Reverse Engineering Variability from Requirement Documents Based on Probabilistic Relevance and Word Embedding

Yang Li, Sandro Schulze, Gunter Saake
Otto-von-Guericke Universität Magdeburg, Germany
{yang.li,sandro.schulze,saake}@ovgu.de

ABSTRACT

Feature and variability extraction from different artifacts is an indispensable activity to support systematic integration of single software systems and Software Product Line (SPL). Beyond manually extracting variability, a variety of approaches, such as feature location in source code and feature extraction in requirements, has been proposed to provide an automatic identification of features and their variation points. Compared with source code, requirements contain more complete variability information and provide traceability links to other artifacts from early development phases. In this paper, we propose a method to automatically extract features and relationships based on a probabilistic relevance and word embedding. In particular, our technique consists of three steps: First, we apply word2vec to obtain a prediction model, which we use to determine the word level similarity of requirements. Second, based on word level similarity and the significance of a word in a domain, we compute the requirements level similarity using *probabilistic relevance*. Third, we adopt hierarchical clustering to group features and we define four criteria to detect variation points between identified features. We perform a case study to evaluate the usability and robustness of our method and to compare it with the results of other related approaches. Initial results reveal that our approach identifies the majority of features correctly and also extracts variability information with reasonable accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; • **Software and its engineering** → **Software product lines**;

KEYWORDS

Feature Identification, Variability Extraction, Reverse Engineering, Software Product Lines, Requirement Documents

ACM Reference Format:
Yang Li, Sandro Schulze, Gunter Saake. 2018. Reverse Engineering Variability from Requirement Documents Based on Probabilistic Relevance and Word Embedding. In *SPLC '18: 22nd International Systems and Software Product Line Conference, September 10–14, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3233027.3233033

SPLC '18, September 10–14, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-6464-5/18/09…$15.00
https://doi.org/10.1145/3233027.3233033

1 INTRODUCTION

A *Software product line (SPL)* allows to develop a family of products that share common functionalities and characteristics satisfying the specific needs of a particular market segment [4]. To this end, Software Product Line Engineering (SPLE) describes a systematic development process taking commonalities and differences (in terms of features) between particular products into account [7, 30]. A feature in this context is a user-visible increment in functionality [2]. While SPLs have been proven to be beneficial, they are rarely introduced for initial development due to a) high upfront cost and b) missing certainty whether a large amount of variants is needed. Instead, traditional and more unstructured techniques for reuse, such as *clone-and-own* [9, 33] are used to cope with diverging products in the beginning. However, at this point, information about commonalities and variability is usually not explicitly given, but rather hidden within the artifacts. Hence, in case of a transition from unstructured to structured reuse using SPLs, this information needs to be recovered.

Reverse engineering techniques, such as feature location and feature extraction, are common means to support automatic extraction of features and, in some cases, even their variability information. While feature location has been subject to intense research [8], it exhibits crucial limitations that hinders applicability in an SPL context. In particular, (1) the majority of existing feature location techniques focus on source code in single software systems, and thus, do not recover variability information, and (2) it may cost additional effort to recover this information for artifacts of earlier development stages. In contrast, feature extraction focus on requirements as artifact to recover variability information [1, 13–15, 20, 32, 37–39], because requirements (1) contain more comprehensive information about commonalities and variability, and (2) establish traceability links to other artifacts of later development phases (e.g., source code).

However, in a recent survey, we observed that current approaches are rather immature, that is, they are lacking accuracy (regarding the features identified), suffer from incomplete variability information and fail to achieve a high degree of automation, which hinders the applicability in practice [21]. Some of these approaches highly depend on the syntactic information of sentences (i.e., parse trees) [13–15, 32, 37, 38], which require manual intervention even though external tools such as Stanford Parser [25] are integrated. Other approaches rely on similarity of each pair of sentences in requirements. One part of these approaches apply external knowledge databases such as WordNet [27] to identify synonyms and compute the similarity [14, 15, 37, 38], while another part utilizes traditional distributional semantic models (DSMs), for instance Vector Space Model (VSM) [20] and Latent Semantic Analysis (LSA) [1, 39], to calculate the similarity. The shortcoming of applying WordNet or other external lexical database of structured semantic knowledge is

that the high quality of the resources is not available for all words, especially for domain-specific technical terms. Meanwhile, VSM and LSA can not achieve synonymy and polysemy, which decreases the similarity accuracy. Moreover, traditional DSMs can be considered as *count* models as they count co-occurrences among words by operating on co-occurrence matrices. Consequently, they usually achieve worse results than neural-network-based natural language processing architectures which can be seen as *predict* models [3].

In order to overcome these limitations, we propose a technique to extract features from word level semantics to requirement level semantics. Our technique is mainly comprised of : 1) a neural word embedding model as a predict model to obtain word level semantic information; 2) a probabilistic relevance framework of information retrieval to obtain requirement level semantic similarity; 3) a clustering algorithm combined with pre-defined criteria to extract features and variability information. We aim to create a generic framework to identify features and extract variation points from textual requirements without manual intervention, reducing the dependence on prior knowledge of natural language and external knowledge databases. In particular, we make the following contributions:

- We implement a full-automated approach which is capable of extracting features and variability information.To this end, we integrate a prediction model into probabilistic relevance framework to process the raw requirements instead of using DSMs.
- We conduct a case study with product requirements and also provide a comparison of our technique with other approaches. In a nutshell, we show that our approach is capable of achieving relatively high accuracy and minimizes manual intervention regarding the extracted features and variability information.

2 METHODOLOGY OVERVIEW

In this section, we provide an overview how our technique processes requirements in order to obtained detailed feature and variability information. Moreover, we introduce briefly the particular techniques we use and how they are applied in our context. We provide an overview of the entire process in Figure 1.

Initially, the input documents (i.e.,textual requirements specifications) are pre-processed, that is, we remove stop words in these requirements and implement *lemmatization* operation, which usually aims to remove inflectional endings and to return the base or dictionary form of a word, by using NLTK [5]. Note that, the requirement specifications we used as input are from different variants. Hence, the input documents includes a *mapping* between *requirements* and *variants (R-V Mapping)*.

Subsequently, we utilize a semantic similarity network to obtain the similarity for each pair of requirements by two steps: First, we utilize a pre-trained word embedding model (*word2vec* in our current approach) to obtain the word level similarity of requirements (cf. Section 3.1); then, we extend a probabilistic relevance framework (*Best Match 25+* in our current approach) to achieve the requirement level similarity, based on the word level similarity and word weighting (cf. Section 3.2). The output of this step is a requirement similarity matrix.

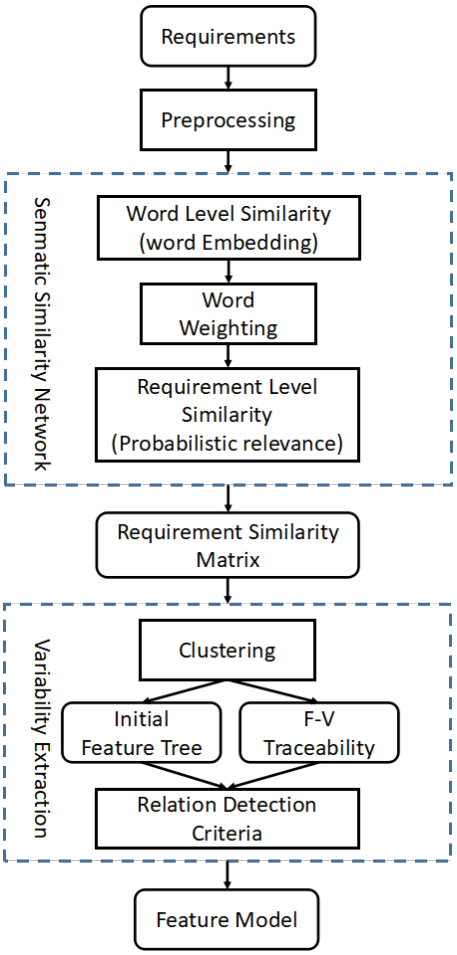

Figure 1: The flow chart of feature and variability extraction

Finally, we feed the requirement similarity matrix into a clustering algorithm (*Hierarchical Agglomerative Clustering* in our current approach), resulting in an initial feature tree structure without variability information. As a result, another *mapping* can be achieved, that is, between *features* and *requirements (F-R Mapping)*. Based on the R-V Mapping and F-R Mapping, we can deduce a *traceability* link between *features* and *variants (F-V Traceability)*. Afterwards, based on four pre-defined criteria taking advantage of F-V Traceability link, the relationship between features can be extracted.

In the context of feature extraction from requirements, we regard features as domain artifacts, and thus, we assume that several requirements, related to the same functionality, belong to a particular feature. Next, we explicate the specific techniques in greater detail (cf. Section 3 and Section 4).

3 SEMANTIC SIMILARITY NETWORK

In this section, we provide details how we integrate word embedding and topic words into a probabilistic relevance framework. Our goal is to achieve a semantic similarity network of requirements that takes both, word level as well as requirement level similarity into account.

3.1 Word Level Similarity

Inspired by the work of Mikolov et al. [26], we use Word2Vec, a two-layer neural network that is used to produce word embeddings (i.e., vectors), to obtain the word semantic similarity. The input of Word2Vec is a text corpus. Given enough text data and contexts, Word2Vec can achieve highly accurate meanings of the words appearing in the corpus and establish a word's association with other words. The output is a set of vectors, that is, vectors of similar words are grouped together in a semantic vector space. We measure cosine value between two word vectors to evaluate the similarity of each pair of words, which is called *cosine similarity*. For two words being entirely similar, the cosine value of their word vectors is 1; otherwise, the cosine value is 0. For instance, consider the words "temperature" and "heat". By using the pre-trained Word2Vec model from [26], the cosine similarity of "temperature" and "heat" is 0.56, accurately indicating the semantic correlation of these two words. More formally, the word similarity is defined by:

$$wordSim(w_1, w_2) = cosine(\vec{w_1}, \vec{w_2}) \tag{1}$$

Although we can achieve a highly accurate word similarity with Word2Vec, it still exhibits the limitation that a vector space is not capable of gaining the association of all words we consider, since the size of a single text corpus is limited. Even if the size of one corpus is large enough, it hardly contains all relevant words for different SPLs, especially for domain-specific terms. Hence, the similarity value of a pair of words can not be determined, if one of these words does not occur in the corpus that is used to train the word vector space. To overcome this limitation, we map such missing words to random vectors, while recording which out-of-range word is mapped to which random vector [19].

3.2 Requirement Level Similarity

After obtaining the word level similarity, we extend it into requirement level similarity based on two characteristics: The *distribution of words* in each requirement; and the *significance* of each word, which is used to measure how much a word contributes to the semantics of a requirement in a specific domain. To get the most out of these two characteristics or even add more (e.g., structural information of requirements), we apply BM25+, an extension of Best Match 25 (BM25)[34]. BM25+ is a so called probabilistic relevance framework for document retrieval which improves the accuracy of processing very long requirements compared with BM25 [24] and eventually results into a semantic similarity network for all pairs of requirements (from the input document).

In a nutshell, BM25+ utilizes *Term Frequency - Inverse Document Frequency (TF-IDF)* weighting technique to measure how much a word contributes to the relevance of two short texts. However, the technique exhibits two limitations: (1) the words in each pair of texts can only contribute to similarity when the words fit each other perfectly; (2) the statistical co-occurrence of words can not entirely represent the significance of words in a text, especially for domain specific terms in requirement documents.

Basic function. We overcome the above-mentioned limitations by using Word2Vec, thus, resolving limitation (1). Moreover, we establish a combination of topic words in the requirement documents and *Inverse Document Frequency (IDF)*, thus, addressing limitation (2). Consequently, the basic function used to compute the similarity of each pair of requirements based on BM25+ is as follows:

$$\begin{aligned} bmSim(r_1, r_2) = & \sum_{w_1 \in \{r_1\}} Weight(w_1) \\ & \times \left(\frac{semSim(w_1, r_2) \times (k_1 + 1)}{semSim(w_1, r_2) + k_1 \times (1 - b + b \times \frac{lr_2}{avglr})} + \delta\right) \end{aligned} \tag{2}$$

with

- $b \in [0,1], k_1 \in [1.2, 2], \delta \in [0, 1.5]$;
- lr_2 is the length of the requirement r_2 (i.e., the quantity of words in r_2);
- $avglr$ is the average length of all requirements in the input requirement documents;
- $semSim(w_1, r_2) = \max_{w_2 \in r_2} wordSim(w_1, w_2)$;
- $Weight(w_1) = \begin{cases} Preset\ Value, & w_1 \in Topic\ words; \\ IDF(w_1), & otherwise. \end{cases}$

Our basic function inherits the main characteristics of BM25+. In detail, we also apply k_1 to get a much smoother function, utilize b together with lr_2 and $avglr$ to perform requirement length normalization, and use δ to avoid very long documents being overly penalized [24, 34]. However, the key differences are that we introduce word level similarity based on Word2Vec and topic words for weighting into BM25+.

The function $semSim(w, r)$ is utilized to compute the semantic similarity between each word w and the requirement r. For the resulting similarity values, we search for the maximum word similarity for each word in r. This word level similarity is introduced to replace term frequency used in original BM25+, and thus, contributes to gain more semantic information.

In order to measure the significance of words (represented by $Weight(w)$) reasonably, we apply two kinds of methods: First, traditional IDF as it is widely used to handle this weighting problem and is also utilized in BM25+. Second, we analyze the requirement documents to extract *topic words* that are the smallest units of domain knowledge representing and describing a certain SPL. We argue that the *subject* and *object* in the sentences of requirements are of more topic information and domain knowledge. Hence, we use Stanford Parser [25] to automatically extract all subjects and objects of each sentence and clause in the requirement documents as topic words. For example, in Table 1, the word "this" also can be regarded as a topic word, in case it is a subject or object of a sentence. However, this kind of pronouns usually belong to stop words as it provides less information than other words. Thus, we remove stop words (e.g., the, this, a, etc) from topic words, which enhances the degree of automation, as no manual filtering has to be applied. Moreover, words with very low IDF value are also excluded from topic words.

In a nutshell, if a word belongs to the set of topic words, we assign a high preset weight value to it; otherwise, the word is weighted by IDF algorithm.

Table 1: Example of topic words

ID	Requirements	Topic Words
R1	If the finger protection has not been triggered, the LED does not light up.	finger, protection, LED
R2	When the alarm system is activated, this is indicated by the light of the LED.	alarm, system, this, light, LED

Ultimate function. From equation 2, we obviously find that the similarity result of a pair of requirements is different, when we utilize one of the compared requirements as a reference, respectively (i.e., $bmSim(r_1, r_2) \neq bmSim(r_2, r_1)$). The reasons are (1) different weight values of words and (2) the different lengths of the two requirements. For instance, consider the requirements R1 and R2 in Table 1; using R2 as the reference (i.e., $bmSim(R_1, R_2)$) to compute the requirements similarity would not take weight values of (topic) words in R2 into account (as indicated by Equation 2). Moreover, assume we have two requirements with different lengths and the words in the shorter requirement constitute a subset of words contained in the longer one. If the longer requirement would be chosen as a reference, only the similarities of words belonging to the subset (i.e., the shorter requirement) would be computed neglecting other words in the longer requirement, which leads to an unreasonable high similarity value. However, in both of the above cases we would obtain different and inaccurate similarity values but also face a loss of semantic information. Thus, to mitigate the impact which requirement is chosen as reference and to obtain an equal and accurate similarity value, we calculate the average of $bmSim(r_1, r_2)$ and $bmSim(r_2, r_1)$. As a result, the final function for requirement similarity calculation is as follows:

$$reqSim(r_1, r_2) = \frac{1}{2} \times \left(\frac{bmSim(r_1, r_2)}{\sum_{w_1 \in \{r_1\}} Weight(w_1)} + \frac{bmSim(r_2, r_1)}{\sum_{w_2 \in \{r_2\}} Weight(w_2)} \right) \quad (3)$$

By applying Equation 3, we achieve a semantic similarity network (i.e., a symmetric matrix) which consists of the similarity values of each pair of requirements.

4 VARIABILITY EXTRACTION

After obtaining the semantic similarity network of requirements, we apply a clustering algorithm and four pre-defined criteria to extract variability information which is crucial for constructing a complete feature model. In the following subsection, we provide details on how variability information is extracted.

4.1 Feature Extraction

To extract the tree-like structure of a feature model, we apply Hierarchical Agglomerative Clustering (HAC) [35]. In a nutshell, similar requirements (i.e., requirements with similar functionality) are grouped into same cluster that is regarded as a feature and we reuse the hierarchical structure produced by HAC to form a feature tree.

First, we utilize the similarity values for each pair of requirements as clustering criterion, taking the semantic similarity network of requirements as input for HAC. Second, the pairwise distance (i.e., dissimilarity) of requirements is measured to find the closest pair of requirements and merge requirements with shortest distance into a single new group, that is, a set of requirements. Then, we compute distances between the newly created group and other requirements. Afterwards, the process of calculating distance and merging requirements is repeated until all requirements are assigned to a group, resulting in a hierarchical clustering tree. Third, we flatten the hierarchical tree to merge clusters based on an *inconsistency coefficient* [16]. More precisely, we compute an inconsistency value between a cluster node and all its descendants in the hierarchical clustering tree. If this value is less than or equal than an *inconsistency threshold*, then all its leaf descendants belong to the same feature. Finally, we inherit the hierarchical tree from the node with the longest distance in extracted features up to root node rather than the whole hierarchical tree by HAC, which reduces the complexity of the feature tree. In detail, the similar features are grouped together again to simplify and generate the tree structure.

Inconsistency threshold selection. How to define an appropriate threshold plays a crucial role in feature extraction by HAC, since the inconsistency threshold makes a huge difference regarding the number of features and the accuracy of the extraction process. In clustering theory, internal validity indices are utilized to evaluate the clustering results when the ground truth of clusters is unknown. Hence, in order to achieve an optimal inconsistency threshold, we apply *internal validity indices* to estimate the accuracy of the extracted features by HAC in terms of different inconsistency threshold. We implement this by defining a fixed step (i.e., the precision of the inconsistency threshold), letting the inconsistency threshold rise from the value equal to the pre-defined fixed step by that fixed step and using every inconsistency threshold in HAC to extract features. In this process, the number of features extracted by HAC decreases, when the inconsistency threshold increases. Hence, the biggest inconsistency threshold is the one which leads the number of extracted features to two, since just one feature being extracted in a certain SPL is unreasonable. After obtaining all the inconsistency thresholds and the corresponding features, we use two internal validity indices, *Dunn index* [10] and *Calinski Harabaz index* [6], to evaluate the goodness of features (i.e., clustering structure). For a given set of features extracted by the clustering algorithm, a higher Dunn and Calinski Harabaz index indicates better feature extraction. In our case, Dunn Index is regarded as main internal index, while Calinski Harabaz index is an auxiliary index. We only employ this index if the Dunn index values for two or more features are equal (though inconsistency thresholds are different) to estimate the goodness of features with same Dunn index value, and thus, to select the appropriate inconsistency threshold. If the Calinski Harabaz index makes no difference, we use the mean value of inconsistency thresholds resulting in the same internal validity index as the final inconsistency threshold.

As an example, we explain the process of feature and variation points extraction in Figure 2 using four requirements (R1–R4) and three variants (V1–V3). The similarity matrix of requirements is firstly fed into HAC to cluster similar requirements. After the optimal inconsistency threshold is defined (i.e., IT=1), we can achieve (1) an initial feature tree containing three concrete features (i.e., Feature A, C, D) and an abstract feature (i.e., Feature B); and (2) a

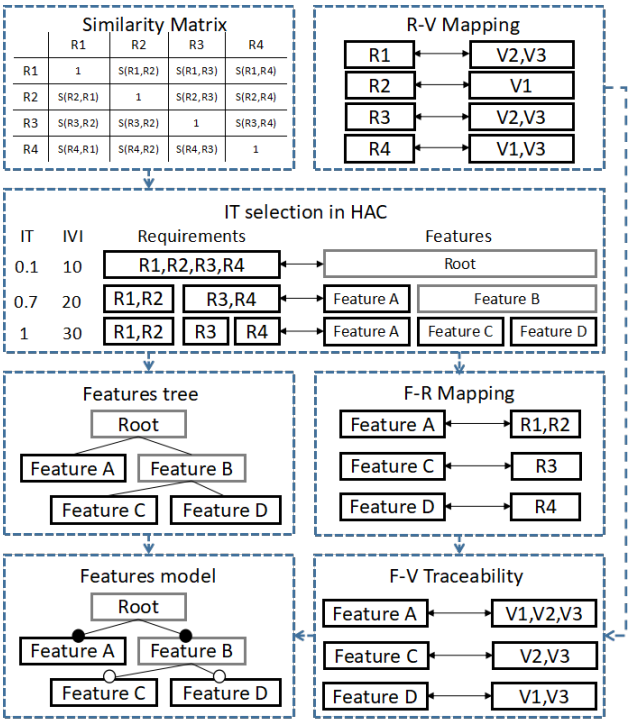

Figure 2: An example of variability information extraction

mapping between concrete features and requirements *(i.e., F-R Mapping)*. The concrete features are directly extracted from functional requirements, while the abstract feature is detected by inheriting the hierarchy from HAC. Moreover, the input requirement documents we used contain the mapping between requirements and variants *(i.e., R-V Mapping)*. Hence, the traceability link between features and variants *(i.e., F-V Traceability)* which is used to detect variation points (cf. Section 4.2) can be inferred from the aforementioned two mappings.

4.2 Variation Point Detection

Detecting variation points between features is a non-trivial task without having in-depth domain knowledge. Thus, we rely on heuristics to decide whether and how a feature is related to other features. To this end, we make use of the *F-V Traceability* link (cf. Figure 2) between features and variants and define the following four criteria to extract variation points for features:

(1) A feature is *Mandatory* if (a) this feature covers all the variants from input requirement documents; or (b) its sub-features covers all the variants.
(2) Consequently, a feature that has *not* been identified as mandatory feature is *Optional*.
(3) Features under a same parent node form an *OR* group, if there are at least $(n-1)$ pairs of these features which appear in the same variant, where n is the number of features under the same parent node. Moreover, (a) every pair of features must be different and unique, (b) these $(n-1)$ pairs of features must include all the features under this same parent node, and (c) these n features are able to cover all variants from input requirement documents.
(4) Features under a same parent node form an *XOR* group, if the union of these features covers all the variants, and these features never occur in the same variant simultaneously.

Next, we illustrate the application of our criteria by means of two abstract examples, denoted in Figure 2 and Table 2. In Figure 2, feature A cover all three variants (V1–V3), thus, rendering feature A as mandatory. Feature B contains two sub-features: feature C and D. Since the union of feature C and D covers all three variants, feature B is mandatory. Obviously, Feature C and D are optional.

In Table 2, features F, G, H, I belong to the same parent feature as well as features J, K belong to the same parent node. According to our criteria, feature F, G, H, I are or-grouped; feature J, K are xor-grouped.

Table 2: Example of OR and XOR group relation extraction

Feature	V1	V2	V3
Feature F	×		×
Feature G	×	×	
Feature H		×	×
Feature I	×		×
Feature J	×		×
Feature K		×	

5 EVALUATION

To demonstrate applicability and benefits of our proposed technique, we conducted a case study. In this section, we present the research questions to be answered, subject systems, and evaluation metrics and report on the overall methodology of our evaluation. Moreover, we present results of a comparison with other, related approaches. All data of our evaluation are available online[1].

5.1 Research Questions

For evaluating the quality of our feature and variability extraction process, we formulate the following three research questions:

RQ1: How accurate is the extracted feature and variability information? For this research questions, we aim at determining the accuracy of the identified features and the variation points in order to assess whether our approach provides relatively accurate results.

RQ2: Does our proposed technique achieve a high degree of automation? For this research question, we put emphasis on how much our technique can prevent manual intervention in the process of identifying features and extracting variability information.

RQ3: Does the proposed approach work properly for different domains? For this research question, we put emphasis on the applicability of our approach to different domains. If the approach would be sensitive for a certain domain, it lacks generalizeability for different domains in practice, and thus, would have a limited applicability.

In order to answer these three research questions, we conduct an experiment with a case study and a comparison with two related approaches in different domains.

[1]https://git.iti.cs.ovgu.de/yangli/PSC_release

5.2 BCS Case Study

In this section, we provide details about the subject system and the general process for conducting our case study.

Dataset. We obtained the requirements (i.e., software requirement specifications) from the *Body Comfort System (BCS)* [22], a case study from a real-world scenario that takes variability into account. Overall, BCS comprises 98 concrete functional requirements, in which each requirement is short text with one or two sentences specifying the functionality of the system. And, it provides the mapping between requirements and variants. Moreover, BCS comes with a feature model, shown in Figure 4. The model has been manually created by domain engineers and encompasses 26 features. Hence, we can make use of this model as a ground truth to evaluate the accuracy of our approach.

Experimental setting. For word level similarity, a word embedding model trained with large corpus is supposed to be of high quality. Hence, we apply a pre-trained model trained on Google News dataset (https://code.google.com/archive/p/word2vec/) including 100 billion words. For requirement level similarity, according to parameter setting in [24], we denote $k1 = 1.2$, $b = 0.75$ and $\delta = 1$ when computing Equation 2 in our experiments. The preset weight value for topic words is 1. For HAC, we use *euclidean metric* to measure the distance between pairs of requirements, while *ward linkage criterion* is applied to determine the distance between sets of requirements. The pre-defined step for inconsistent threshold selection is 0.01. Moreover, our implementation depends on gensim [31], SciPy [17] and scikit-learn [29].

Process of feature model extraction. First of all, we extracted 42 topic words from the original short text requirements (before pre-processing the raw requirements) of BCS by applying Stanford Parser, since Stanford Parser need complete information of a sentence. Then, the raw requirements were pre-processed by removing stop words, implementing lemmatization and lowercasing letters.

Subsequently, the pre-processed requirements were fed into the semantic similarity network. By using Equation 3, the 98 requirements were transformed into a similarity matrix (a 98×98 symmetric matrix) consisting of the similarity value for each pair of requirements.

Finally, the requirements were further processed to extract features and variability information, obeying the following steps: First, we fed this 98×98 similarity matrix into HAC to cluster features. Based on Dunn and Calinski Harabaz index, the appropriate inconsistency threshold for these requirements is computed (for BCS, it is 1.10). Then, we obtained 23 clusters of requirements which we interpret as 23 concrete features in terms of the previously computed inconsistency threshold. Moreover, according to the initial tree structure by HAC, we also obtained 4 abstract features. Second, on the basis of our four proposed criteria, the variation points and relationships between features were extracted. Among the 27 extracted features, we found eight features to be *mandatory*, while 19 of them have been identified to be *optional* features. Furthermore, neither or-grouped nor xor-grouped features have been identified from the requirements.

After gaining the features and variability information by our proposed technique, we employ *FeatureIDE* [18] as graphical and textual editor to visualize the extracted feature model, which we show in Figure 3.

5.3 Evaluation Metrics and Results

In order to achieve a comprehensive evaluation of our technique, we have to consider two processes in our technique to be evaluated based on the ground truth of BCS:

- First, we implement *clustering evaluation*, which is used to estimate whether a reasonable clustering is selected based on how well the clusters produced by HAC match the ground truth. In detail, this evaluation step is to measure whether the similar requirements are grouped in correct features and the dissimilar requirements are separated well enough.
- Second, we implement *feature model evaluation* to measure how accurate we extracted the features and variability information. To this end, we measure how well our the extracted feature model complies with the original BCS feature model.

Clustering evaluation. We use two external validation techniques to estimate the clustering performance: *average accuracy (AA)* [23] and *normalized mutual information (NMI)* [36]. By using two different validity techniques, we prevent our evaluation form being biased when evaluating the accuracy of clusters.

Evaluating clustering performance with AA relies on how many documents with the same topics are in the same cluster and also how many documents with different topics are in different clusters, which actually is a mean value of positive and negative accuracy. Based on value of AA, we can obtain general understanding regarding the accuracy of feature clusters. NMI [36] is an information theoretic measure to scale the mutual information of requirements between clusters. Based on value of NMI, we know about how related two requirements are and whether the requirements are well separated or grouped, respectively. The larger the values of AA and NMI are, the better the clustering performance of our approach is.

Table 3: Results of Clustering evaluation

	AA	NMI
Clustering performance	0.79	0.84

We evaluate the clustering results (i.e., extracted concrete features) of our approach by comparing them with the ground truth. In Table 3, we show results for AA and NMI metrics. Both are around 0.8 which is close to 1, and thus, indicate a relatively accurate clustering result. Moreover, the result reveals that the majority of the requirements is assigned to the correct features, which matches the ground truth accurately.

Feature model evaluation. Although, we obtain a relatively accurate matching between our predicted clusters and the ground truth, the results of clustering evaluation can not completely reveal the rationality and applicability of the extracted features, considering that manually extracting feature and variability may pose the risk of uncertainty and bias due to differences between the results from two (or more) domain engineers. Hence, in order to take this kind of uncertainty and bias into account, we make a mapping (cf. Table4) between our extracted features and the corresponding features in the ground truth to evaluate the extracted feature model.

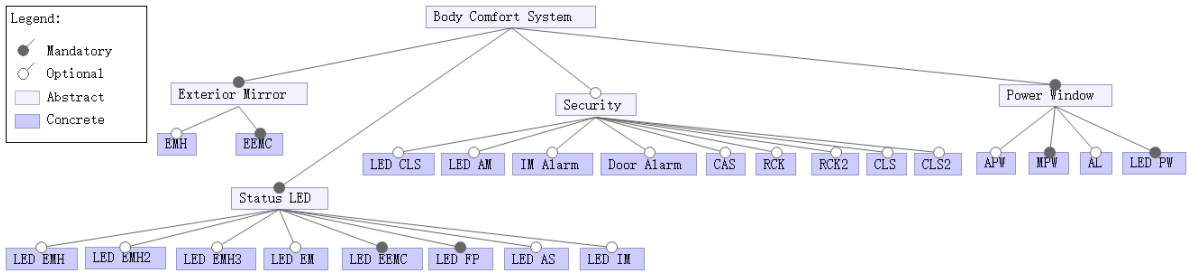

Figure 3: The extracted BCS feature model

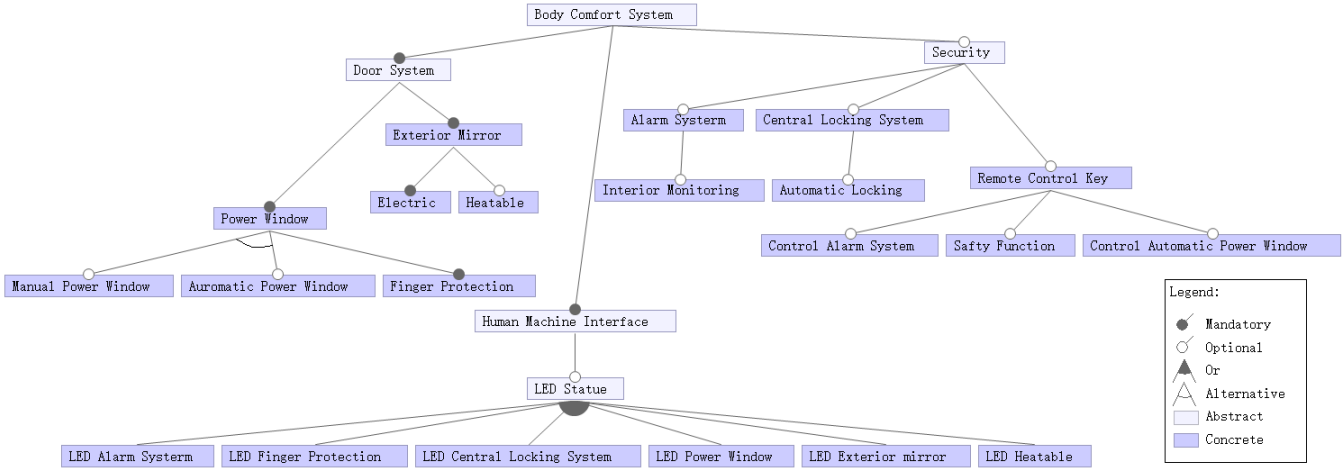

Figure 4: The ground truth of BCS feature model

Moreover, we utilize the common measures *precision*, *recall*, and *F-measure* to evaluate the accuracy of the extracted features. In the context of feature extraction in SPLs, precision is used to determine how many features have been extracted correctly. Consequently, recall is employed to measure the proportion of correctly identified and extracted features compared with all features in truth sets. Finally, to put precision and recall in relation, we use F-measure as a harmonic average of both of these measures. To this end, we denote 1) the true features extracted by our approach as *true positives*, 2) features generated by our approach but absent in the ground truth set as *false positives*, and 3) the pertinent features (in truth set) which are absent in the feature models generated by our process as *false negatives*.

Given this definition of evaluation metrics, we take two situations into consideration:

(i) We regard the feature model generated by domain engineers as the only ground truth, not considering the possible bias of manual analysis.
(ii) We analyze the extracted features, taking the uncertainty and bias of manual analysis into account. If the extracted features are reasonable and applicable, we also regard them as true features.

According to Table 4, the number of features belonging to true positive, false positive and false negative is 19, 8 and 5, respectively in situation (i), where the features, LED EMH2, LED EMH3, LED IM, LED AM, IM Alarm, Door Alarm, RCK2, CLS2, belong to false positives. In situation (ii), we analyzed these 8 false positive features and found that LED IM, LED AM, IM Alarm, and Door Alarm are actually applicable, since the functionality regarding LED is well separated from other functions. Hence, we also regard LED IM, LED AM, IM Alarm, and Door Alarm as true features belonging to true positives. As a result, in situation (ii), the number of features belonging to true positive, false positive and false negative is 23, 4 and 5, respectively. Table 5 illustrates that the extracted features by our technique are relatively accurate, especially considering the bias of manual analysis in situation (ii).

For variability information, we can not extract all the relationships between features compared with ground truth. For example, we missed an xor-group relation, since our approach assigns Finger Protection into Manual Power Window inappropriately. Moreover, we lost an or-grouped relation, since two mandatory features (LED EEMC and LED FP) were extracted, which, however, is reasonable. Finally, cross-tree constraints are missing in our extracted feature model, since they are currently not captured by our criteria (cf. Section 4.2). Nevertheless, the extracted variability information is relatively accurate, as indicated by Figure 3.

The difference of the tree structures between the two models (cf. Figure 3 and 4) is caused by different standards for creating the feature tree. In our approach, we inherit the hierarchical structure from HAC based on the similarity between each pair of requirements, while the ground truth is generated by manual analysis

Table 4: Comparison of extracted features and ground truth

Extracted Feature	Corresponding Feature in Ground Truth
Exterior Mirror	Exterior Mirror
EMH	Exterior Mirror Heatable
EEMC	Electric Exterior Mirror Controls
Status LED	Status LED
LED EMH	LED Exterior Mirror Heatable
LED EMH2	LED Exterior Mirror Heatable
LED EMH3	LED Exterior Mirror Heatable
LED EM	LED Exterior Mirror
LED EMMC	Electric Exterior Mirror Controls
LED FP	LED Finger Protection
LED AS	LED Alarm System
LED IM	Interior Monitoring
Security	Security
LED CLS	LED Control Locking System
LED AM	LED Alarm System, Alarm System
IM Alarm	Interior Monitoring
Door Alarm	Alarm System
CAS	Control Alarm System
RCK	Remote Control Key
RCK2	Remote Control Key
CLS	Central Locking System
CLS2	Central Locking System
Power Window	Power Window
APM	Automatic Power Window, Control Automatic Power Window
MPW	Manual Power Window, Finger Protection
AL	Automatic Locking, Safety Function
LPW	LED Power Window

considering affiliations between features. Although our feature tree structure doesn't match the ground truth perfectly, it also provides a usable hierarchical feature structure.

Table 5: Evaluation results of situation (i) and (ii)

Situation	Precision	Recall	F-measure
(i)	0.70	0.79	0.74
(ii)	0.85	0.82	0.83

5.4 Comparison with Related Approach

In this subsection, we make a comparison between our approach and the approaches described in [14]. Itzik et al. proposed an approach named SOVA [14] and two feature models for a structural perspective profile and a functional perspective profile were generated respectively by SOVA, compared with the feature model created using ArborCraft Tool [39]. In order to make a convincing comparison and verify whether our approach can process requirements in different domains, we use the same requirements of E-shop from paper [14] which is small dataset comprised of 22 requirements to extracted feature models by our approach, while we apply the same experiment setting described in Section 5.2 and the inconsistent threshold (0.83) is also generated automatically by applying internal validity indices. Then, we implement qualitative analysis, since the ground truth is not provided in paper [14].

Figure 5 presents the extracted feature model by our approach, which contains 12 features (3 abstract features and 9 concrete feature). Compared with feaure models generated by SOVA and ArborCraft, we found that every single requirement was regarded as a concrete feature in terms of the features from SOVA and ArborCraft. In contrast, our approach groups requirements with similar functionality into a same feature.

In order to make a detailed comparison, we made a mapping of features (cf. Table 6) produced by our approach and the other two approaches based on our concrete features, since concrete features which are directly extracted from requirements represents real functionality and applicability. Compared with the features from SOVA (structural perspective), two pairs of features matches perfectly with each other (Product Reviews and Update Inventory). In terms of features from SOVA (structural perspective), five pairs of features fit perfectly with each other. And we also have four pairs of features match accurately with each other based on the features of ArborCraft. Except for these perfectly matching features, other features also are capable of presenting the specific functionalities of E-shop. In detail, Payment feature consists of all the payment method by using credit card, gift card or PayPal. Update Inventory feature presents customers' behaviors lead to updating inventory. Available Products feature shows system's operation which is caused by customers' buying behavior, when available products are displayed. What's more, the three abstract features present their sub-features relatively accurately.

For variability information, based on the four criteria we proposed, we found 11 mandatory features and 1 optional feature, while neither of or-grouped and xor-grouped features has been detected. Although our approach can not extracted all the variability information successfully, the extracted relations between features are highly accurate. We argue that the feature model generated by our approach is more applicable than the feature model by ArborCraft, since the hierarchical structure is more explicit. The key difference between our approach and the compared approaches is that we avoid any manual intervention. If manual analysis is also introduced into our approach, we argue that the accuracy and completeness of variability information of our proposed approach would be improved.

5.5 Answering RQs

For RQ1, firstly, in terms of quantitative analysis of BCS case study, the approach provides relatively high precision and recall of extracted features, especially taking situation (ii) into account. For variability information, although we fail to detect all the relations between features, the extracted variability information is reasonable and applicable. Secondly, through the comparison with two related approaches, our approach also provide an accurate result of applicable feature model.

For RQ2, our approach is used for, both BCS case study and the comparison with other tools. In spite of extracting features by using requirements in different domains, we utilize the same experiment

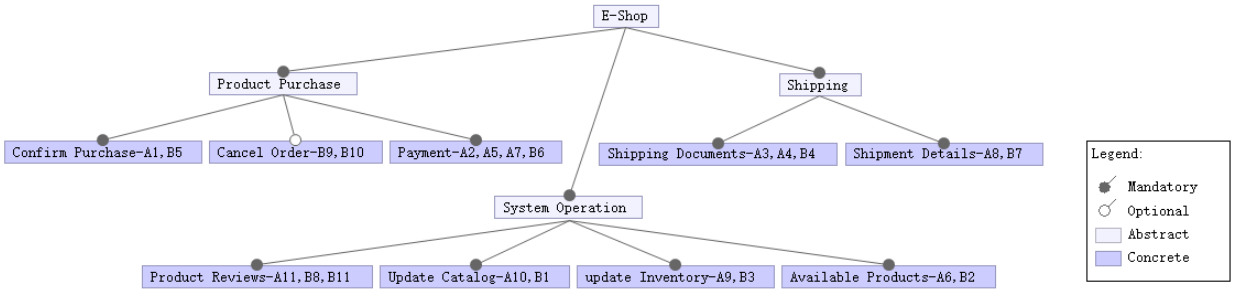

Figure 5: The extracted E-Shop feature model

Table 6: Comparison between extracted features and related approaches

Extracted Feature	SOVA (structural perspective)	SOVA (functional perspective)	ArborCraft
Confirm Purchase	ShipmentDetails-A1, ShippingCart-B5	ConfirmPurchase	A1, B5
Cancel Order	Payment-B9, Payment-B10	CancelOrder	feature27
Payment	Payment-A2, AirMailShip-A5, LandShipping-A7, PaymentInformation-B6	PayWithCreditCard-A2, BuySmallProduct-A5, PayWithGiftCard-A7, PayWithPayPal-B6	A2, A5, A7, B6
Shipping Documents	ShippingDocuments-A3, ShippingDocuments-A4, ProductDeliveryStatus-B4	ShipOrderedProduct	feature26
Shipment Details	ProductDeliveryStatus-A8, ShipmentDetails-B7	TrackStatus	A8, B7
Product Reviews	ProductReview	ReviewProduct, ReturnDamagedProduct-B8	A11,B8,B11
Update Catalog	Catalog	EnterNewProduct	feature22
Update Inventory	Inventory-A9, Inventory-B3	ReturnProduct-A9, PurchaseProduct-B3	feature24
Available Products	Inventory-A6, ShippingCart-B2	BuyProduct-A6,PurchaseProduct-B2	A6,B2

and parameter setting. In the process of feature identification and variability information extraction, we successfully avoid manual intervention not only in BCS case study, but also in the comparison with two related researches. What's more, our approach of full automation also achieves feature model generation of relatively high quality.

For RQ3, comparing the results of two different SPLs, we evaluate whether our approach works properly for different domains. We first conduct a BCS case study with both quantitative and qualitative analysis which reveals the relatively accurate result of our approach. Second, although there is no ground truth in SPL of E-shop, the feature model generated by our approach is of applicability based on the comparison of results between other two tools. Taking both qualitative and quantitative analysis of the results in two SPLs into account, our approach is of relative universality for different SPLs. Although we obtain both accurate results in two different SPLs, there are still many challenges needed to be overcome to make the generated feature model more reasonable.

6 THREATS TO VALIDITY

Construct Validity: We apply semantic similarity of requirements to extract feature by clustering algorithm, regarding requirements with similar functionality as a feature. Although this method may lead to bias of identifying features, we still achieve relatively accurate features compared with ground truth and feature models produced by other tools.

Internal Validity: Firstly, the weight value for topic words used in requirements level similarity are predefined values. If the topic words and the corresponding weighting values are not appropriate, it will bring bad impact on the result. Hence, we evaluate our approach in two domains, BCS and E-shop. Based on evaluations, most results are relatively precise. Secondly, We define four criteria to extract variability information, which lacks more domain knowledge support and may not be satisfied with all situations.

External Validity: Many texts in requirements may not be related to features at all, for example non-functional requirements, organizational constraints, the objective of the software, etc, while some texts are even not related to the software at all (e.g., stakeholders, problem statements, potential customers). Hence, in these cases, some manual semantic analyses may still be needed to gain the final features. Our approach is capable of extracting features from short text requirements. However, if a requirement describing a functionality is of too many sentences, our approach maybe can not provide an accurate result.

Conclusion Validity: In order to verify the applicability of our approach for different domains, we apply our approach in two SPLs. However, the results from just two domains may be not enough to support the conclusion.

7 RELATED WORK

In the context of feature extraction from requirements in SPLs, some approaches highly depend on syntax and external knowledge database. Reinhartz-Berger et al. proposed an approach to extract features and variability information based on combining semantic text similarity of requirements with similarity of behavioral aspects

of requirement statements [32]. They applied Semantic Role Labeling (SRL) techniques that highly relies on syntactic information to extract different roles which are of special importance to functional requirements and then classify these roles in to the initial state, external events, and final state of software behavior. However, both SRL technique and software behaviors highly relies on accurate syntactic information, which leads to manual intervention.

Itzik et al. proposed an ontological approach that calculates the semantic similarity, extracts features and variability, and generates feature models that represent structural (objects-related) or functional (actions-related) perspectives [14]. They also used SRL to transform each sentence into six roles and computed similarity of each pair of semantic roles by applying WordNet [14], focusing on objects (or components) and actions (or functions), respectively. Afterwards, HAC is applied to these roles with different weights to cluster features. However, they manually define a fixed distance threshold to extract features in HAC, which lacks applicability for different domains. In contrast, we utilize internal validity indices to select the inconsistent values automatically for different datasets. Except for the limitation of SRL and HAC they used, WordNet as an external knowledge database is not capable of containing the all words meanings of high quality.

Based on the papers above [14, 32], Itzik et al. proposed an approach named semantic and ontological variability analysis (SOVA) to analyze variability of functional requirements [15]. This approach uses ontological and semantic considerations to automatically analyze differences between initial states, external events, and final states of behaviors, and thus, identify features, considering different perspectives which reflect stakeholders' needs and preferences [15]. Hence, SRL technique is also used to extract semantic roles, which causes manual analysis. They apply two different ways, LSA and MCS technique by Mihalcea, Corley, and Strapparava, to compute similarity. Compared word2vec model we used, LSA is a traditional DSMs with lower accuracy. And MCS is weighted calculation of words regardless of structure information of requirements, for example the requirement length.

Wang proposed a method to build semantic frames for frequent verbs appearing in requirements by applying SRL with the assistance of Stanford Parser and WordNet [37, 38]. However, Wang's research only extracted semantic information of requirements and didn't extract features in the context of SPLs. Although we also used Stanford Parser to obtain the topic words, we don't need any manual analysis for the results from Stanford Parser.

Another approaches focus on traditional DSMs techniques. Alves et al. conducted an exploratory study on leveraging information retrieval techniques for feature and variability extraction [1]. They presented a framework by employing LSA and VSM technique to measure the similarity between sentences and also applied HAC to cluster features based on this similarity. Weston et al. proposed a tool suite for processing requirements into candidate feature models, which can be refined by domain engineers [39]. They applied LSA to measure similarity and HAC to cluster similar texts into same feature groups to create a feature tree. Moreover, they built a variability lexicon and grammatical patterns to detect latent variability information. Tsuchiya et al. proposed an approach to recommend traceability links between sentences in requirements and design-level UML class diagrams as structure models [20]. They used VSM to determine similarity between sentences in requirements. However, their research direction is not to discover features based on the similarity. In contrast to the research above, we utilize word embedding instead of using traditional DSMs to gain word vector representation of the requirements, which contains more semantic information.

There are also related works focusing on extracting features by analyzing the domain-specific terms [12, 28]. However, in Ferrari et al.'s research, they only detected mandatory and optional features, while Nasr et al. aimed at extracting features from informal product description to synthesize a product comparison matrix rather than detecting the variation points. Moreover, Fantechi et al. explored the feasibility of extracting variability from the ambiguity defects mainly based on occurrence of the word with different meanings in requirements [11].

8 CONCLUSION

Natural language requirement documents contain original and complete information of products, especially for software requirements specifications as the primary artifacts in the development of software products. Meanwhile, feature and the variation points extraction from requirements documents can provide a explicit mapping of feature and variants to other artifacts.

In this paper, we proposed an approach to identify features and extract variability information from software requirements specifications. In order to improve the accuracy of feature extraction, semantic information of both the words and requirements is analyzed and used for compute the similarity. Moreover, we take topic words regarding the requirements of a domain into account, while topic words can be seen as the smallest domain knowledge. We use this knowledge to assign weight values to our similarity measures, and thus, improve the accuracy. We then apply hierarchical clustering and make use of predefined criteria to identify features and their variation points and, finally, to construct the feature model. To evaluate accuracy of our approach, we conduct a case study from a real-world scenario and evaluate the results qualitatively and quantitatively. Meanwhile, we make a comparison with other two related approaches in a different domain. Our results reveal that we gain relatively accurate features fully automatically. Although our approach is not capable of detecting all the variability information, the majority of the extracted relationships between features is reasonable.

As future work, we will focus on improving the accuracy and completeness of relationships detection between features. Because of the complexity of variability information, it's impossible to extract all the relationships between features very accurately only by semantic similarity comparison and four criteria. Moreover, in this paper, we didn't analyze the cross-tree constraints between features. Hence, we plan to integrate other techniques and more domain knowledge to enhance the relationship detection, especially for cross-tree constraints.

REFERENCES

[1] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. 2008. An Exploratory Study of Information Retrieval Techniques in Domain Analysis. In *Proc. Int'l Software Product Line Conference*. IEEE, 67–76.

[2] S. Apel, D. Batory, C. Kästner, and G. Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

[3] M. Baroni, G. Dinu, and G. Kruszewski. 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proc. Conf. Association for Computational Linguistics (ACL)*. ACL, 238–247.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Proc. Int'l Conf. Advanced Information Systems Engineering*. Springer-Verlag, 491–503.

[5] S. Bird, E. Loper, and E. Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

[6] J. Calinski, T.and Harabasz. 1974. A Dendrite Method for Cluster Analysis. *J. Communications in Statistics - Theory and Methods* (1974).

[7] P. C. Clements and L. M. Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional.

[8] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature Location in Source Code: A Taxonomy and Survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.

[9] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *Proc. Eur. Conf. Soft. Maint. and Reeng.* IEEE, 25–34.

[10] J. C. Dunn. 1974. Well-Separated Clusters and Optimal Fuzzy Partitions. *J. Cybernetics* (1974).

[11] A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. 2018. Hacking an Ambiguity Detection Tool to Extract Variation Points: an Experience Report. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems*. 43–50.

[12] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta. 2013. Mining Commonalities and Variabilities from Natural Language Documents. In *Proc. Int'l Software Product Line Conference*. ACM, 116.

[13] M. Hamza and R. J. Walker. 2015. Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines. In *Proc. Int'l Workshop Realizing Artificial Intelligence Synergies in Soft. Eng.* IEEE, 25–31.

[14] N. Itzik and I. Reinhartz-Berger. 2014. Generating Feature Models from Requirements: Structural vs. Functional Perspectives. In *Proc. Int'l Software Product Line Conference*. ACM, 44–51.

[15] N. Itzik, I. Reinhartz-Berger, and Y. Wand. 2016. Variability Analysis of Requirements: Considering Behavioral Differences and Reflecting Stakeholders' Perspectives. *IEEE Trans. Soft. Eng.* 42 (2016), 687–706.

[16] A. K. Jain and R. C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice-Hall, Inc.

[17] E. Jones, T. Oliphant, P. Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. (2001–). http://www.scipy.org/

[18] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. 2009. FeatureIDE: A Tool Framework for Feature-Oriented Software Development. In *Proc. Int'l Conf. Software Engineering*. IEEE, 611–614.

[19] Y. Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proc. Int'l Conf. Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1746âĂŞ1751.

[20] K. Kumaki, R. Tsuchiya, H. Washizaki, and Y. Fukazawa. 2012. Supporting Commonality and Variability Analysis of Requirements and Structural Models. In *Proc. Int'l Software Product Line Conference*. ACM, 115–118.

[21] Y. Li, S. Schulze, and G. Saake. 2017. Reverse Engineering Variability from Natural Language Documents: A Systematic Literature Review. In *Proc. Int'l Conf. Systems and Software Product Line Conference*. ACM, 133–142.

[22] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. 2013. *Delta-oriented Software Product Line Test Models - The Body Comfort System Case Study*. Technical Report. TU Braunschweig.

[23] L. Liu, J. Kang, J. Yu, and Z. Wang. 2005. A Comparative Study on Unsupervised Feature Selection Methods for Text Clustering. In *Proc. Int'l Conf. Natural Language Processing and Knowledge Engineering*. 597–601.

[24] Y. Lv and C. Zhai. 2011. Lower-bounding Term Frequency Normalization. In *Proc. Int'l Conf. Information and Knowledge Management*. ACM, 7–16.

[25] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, Steven J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proc. Conf. Association for Computational Linguistics (ACL)*. 55–60.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proc. Int'l Conf. Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 3111–3119.

[27] G. A. Miller. 1995. WordNet: A Lexical Database for English. *Comm. ACM* 38 (Nov 1995), 39–41.

[28] S. B. Nasr, G. Bécan, M. Acher, J. B. Ferreira Filho, N. Sannier, B. Baudry, and J. Davril. 2017. Automated Extraction of Product Comparison Matrices from Informal Product Descriptions. *J. Sys. and Soft.* 124 (2017), 82 –103.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Machine Learning Research* 12 (2011), 2825–2830.

[30] K. Pohl, G. Böckle, and F. van Der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer.

[31] R. Řehůřek and P. Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. 45–50.

[32] I. Reinhartz-Berger, N. Itzik, and Y. Wand. 2014. Analyzing Variability of Software Product Lines Using Semantic and Ontological Considerations. In *Proc. Int'l Conf. Advanced Information Systems Engineering*. Springer, 150–164.

[33] C. Riva and C. Del Rosso. 2003. Experiences with Software Product Family Evolution. In *Proc. Int'l Workshop on Principles of Software Evolution*. IEEE, 161–169.

[34] S. Robertson and H. Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* (APR 2009).

[35] L. Rokach and O. Maimon. 2005. *Clustering Methods*. Springer US, 321–352.

[36] A. Strehl and J. Ghosh. 2003. Cluster Ensembles - a Knowledge Reuse Framework for Combining Multiple Partitions. *J. Mach. Learn. Res.* (Mar 2003), 583–617.

[37] Y. Wang. 2015. Semantic Information Extraction for Software Requirements using Semantic Role Labeling. In *Proc. Int'l Conf. Progress in Informatics and Computing (PIC)*. IEEE, 332–337.

[38] Y. Wang. 2016. Automatic Semantic Analysis of Software Requirements Through Machine Learning and Ontology Approach. *J. Shanghai Jiaotong University* 21 (2016), 692–701.

[39] N. Weston, R. Chitchyan, and A. Rashid. 2009. A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements. *Proc. Int'l Software Product Line Conference* (2009), 211–220.