorbonne Universités,
UPMC Univ Paris 06, CNRS
jabier.martinez@uni.lu

Tewfik Ziadi
LIP6, Sorbonne Universités,
UPMC Univ Paris 06, CNRS
Paris, France
tewfik.ziadi@lip6.fr

Tegawendé F. Bissyandé
Jacques Klein
Yves Le Traon
SnT, University of Luxembourg
firstname.surname@uni.lu

ABSTRACT

Reengineering a Software Product Line from legacy variants remains a challenging endeavour. Among various challenges, it is a complex task to retrieve enough information for inferring the variability from experts' domain knowledge and from the semantics of software elements. We propose the VariClouds process that can be leveraged by domain experts to understand the semantics behind the different blocks identified during software variants analysis. VariClouds is based on interactive word cloud visualisations providing name suggestions for these blocks using tf-idf as weighting factor. We evaluate our approach by assessing its added-value to several previous works in the literature where no tool support was provided to domain experts to characterise features from software blocks.

CCS Concepts

•**Software and its engineering** → **Software reverse engineering; Software product lines;**

Keywords

Software Product Lines; Feature Identification; Visualisation; Word Clouds; Feature Naming

1. INTRODUCTION

Software Product Lines (SPLs) are reported to have numerous benefits, among which their established capabilities to reduce time-to-market, to increase productivity and to ensure product quality [12]. Unfortunately, the barriers to achieve SPL adoption are equally numerous, ranging from organizational issues to purely technical details. In practice, before considering assuming the high up-front investment towards systematic and planned reuse with SPL, developers more commonly rely on ad-hoc reuse techniques such as *copy-paste-modify* [3]. When managing variants becomes challenging, developers may consider reverse engineering the underlying SPL to benefit from its advantages in validating, generating and maintaining new variants. Our work aims at facilitating the first encounter between the domain experts and the variants so as to help in understanding the semantics hidden in the variants and in the variability among them.

In the SPL reengineering context, the first objective in *extractive* approaches [8] is *feature identification* where a set of artefact variants are taken as input for comparison and analysis without assuming a complete upfront knowledge of the features. A feature is then a prominent or distinctive characteristic, quality or user-visible aspect of a software system or systems [7]. In practice, automated comparison approaches will identify distinguishable blocks shared by software variants, and will require domain experts to manually map them with actual features [2, 9, 10, 18]. To that end, domain experts must look at the elements of the blocks, understand their semantics, and guess the functionality that each block provides when present in a variant.

Although nascent in the SPL engineering field, the use of visualisation techniques have proven helpful in supporting stakeholders with their work tasks. We present VariClouds as a practical approach to support domain experts feature labeling during feature identification. In other words, we help experts to name the identified blocks. This support is based on a summarization of the content of each identified block in order to suggest feature names. We leverage word clouds, a widely adopted visualisation for textual data [16]. Blocks can be renamed by interacting with the suggested words of the clouds. We also propose an automated approach for heuristically assigning the names.

VariClouds aims at being generic (e.g. source code, design models or component based systems), by assuming that an artefact can be decomposed into elements and that words can be automatically extracted from them. Our implementation of VariClouds is available in the BUT4Reuse framework [10].

We assess the soundness of VariClouds via experiments for answering the following research questions:

- RQ-1: To what extent the blocks of implementation elements (e.g. source code elements) can automatically provide insights that correspond to expert judgement about the semantics of these blocks?
- RQ-2: Is the word cloud visualisation paradigm effective for naming during feature identification?

The paper is structured as follows: Section 2 presents background concepts and our motivation. Section 3 details VariClouds and Section 4 presents its evaluation. Section 5 discusses the threats to validity. Section 6 presents related work and Section 7 concludes the paper.

SPLC '16, September 16-23, 2016, Beijing, China

DOI: http://dx.doi.org/10.1145/2934466.2934480

2. BACKGROUND AND MOTIVATION

2.1 Feature identification

To distinguish features and their associated artefact elements, researchers have proposed to analyze and compare artefact variants for identifying their common and variable parts [2, 6, 9, 10, 13, 18, 21]. We refer to each of such distinguishable parts as a *Block* [10]. Existing techniques are based either on static analysis using diff match policies, or on dynamic analysis of the system. Some approaches further leverage information retrieval (IR) techniques. In the use of VariClouds, we assume an effective block identification technique is available and yields the necessary blocks for use as part of the feature identification process. Thus, it is important to clarify that block identification techniques themselves are out of the scope of this paper. In Figure 1, we illustrate the block identification process via an example where each artefact (ellipses) is represented as a set of elements (rhombuses) and the blocks are identified as the different intersections obtained through the *Interdependent elements* algorithm [20]. For example, `Block 0` is common to all artefacts, `Block 1` is shared by all artefacts except `Artefact 1`, and `Block 4` is specific to `Artefact m`. This algorithm will be the one used in the case studies of this work.

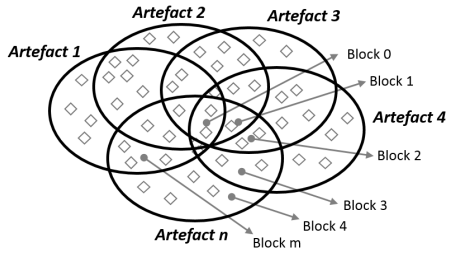


Figure 1: Artefact variants and identified distinguishable Blocks. Source [10]

2.2 Word Clouds and Weighting Factors

Word clouds gained momentum in the Web as aggregators of activity, as a means to measure popularity, and as a mechanism for social tagging/indexing [16]. Word clouds have been also used for text summarization and analysis in several domains. The principle underlying word clouds is a weighting factor of the words appearing in a document. This weighting factor is used to change the relevance of the word in the visualisation, typically by assigning larger font sizes to the more weighted words. *Term frequency* (noted `tf`) is a metric consisting in giving more relevance to the terms that appear with more frequency in a document d. When dealing with a set D of several documents $d_1, ..., d_n$, *term frequency-inverse document frequency* (`tf-idf`) is another metric used in IR. For a document d, `tf-idf` penalizes common terms that appear across most of the documents in D and encourages those terms that are more specific to d. In this work, we used the formulas presented in Equations (1). Inverse document frequency `idf` measures how much rare or common a term is across all the documents using a logarithmic scale, and `tf-idf` uses raw term frequency `tf` multiplied by `idf` to penalize or encourage a term depending on its occurrence across D.

$$idf(term_i, D) = \log\left(\frac{|D|}{|\{d \in D : term_i \in d\}|}\right)$$
$$tf\text{-}idf(term_i, d, D) = tf(term_i, d) \times idf(term_i, D) \quad (1)$$

2.3 Motivation: Why VariClouds?

Most contributions in the literature of SPL migration, including our own previous work, overlook the feature naming step during identification. Since current automated blocks identification processes are not focused on the naming problem nor in support for final users through visualisation, often this step is carried out manually and with no defined process [1, 2, 9, 10, 18, 21]. The lack of support for naming in state-of-the-art approaches is an important threat to their efficiency and challenges their adoption. In fact, migration scenarios can vary in number of blocks, features, stakeholders and in the degree of availability of the domain knowledge. Several domains of expertise are often required to build a product and different stakeholders are responsible for different functionalities. In this context, we can assume that domain knowledge about the features of legacy variants is scattered across the organization. Our work, does not assume domain knowledge about the features. This assumption is perhaps too pessimistic and is thus limited to a few scenarios. However, assuming that an exhaustive list of features can be easily elicited from domain knowledge is also too idealistic. Our work is motivated by the needs to 1) close the gap for providing support for the manual task of naming during feature identification by leveraging legacy variants and 2) speed up and improve the quality of feature identification.

3. THE VariClouds APPROACH

We detail VariClouds by first explaining how words are retrieved from product variants, and then overviewing the approach phases from the perspective of a domain expert.

3.1 Retrieving the words

Implementation specificities for different artefact types (e.g. source code or models) must be dealt with through the *adapter* concept [10]. An adapter is responsible for decomposing a given instance of its associated artefact type into a set of Elements. To support VariClouds, each adapter must be enriched for yielding the words exposed for each Element. A word is therefore a term inferred from an implementation element (e.g. the name of a java method). We overview how the relevant adapters to our case studies were implemented:

Source code: Source code is represented in the form of their Abstract Syntax Trees (AST). AST nodes, mainly classes and methods, usually have meaningful names provided by developers, which can be exploited to retrieve relevant words regarding the implemented functionality. In this work, we use names of classes, methods and declared fields.

Models: Meta-Object Facility models can be decomposed in atomic model elements [9, 11]. We consider the *Class*, *Attribute* and *Reference* which are the mainly used concepts to define domain specific languages (DSL). To get the name of a Class instance we use its *EMF Item Label Provider* defined by the DSL implementation. For Attributes, we get their value in string format. We ignored the text from References in the current implementation. Finally, we tokenize the text of Class and Attribute.

Eclipse: Eclipse plugins are the components that provide different functionalities targeting different development scenarios. The Eclipse adapter decomposes an Eclipse installation in its set of plugins. For each Plugin we take its name as defined by the plugin providers in the plugin metadata. We tokenize the plugin name to obtain the set of words.

3.2 Using VariClouds

An overview of the process is shown in Figure 2. The inputs are the legacy variants and the blocks obtained from an automatic block identification process. The outputs are the emerging vocabulary and insights for helping in naming during feature identification. VariClouds define two phases for the domain experts: 1) Preparation of the word clouds and 2) Block naming. Each phase is explained using as example the Vending Machine Statecharts case study [11].

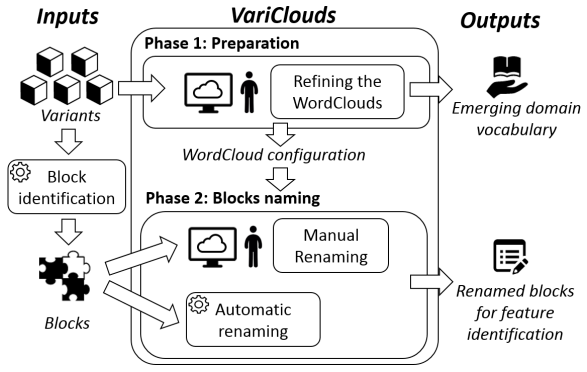


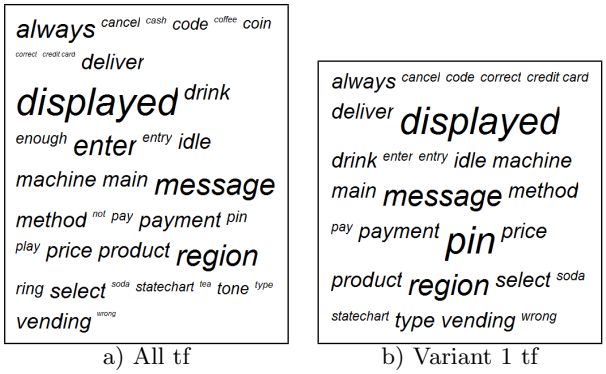
Figure 2: Overview of the VariClouds process

3.2.1 Phase 1: Preparation of the word clouds

The word clouds can be created using the words from any set of Elements. Figure 3a presents a word cloud using `tf` considering all the variants. Therefore, it is a summarization of the whole family of variants. Figure 3b displays the words which are frequent `tf` in `Vending Machine 1`. We could also display another word cloud of a variant but using `tf-idf` showing the words that make this variant special regarding the other variants. The domain experts can explore these kind of word clouds in order to decide to use word filters or to refine the configuration of a given filter. Filters are intended to provide more meaningful word clouds by preprocessing the words of the elements provided by the adapter. In this work we have implemented a set of traditional filters (shown in Figure 4) which activation are optional to the domain experts. We also provide an extension to add new filters.

3.2.2 Phase 2: Block naming

Blocks are obtained using the block identification algorithm. Then, the domain experts use interactive word cloud constructed with the elements of each block in order to name each block. The hypothesis of the VariClouds approach for block naming is that relevant words are those that make



a) All tf b) Variant 1 tf

Figure 3: Word cloud examples of vending machine variants

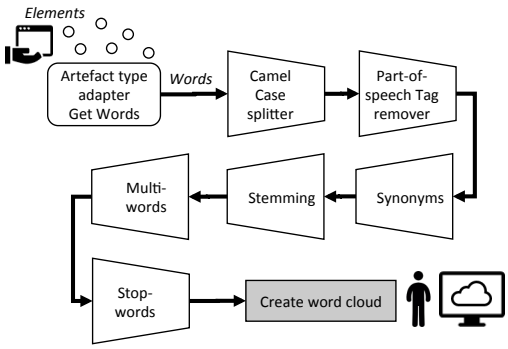


Figure 4: Optional filters from the Elements' words to the final word cloud

each block special regarding the rest of the blocks. For this reason, `tf-idf` weighting factor is used. Figure 5 shows the `tf-idf` word clouds of the blocks identified while automatically analysing and comparing the vending machine statechart variants. The domain experts can interact on the suggested names to set the block names.

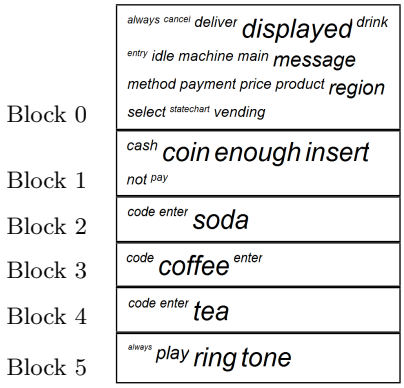


Figure 5: Word clouds of the identified blocks in the Vending Machine Statechart variants

VariClouds also proposes an automatic algorithm to set the names of each block for further refinement. The parameter k is the initial number of words to be used when renaming. Using $k = 1$ each block name will initially have only the word with highest `tf-idf` score. As we will present later, given the empirical results of our case studies, by default we suggest to use at least $k = 2$. For each block, we assign the concatenation of the first k words with the highest `tf-idf` scores. When renaming all blocks, it is possible that two or more blocks have the same name. We avoid name conflicts by iteratively appending the word with the next highest score until there are no conflicts. If there are no more remaining words in the ranking and there are still conflicts, different numbers are appended.

4. EVALUATION

In order to evaluate RQ-1, we selected six previously published case studies where the name of the features are reported. This provides us a ground truth to evaluate the naming process given that the result of the manual naming is available in these publications. ***ArgoUML*** are 10 variants [2, 20] with an average of 141 KLoC of Java source code. ***Notepad*** are 8 variants [21] with an average of 691 LoC. ***Draw application*** are 12 variants [6] with an average of 200 LoC and ***Mobile media*** are 8 variants [19] with 1.7

KLoC in average. ***In-Flight Entertainment*** systems are 3 model variants [9] with an average of 7182 classes. The term classes should not to be confused strictly with UML Classes. The different types of classes of a model are defined in the DSL specification. ***Vending Machine Statecharts*** are 6 variants [11] with an average of 17 classes. ***Banking Systems*** are 3 variants [9] with an average of 25 classes.

We evaluate RQ-2 using the ***Eclipse*** case study [10] with domain experts. We consider 12 Eclipse IDE variants of the Eclipse Kepler release with an average of 609 plugins. We provide time measurements and qualitative results.

4.1 RQ-1: Quality of the word clouds

Mean Reciprocal Rank (MRR) [17] captures how early the relevant result appears in a ranking. It is considered that MRR measures users' effort in their search length. This is the case of word clouds where the user looks at the largest name, then to the second largest and so on. The reciprocal rank (RR) of a given feature name in its associated block is calculated as $1/rank_i$ where $rank_i$ is the position in the ranking where the feature name from the ground truth appears. Let F be all the features, MRR formula is presented in Equation 2. A MRR of 1 means that all feature names were the largest in the word cloud of its associated block.

$$MRR = \frac{1}{|F|}\sum_{i=1}^{|F|}\frac{1}{rank_i} \quad (2)$$

In the worst case scenario the name is not found in any rank position. In this special case we consider $1/rank_i = 0$ and we will discuss these cases separately. The core components that are common to all variants use to be encompassed in a feature which name are *Core* or *Base*. These names use not to be part of the emerging vocabulary and for this reason, we refer to as MRR2, the MRR metric where the *Core* or *Base* feature is excluded from the set F. Table 1 shows the results. If a name is not found we denote it with the empty set symbol $\emptyset$. $MRank$ shows the average of the rank of the features without considering the core feature and the features that were not found.

The mean of MRR2 in this set of case studies is 0.79. This result is promising and, therefore, guarantee the soundness of the VariClouds which will show the largest words for the most relevant terms thus reducing users' search effort. The mean of MRank is 1.62 which indicates that, ignoring the core features and the not found features, the feature names appear in the first 2 positions of the ranking. Despite the promising average results, there are some unsuccessful results that cannot be neglected. We discuss the two main reasons for unsuccessful RR results with a special focus on the two not-found feature names ($rank = \emptyset$).

Mismatch between domain names and implementation details: In the case of *Cognitive support* in ArgoUML, there is a complete mismatch between the feature name and the vocabulary emerging from the implementation. The largest words of its associated blocks are *cr*, *criticized* and *design*. *Cognitive support* is actually implemented in the Critics subsystem [4].

Filters undesired effect: In the case of *WithdrawWithoutLimit* of the Banking systems, *with* and *without*, despite being prepositions, were important words which were discarded by the parts-of-speech tag remover. By deactivating this filter, MRR2 can be 0.875 if we consider *without* similar to WithdrawWithoutLimit (*withdraw* and *limit* are the rank items following the first one that is *without*). In the preparation phase, the camel case splitter was activated for all the source code based case studies. However, in the case of the Notepad case study, camel case is not the main style followed by the developers. Fortunately, it did not affect the RR (e.g. there are methods called *finD* or field declarations called *findNexT* therefore the largest words for the Find feature were *find*, *fin* and *d*.

4.2 RQ-2: Word Clouds as visualisation

Plain textual representations were used in previous works to characterize implementation elements (e.g. [1, 2, 11, 21]). The textual representation of all the elements of each block were presented without any summarization. In previous work we considered the scenario of Eclipse variants [10]. We requested the expertise of 3 domain experts who analysed, independently from each other, the Elements' textual representations of the 61 Blocks that were identified. As reported [10], this manual task took an average of 51 minutes.

For the evaluation of RQ-2 we consider the Eclipse case study. We evaluated VariClouds with 3 domain experts on Eclipse which are not the same persons as in previous work [10]. Independently from each other, they performed feature identification tasks using the word clouds as support for the element textual representations of each block. We asked to report their mental process for block naming. All of them stated the following process: 1) Read very quickly the textual descriptions of the elements (beginning, middle and end) to have an initial clue about the logic and identify a word in their mind, 2) Read the word cloud largest

Table 1: Evaluation of the quality of the word clouds

	MRR2	MRR	MRank	Rank of each feature, $\emptyset$ for not found
ArgoUML	0.71	0.63	1.57	*Core* ($\emptyset$), Logging (1), Activity diagram (1), State diagram (1), Collaboration diagram (1), Sequence diagram (4), Use case diagram (1), Deployment diagram (2), Cognitive support ($\emptyset$)
Notepad	0.83	0.62	1.33	*Base* ($\emptyset$), Cut-Copy-Paste (1), Find (1), Undo-Redo (2)
Draw application	1.00	0.80	1.00	*Base* ($\emptyset$), Line(1), Rect (1), Color (1), Wipe (1)
Mobile media	0.63	0.57	3.33	*Core* ($\emptyset$), ExceptionHandling (1), LabelMedia (2), Sorting (6), Favourites (1), Photo (2), Music (2), Video (1), SMS (1), CopyMedia (14)
In-Flight Entertainment	1.00	0.66	1.00	*Core* ($\emptyset$), Wi-Fi (1), ExteriorVideo (1)
Vending Machines	0.76	0.69	1.83	*Main* (4), Soda (1), Coffee (1), Tea (1), Cash payment (4), Credit card payment (3), Ring tone alert (1)
Banking systems	0.62	0.50	1.33	*BankCore* ($\emptyset$), CurrencyConverter (2), WithdrawWithLimit (1), Consortium (1), WithdrawWithoutLimit ($\emptyset$)
Mean:	*0.79*	*0.63*	*1.62*	

names and contrast them with the one in their mind, 3) Select one from the word cloud or use the guessed one, and 4) Optionally refine the selected word with an extra word.

The average time was reduced in this case study from 51 to 28 minutes (≈45% decrease). All of them stated that the word clouds were useful for assigning the names. Specially when they were not completely sure about the logic of the block. They stated that the word clouds served as reinforcement or confirmation for the naming decision. According to the time reduction and their mental process, we can say that word clouds reduce domain experts' comprehension time and help them to be more confident with the naming decisions while accelerating the process.

5. THREATS TO VALIDITY

We cannot assure that the findings can be generalized. The feature names from SPL literature and the words used by the developers that implemented the artefacts are conditioned to human factors (i.e. decided by experts). Also, each case study considers variants that belongs to the same developers. In the Eclipse case study, even if we consider that the new domain experts have a very similar background to the previous domain experts, we cannot assure that the difference is because they have a different set of skills.

VariClouds claims for genericity in supporting artefact types but it assumes the existence of an adapter. Some artefacts can have the limitation that their elements may have absence of meaningful names (e.g. compiled or obfuscated). In the same way, VariClouds assumes the existence of a block identification algorithm. BUT4Reuse [10] provides a set of these algorithms. Nevertheless, it is accepted that there are many factors that affect the quality of the results of these algorithms (e.g. [6, 9, 14, 21]). The research conducted to propose VariClouds is complementary to block identification given that VariClouds is focused in the interaction and visualisation for domain experts.

6. RELATED WORK

Apart from the works already mentioned in Section 2.3, a related work can be found in a small remark of Shatnawi et al. [15]. For the purpose of readability of an example, they assign names based on "*the most frequent tokens*" of the identified blocks. The objective of their paper is other so they do not evaluate these namings and they do not provide details about the process. We conducted experiments using `tf` to verify our intuition that `tf-idf` provides better results.

Davril et al. [5] presented a feature naming approach as part of their automatic feature model extraction method. As input they focused on large sets of product descriptions in natural language. Their approach is fully automatic while we propose a visualisation to include the domain experts early in the naming process.

7. CONCLUSION

VariClouds is an approach that extensively uses word cloud visualisations in order to provide insights of the emerging vocabulary and variability from a set of variants. It is designed for helping domain experts in feature identification and naming. We evaluated it in several case studies dealing with different artefact types to show its soundness and genericness. As further work we aim to evaluate the use of weights for different Element types. For example, in source code, we can consider that words belonging to a class name have more relevance than words from a method name.

8. ACKNOWLEDGMENTS

Funded by FNR Luxembourg under the AFR grant agreement 7898764. The authors would also like to thank Arthur Joanny for his great work and help in the implementation.

References

[1] Ra'Fat Al-Msie'deen, Abdelhak-Djamel Seriai, Marianne Huchard, Christelle Urtado, and Sylvain Vauttier. Mining features from the object-oriented source code of software variants by combining lexical and structural similarity. In *ICSR*, 2013.

[2] Ra'Fat Al-Msie'deen, Abdelhak-Djamel Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Hamzeh Eyal Salman. Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing. In *SEKE*, 2013.

[3] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. A survey of variability modeling in industrial practice. In *VaMoS*, 2013.

[4] Marcus Vinicius Couto, Marco Tulio Valente, and Eduardo Figueiredo. Extracting software product lines: A case study using conditional compilation. In *CSMR*, 2011.

[5] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *ESEC/FSE*, 2013.

[6] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Enhancing clone-and-own with systematic reuse for developing software variants. In *ICSME*, 2014.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.

[8] Charles W. Krueger. Easing the transition to software mass customization. In *PFE*, 2001.

[9] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Automating the extraction of model-based software product lines from model variants. In *ASE*, 2015.

[10] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up adoption of software product lines: a generic and extensible approach. In *SPLC*, 2015.

[11] Jabier Martinez, Tewfik Ziadi, Jacques Klein, and Yves Le Traon. Identifying and visualising commonality and variability in model variants. In *ECMFA*, 2014.

[12] Linda M. Northrop, Paul C. Clements, et al. A Framework for Software Product Line Practice, Version 5.0. www.sei.cmu.edu/productlines/framework.html, 2009.

[13] Julia Rubin and Marsha Chechik. Combining related products into product lines. In *FASE*, 2012.

[14] Julia Rubin and Marsha Chechik. N-way model merging. In *ESEC/FSE*, 2013.

[15] Anas Shatnawi, Abdelhak Seriai, and Houari A. Sahraoui. Recovering architectural variability of a family of product variants. In *ICSR*, 2015.

[16] Gene Smith. *Tagging: People-powered Metadata for the Social Web*. New Riders Publishing, 2007.

[17] Ellen M. Voorhees. The TREC-8 question answering track report. In *TREC*, 1999.

[18] Yiming Yang, Xin Peng, and Wenyun Zhao. Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In *WCRE*, 2009.

[19] T. Young and G. Murphy. Using AspectJ to build a product line for mobile devices. In *AOSD*, 2005.

[20] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. Feature identification from the source code of product variants. In *CSMR*, 2012.

[21] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *SAC*, 2014.