Automated Extraction of Domain Knowledge in Practice: The Case of Feature Extraction from Requirements at Danfoss

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

Helene Hvidegaard Scherrebeck
Thomas Sorensen Fogdal
Danfoss
Gråsten, Denmark
{helene.scherrebeck,tfogdal}@danfoss.com

ABSTRACT

Software product line supports structured reuse of software artifacts in order to realize the maintenance and evolution of the typically large number of variants, which promotes the industrialization of software development, especially for software-intensive products. However, for a legacy system, it is non-trivial to gain information about commonalities and differences of the variants. Meanwhile, software requirements specifications as the initial artifacts can be used to achieve this information to generate a domain model. But, manually analyzing these requirements is time-consuming and inefficient. To address this problem, we explored to use feature extraction techniques to automatically extract domain knowledge from requirements to assist domain engineers. In detail, we applied Doc2Vec and clustering algorithm to process the requirements for achieving the initial feature tree. Moreover, we utilized key words/phrases extraction techniques to provide key information to domain engineers for further analyzing the extraction results. In particular, we developed a GUI to support the extraction process. The empirical evaluation presents that most of the extracted features and terms are beneficial to improve the process of feature extraction.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Yang Li, Sandro Schulze, Helene Hvidegaard Scherrebeck, and Thomas Sorensen Fogdal. 2018. Automated Extraction of Domain Knowledge in Practice: The Case of Feature Extraction from Requirements at Danfoss. In *Proceedings of 24th ACM International Systems and Software Product Line Conference (SPLC'20).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

SPLC'20, 19–23 October, 2020, Montreal, Canada

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Nowadays, software is subject to *mass production*, leading to business-critical aspects such as reliability or time to market. However, developing software for the masses is only one challenge in software development today. At the same time, the demand for *customization* of software systems heavily increases [18], and thus, requires to tailor a software system according to the specific needs of a customer. Usually, this demand for customization is impossible to estimate and foresee, and thus, is accomplished in an ad hoc fashion by adding new features as needed. While this is a straightforward process that comes with almost no costs in the short term, it exhibits possible severe consequences in the long term: with an increased number of features (for different customers), the relation and dependencies between them are usually not documented, thus, giving rise to inconsistencies. Moreover, maintenance tasks may be hindered as changes can not be propagated due to missing domain knowledge. Finally, an overall domain model is absent, and thus, makes reasoning or reuse across several parts of the software system impossible.

These shortcomings can be addressed by structured reuse using *Software Product Lines (SPL)*, which allow developing a family of products that share common functionalities and characteristics satisfying the specific needs of a particular market segment [6]. In order to achieve this level of automation, *Software Product Line Engineering (SPLE)* describes a systematic development process to facilitate the product development by taking commonalities and differences (also known as variabilities) among particular products into account [7, 30]. The concept of *feature* plays a pivotal role in SPLE, since features are the basic units to describe domain characteristics, map them to implementation artifacts, and eventually allow to reason about commonalities and differences between products and stakeholders. Hence, a feature in this context is a user-visible increment in functionality, paving the way for systematically structuring, varying, and reusing software artifacts across all phases of the software life cycle [3].

While SPLE has been proven to be beneficial in practice [10, 12, 32], it takes significant efforts to introduce SPLE from scratch, as domain features and their relations must be known, requiring a detailed domain analysis [16]. Also, setting up the whole process usually implies a considerable overhead, as it must be ensured that both, domain as well as implementation artifacts are evolved together and exhibit clearly defined variation points [30]. Consequently, the decision for applying SPLE is usually postponed to a tipping point, where this overhead is considered to be beneficial in the long term [18]. However, the information needed to introduce

SPLE for a legacy system (i.e., a system that has been evolved for years) is non-trivial and requires information that is usually not explicitly documented such as domain features, the corresponding implementation artifacts or even an overall domain model, which allows comprehensively reasoning about all features.

In previous work, we (1) have shown that Software Requirements Specifications (SRS) are suitable artifacts to extract such information [21] and (2) proposed a technique that makes use of advanced Natural Language Processing (NLP) concepts to reliably extract features and variability from SRS [22]. So far, our technique has been only evaluated with a small set of rather artificial requirements, and thus, it is unclear whether it can cope with specialities of real-world requirements as well as scales up to a large amount of SRS documents.

In this paper, we address this problem by analyzing Software Requirements Specification from Danfoss, which is a Danish company with more than 28,000 employees worldwide. The company does businesses within power solutions, cooling, heating and drives, of which the latter has existed for more than half a century. In order to automate the process of domain analysis, we utilize machine learning and NLP based techniques to process and analyze the requirements. Basically, we rely on our previously introduced approach [22], that is, we apply neural word embedding techniques to achieve the similarity of requirements and a clustering algorithm to extract features by grouping requirements with related functionality. However, according to the specificity of Danfoss' case, most requirements are structured well with a proper name. Moreover, the structure of the requirements documents and the names of the requirements can be regarded as a kind of expert knowledge derived from the experience in practice. Hence, in this paper, we propose a revised approach that takes into account not only the similarity of the body of requirements but also the information about the structure and names of the requirements. Moreover, we substitute Doc2Vec for Word2Vec to achieve not only the vector representation of each word in the requirements but also the vector representation of each requirement. Additionally, we utilize our recently proposed the technique [23] to identify the feature-related terms that can benefit understanding the notions of the extracted features. Finally, we present a GUI that integrates the above-mentioned methods to support domain engineers to visualize and adjust the extracted features in practice. In summary, we make the following contributions:

- We present the first study that applies automated (domain) feature extraction and feature tree creation on a large amount of real-world requirements.
- We present and discuss peculiarities in real-world requirements that may be challenging by such automated approaches. These insights can be used by others for future research on analyzing natural language requirements.
- We propose a refined technique (compared to previous work [22]) combined with the technique in research [23] to form a complete approach for automated feature extraction that addresses the identified specialities of real-world SRS.
- We provide an empirical evaluation and a qualitative analysis of our technique to discuss its applicability in practice.

2 RELATED WORK

In this section, we discuss prior research in feature extraction from three different perspectives: 1) the different Distributional Semantic Models (DSMs) used for achieving similarity; 2) the application and effect of requirement parsing; 3) the analysis of different types of textual documents for feature extraction.

2.1 Traditional DSMs

There exist some previous research focusing on applying traditional DSMs to achieve the similarity of the requirements. Alves et al. utilized Latent Semantic Analysis (LSA) and Vector Space Model (VSM) respectively to compute the similarity of each pair of the requirements, and then, applied Hierarchical Agglomerative Clustering (HAC) to achieve the initial feature tree in terms of the similarity of the requirements [2]. Weston et al. proposed a tool named ArborCraft to identify features from requirements also based on applying LSA and HAC [35]. Kumaki et al. applied VSM to measure the similarity of each pair of sentences in requirements and also calculate the similarity between classes of design-level UML class diagrams to support the analysis of commonality and variability [19].

By contrast, we use neural network based technique, Doc2Vec that is an extension of word2vec [26], to achieve the vector representation of requirement rather than applying traditional DSMs. Doc2Vec can be regarded as a prediction model, while the traditional DSMs are the count models [5]. In terms of Baroni et al.'s research, prediction models have been proven to outperform common count models [5]. Although we also use HAC to extract the initial feature tree, we apply a different metric to gain clusters and simplify the hierarchical clustering tree (Section 3.2).

2.2 Requirement Parsing

Besides directly applying traditional DSMs to gain similarity, some approaches firstly parse the requirement in terms of different rules. The purpose of parsing the requirements is to extract the potential semantic roles or behaviors with respect to functionality from requirements. And then, features are identified further based on the semantic roles or behaviors information. In research [33, 34], the semantic frames of frequent verbs in requirements are built in order to extract the structured semantic information. The concept of semantic frames comes from Semantic Role Labeling (SRL), while authors applied Stanford Parser [25] and WordNet [27] to assist the process of analysis. Although the semantic information was identified, the authors didn't propose a complete approach to use semantic frames to extract features in the SPL context. Itzik and Reinhartz-Berger analyzed the requirements based on SRL and parsed each sentence in the requirements in terms of different semantic roles they predefined [13]. And then, the similarity of requirements is computed based on the similarity of each pair of semantic roles by utilizing WordNet. After that, HAC is also used to extract features. In research [14], authors extended their prior researches [13, 31] to apply the ontological and semantic considerations to analyze requirements based on the behavior-oriented extraction method. In detail, SRL is also applied to extract semantic roles from requirements, and HAC is used to identify features in terms of the similarity computed by applying LSA.

Even though parsing the requirements improves the process of feature extraction, it takes extra efforts to parse the requirements by manual analysis even with the assistance of NLP tools, such as Stanford Parser. This is mainly because 1) requirement parsing relies on accurate syntactic information of sentences that needs to be checked by manual analysis and 2) the parsing task usually follows several particular pre-defined rules which need to be mastered by engineers. Both of them increase the cost for domain engineers to have the usable parsed requirements.

2.3 Miscellaneous Textual Documents

Except for requirements specifications, some researches are focused on extracting features from other types of textual documents, for example, informal product descriptions [1, 8, 9, 29] or online software reviews [4, 28]. They also used different techniques, such as K-means [28], Fuzzy C-Means [4], Association Rule Mining [8], to aid feature extraction. However, informal product descriptions and online software reviews just contain a very small part of information regarding features, compared with requirements specifications. Hence, the features extracted from these informal textual documents are really limited. By contrast, we analyze the requirements specifications that contain complete information regarding functionality to extract features.

The majority of the aforementioned research was conducted on small datasets each of which is less than 100 individual requirements to be used to explore the methods to automate the process of feature extraction from requirements. However, feature extraction in practice may face a large dataset of requirements and automated extraction is impossible to provide a result with 100% accuracy. Hence, manual analysis is indispensable for finally correct the feature extraction results in practice. In contrast with previous researches, we provide a practical framework that not only can produce the recommended features from real-world requirements of relatively large size, but also offers a GUI that is able to visualize all the extracted features and restructure them based on some key information.

3 METHODOLOGY

The overall goal of our technique is to extract feature information from requirements (i.e., which requirements belong to which domain feature), put these features into relation by means of a domain model (here: a feature tree), and to provide semantic information what a feature is about (i.e., which functionality it encompasses). In Figure 1, we show an overview of our proposed approach. First of all, we extract the initial feature tree mainly based on Doc2Vec and Hierarchical Agglomerative Clustering (HAC) combined with the information about the structure and names of requirements (cf. Section 3.1 and Section 3.2). Second, we identify feature terms to provide key information of a feature by using a prediction model that analyzes different attributes of words and phrases in the requirements (cf. Section 3.3). Finally, we propose an GUI that presents the initial feature tree and extracted feature terms to domain engineers, and thus, allows reviewing and revising the extracted feature information.

Note: since the requirements from Danfoss are confidential, in this section we use requirements from Body Comfort System [24] and Digital Home [11] as examples to illustrate our approach.

3.1 Preprocessing

The indispensable step to initialize automated feature extraction is to preprocess the requirements in order to satisfy the demands of different NLP-based sub-tasks.

Text extraction. Usually, there is no uniform format for requirements of describing different functionalities. Taking the development of a large number of product variants into account, diverse requirements with multiple types of data and formats are written to meet different customers. Although SRS contain various types of data, such as texts, tables, and figures, the textual information is the main medium to convey the concrete specifications between customers and suppliers on how the products should function. In order to obtain all the natural language texts from SRS, we initially process the requirements and extract any textual information by removing the non-textual data such as figures. Moreover, textual information should be preserved as much as possible, not only from the plain texts but also from tables that contain plenty of textual data that describe the functionalities. Hence, texts in tables are also extracted in order to achieve more information regarding feature extraction.

After achieving the pure textual requirements, further techniques will be used to process the requirements. For example, we apply *tokenization* by which the requirements are decomposed into a list of elements such as words, punctuation, and symbols, called *tokens*. However, the most important step for preprocessing is to reduce the complexity of the requirements for some specific NLP tasks. To this end, we apply stop words removal and lemmatization:

Stop words removal. Stop words are the most commonly used words in a language, such as "the", "this", "that", etc. In our approach, these stop words are removed, since they do not contain enough semantic information. Moreover, we also find that there are some non-functional terms in the requirements specification, such as some specific titles. The high frequency of the occurrences of these non-functional terms has a bad effect on the further process of grouping requirements with similar functionality. Hence, we add these non-functional terms into a blacklist to remove them.

Lemmatization. Lemmatizing a word is to convert the infected words into their dictionary forms. Usually, lemmatization is used to solve the sparse data problem. In our approach, since the infected words increase the complexity and unexpected noises for further similarity calculation of the requirements (cf. Section 3.2), we also apply lemmatization to process all the words in the requirements.

3.2 Feature Extraction

For feature extraction, we compute the similarity of requirements including the similarity of the structure, name, and body of the requirements. Note that in the following sections, we use the terms *requirement structure, requirement name, and requirement body* to denote the structural information of a requirement, the name of a requirement, and the body of a requirement, respectively. Eventually, we use these similarities to create a feature tree using clustering.

Vectors of requirement bodies. Doc2Vec is a neural network based, but unsupervised learning algorithm to generate a vector space of documents [20]. Thus, by using Doc2Vec, each document can be converted into a vector representation. One of the advantages of

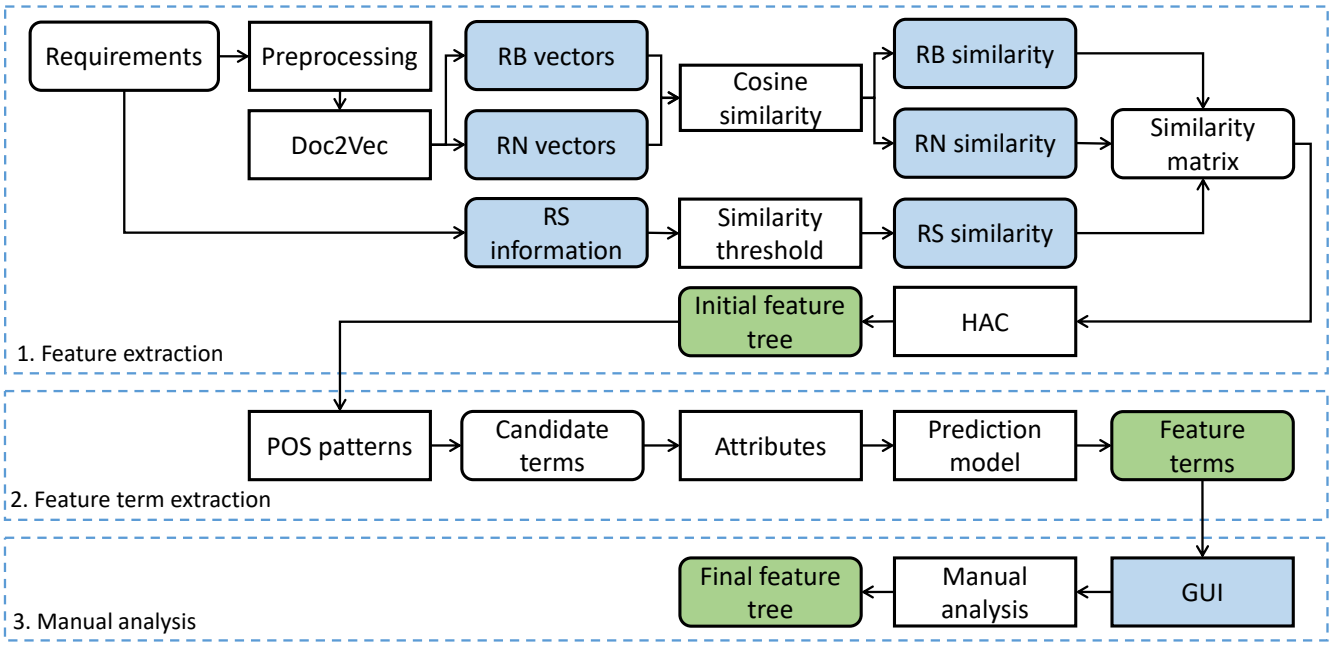


Figure 1: Overall workflow of our proposed approach for feature extraction

Doc2Vec is that the size of the document can be variable. Hence, Doc2Vec can cope with requirements that encompass different numbers of sentences. In our context, we regard each individual requirement as a document. After training the preprocessed requirements by using Doc2Vec, we obtain a vector $\vec{v}$ for each requirement that is regarded as a vector of requirement bodies.

Vectors of requirement names. Besides the vectors of the body of each requirement, we can also obtain the vector representation of each word in the requirements by using the same pre-trained Doc2Vec model for requirement bodies. However, in the case of requirements at Danfoss, the name of a requirement usually comprises a few words rather than only one word, and the average number of the words in the names is around three. While analyzing the requirements at Danfoss, we observed that, although the length of a requirement name (i.e., number of words in a requirement name) affects the similarity of each pair of requirement names, the most important influence on similarity comes from some important words. Hence, we do not take advantage of the similarity calculation method in [22], in which the length of a text will make a big difference to the similarity. In order to achieve a reasonable similarity of each pair of requirement names, we need to acquire a suitable vector representation of requirement names. To this end, we obtain the vectors of requirement names by averaging the weighted vectors of each word in a requirement name, while we use *Inverse Document Frequency (IDF)* to weigh the vectors of words [36].

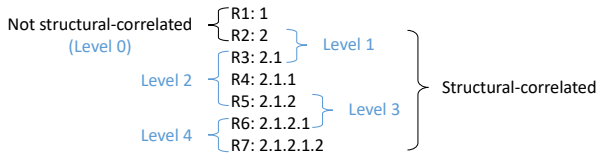
$$Vector(rn) = \frac{\sum_{i=1}^{n} IDF(w_i) \times Vector(w_i)}{n} \quad (1)$$

where,

- $Vector(rn)$ is the vector representation of a requirement name rn;
- $IDF(w_i)$ denotes the IDF value of word w_i, while $Vector(w_i)$ is the vector representation of word w_i derived from the pretrained Doc2Vec model; And, w_i belongs to rn;
- n is the number of words in rn.

Similarity of requirement bodies and names. Given all the vectors of requirement bodies, we use *cosine similarity* to measure the similarity value between two requirement bodies, that is, we measure the cosine value of the angle between two vectors of requirement bodies. In the same way, we can compute the similarity between each pair of vectors of requirement names. The smaller is the angle, the higher is the cosine similarity. In contrast with other metrics, such as Euclidean distance, even if two requirements are really similar, they may be of a large Euclidean distance, especially for the large size of requirements. This means the similarity in terms of Euclidean distance is not accurate. However, there is still a chance that the angle between the two vectors is small. As a result, cosine similarity is capable of properly calculating similarity of different sizes of requirements. Since Doc2Vec is a technique of DSMs based on neural network, we can regard the cosine similarity based on Doc2Vec as *distributional semantic similarity*.

Similarity of requirement structures. Moreover, we also take the structural information of requirements specifications into account, in particular, the hierarchical structure of such documents. For instance, functionalities are usually decomposed into different sub-functionalities described by different requirements. This way, requirements are physically grouped together according to the functionality they describe/specify. Hence, although these groups of



"R1, R2 , ..., R6": requirement id; "1, 2, 2.1, ..., 2.1.2.1.2": requirement structure.

Figure 2: An example of structural similarity

requirements might not be semantically similar to each other measured by cosine similarity, they all describe some specific functionalities that are correlated to each other. Hence, this structural correlation can also be used to adjust the similarity of requirements. We use an example shown in Figure 2 to illustrate the structural correlation. The numbers, such as "1", "2", "2.1", are used to represent the hierarchy of the requirements. We can see the structures of R1 and R2 are not correlated, since the numbers of the corresponding requirement structures (i.e., "1" and "2") are totally different. In contrast, R3, R4, R5, R6, and R7 are sub-requirements of R2 (i.e., the numbers of all these six requirement structures start from "2"), which means that these six requirements are structurally correlated. In our example, we apply five levels to measure how much the requirements are related. If the first number of the two requirement structures is not the same, the corresponding requirements are not structurally related, and thus they are at level 0 (e.g., R1 and R2). If the first number of the requirement structures is identical, the corresponding requirements are at level 1 (e.g., R2 and R3). If both, the first and second number of the requirement structures are equal, the corresponding requirements are at level 2 (e.g., R3, R4, R5). Likewise, requirements of the first three identical numbers belong to level 3, while requirements of the first four equal numbers are at level 4. And then, we preset different thresholds for *structural similarity* of the requirements at different levels.

Similarity Matrix. As a result, we compute the similarity of a pair of requirements using both, distributional semantic similarity and structural similarity, according to Equation 2. We compute this similarity for all pairs of requirements, and thus, obtain a similarity matrix of all the requirements in terms of Equation 3.

$$simReq(r_i, r_j) = wb \times SB(r_i, r_j) + wn \times SN(r_i, r_j) + ws \times SS(r_i, r_j) \quad (2)$$

$$simMatrix = \sum_{i=1}^{n} \sum_{j=1}^{n} simReq(r_i, r_j) \quad (3)$$

where,

- n is the number of requirements, while r denotes an individual requirement.
- $SB(r_i, r_j)$ is the similarity of two requirements bodies. Moreover, $SB(r_i, r_j) = cosine(\vec{v}_i, \vec{v}_j)$, in which $\vec{v}$ is the vector representations of the body of the requirement r. $SN(r_i, r_j)$ denotes the similarity of two requirements names which can be computed in the same way.
- $SS(r_i, r_j)$ denotes the structural similarity of requirements. It is a threshold predefined based on different levels that represent the extent of the structural correlation (cf. Section 4.2).
- wb, wn, and ws stand for the weights of the similarity of requirement body, name, and structure, respectively. In addition, the sum of these three weights is 1 (i.e., $wb + wn + ws = 1$).

Initial feature tree. For this step, we rely on the same technique as we introduced in a previous paper [22]. Thus, we only briefly summarize the key aspects and refer to the paper for more details. We apply HAC to group similar requirements. In this context, we consider requirements with similar functionality belonging to a particular feature. In this process, (1) the similarity matrix of the requirements is used as input of HAC; (2) the distance (i.e., dissimilarity) of each pair of requirements is calculated in order to achieve the pair of requirements with the shortest distance and merge these two requirements into a new group; (3) the distances between the newly generated group and other requirements are calculated and the pair with the shortest distance is merged again; (4) the process of computing the distance and merging the pair with the shortest distance is repeated until all the requirements are in one group, which results in a hierarchical clustering tree; (5) we use *inconsistency coefficient* [15] to flatten the hierarchy of clusters to achieve the initial feature tree.

After feature extraction, we can achieve the Feature-Requirement mapping (i.e., F-R mapping) in terms of which we know which requirements belong to a particular feature. An example of the initial feature tree and F-R mapping are shown in Figure 3.

3.3 Feature Terms Extraction

As a result of feature extraction, we gain the F-R mapping based on which domain engineers can further refine the features on different demands and existing knowledge. However, different features are comprised of different numbers of requirements. If there is a relatively large size of requirements belonging to a particular feature, it is also time-consuming to check each requirement in this feature. In order to relieve the pain of manual analysis, we improve this process by providing the feature terms from requirements to provide the semantics of the corresponding feature, that is, what is this feature about. For example, in the E-shop SPL [13], if a feature includes terms like "Credit card", "Bank transfer" and "PayPal", it is easy to know this feature is related to "Payment".

In order to obtain feature-related terms including words and phrases from the corresponding requirements, we analyze the *linguistic information* in the requirements belonging to the feature at word level using *Part-of-Speech (POS) Patterns*. The rationale behind this idea is that within requirements, certain words with specific parts of speech, such as nouns, proper nouns and adjectives, are more likely to denote functionalities than other words, and thus, are viable indicators for describing features. In this context, *proper nouns* denote domain-specific things different from common nouns, and always begin with a capital letter. Table 1 illustrates the part of speech patterns with corresponding examples.

A feature with large size of requirements may include plenty of fine-grained terms identified by using POS patterns. Moreover, not all of these terms are related to the feature. Hence, it also takes some efforts to identify the correct meaning of features just based on the terms with linguistic information (i.e., POS patterns). In order to provide hints to accelerate this process, we use a prediction model from

our previous research [23] to identify the feature-related terms by analyzing the different attributes of each candidate term extracted by using POS Patterns. These attributes comprise:

(1) Proper Noun (PN) used to indicate whether a term contains a proper noun;
(2) Sub-Term (ST) used to indicate whether a term appears in other terms;
(3) Term Frequency – Inverse Document Frequency (TF-IDF) used to indicate the significance of a term in the requirements;
(4) TextRank (TR) used to rank the terms in the requirements;
(5) C-Value (CV) used to rank the multi-word terms in the requirements;
(6) Ratio of Terms (RT) used to indicate the ratio of the number of requirements containing a term to the total number of requirements in a particular feature.

Table 1: Part-of-Speech Patterns

Pattern	Example
[PROPN]	LED
[PROPN+PROPN]	RF Module
[PROPN+Noun]	DH user
[PROPN+PROPN+PROPN]	Internet Service Provider
[Noun]	mirror
[Noun+Noun]	finger protection
[ADJ+Noun]	remote control
[ADJ+Noun+Noun]	central locking system
[ADJ+ADJ+Noun]	digital programmable thermostats

PROPN: Proper noun. ADJ: Adjective.

As the last step, not directly related to the actual extraction, our technique includes a GUI to assist domain engineers to analyze the extracted features and adjust them if necessary. In Figure 3, we present a screenshot of this GUI. The initial feature tree is visualized on the left side of the figure, while the requirements belonging to a feature are also listed in terms of F-R mapping at the top right of the figure. Moreover, the candidate terms extracted from features are also presented with six attributes at the bottom right of the figure. TF-IDF, TR, and CV are normalized by using Min-Max Scaling. Furthermore, if the "Prediction" of a candidate term is marked as 1, this term is regarded as a feature-related term (e.g., "finger protection" and "led"). Domain engineers can further revise, name, add and remove the features, while the structure is also able to be adjusted by using the GUI.

4 EVALUATION

In this section, we provide details about the empirical evaluation of our feature extraction technique. In particular, we present our research questions and details about the subject system, briefly describe how we applied our technique for the extraction process, and eventually present the results. For the latter, we focus on not only the accuracy of the extracted features but also the applicability of the extracted terms and the effectiveness of our approach in order to assist domain engineers for extracting features in practice.

4.1 Subject System and Research Questions

Dataset. The requirements we used for our evaluation come from drives, also known as frequency converters, which convert incoming power, usually 50 or 60 Hz, into a different output frequency. Being able to control this makes it possible to adjust the shaft speed or torque of an electrical motor. This may prolong the lifetime of the motor and help on saving both energy and money. Frequency converters are used within a wide range of applications, spanning from simple fan control to complex crane and bottle plant applications. Being able to cover these different applications requires an enormous amount of software, originating from an extensive requirement specification that evolves for years.

Overall, our dataset contains 2389 individual requirements from Danfoss, which sums up to 409 339 words (459 567 tokens). Moreover, the format of the requirements is not uniform. The size of each individual requirement ranges from dozens of words to hundreds of words. Finally, the requirements came with some specialities that we did not expect and had to address in our extraction process. First, each requirement was similarly structured in paragraphs, each with a dedicated name such as "description" or "rationale". Obviously, this would falsify our results, as these words would be considered as important and key terms by our technique. Thus, we introduced the blacklist in the preprocessing part. Second, the requirements are structured hierarchically, and thus, come with some kind of context. This additional information has finally lead us to the decision, to include the structural similarity, described in Section 3.2.

Research Questions. For our study, we formulate the following research questions.

***RQ1**: What is the accuracy of the automatically extracted features?*

Only with relative high confidence in the extracted features, our technique is practically applicable. Hence, with this RQ we aim at measuring the accuracy by means of precision.

***RQ2**: Do the extracted feature terms provide hints for domain engineers to identify the extracted features?*

To evaluate our feature term extraction part, we use a quantitative indicator to show whether the extracted terms constitute some important characteristics of the functionally related feature. To this end, we also calculate the *precision* of the meaningful terms.

***RQ3**: How can our proposed approach improve the process of domain analysis?*

While accuracy is a fundamental aspect of our technique, it is also of superior interest whether and how this information can assist domain engineers in recreating domain knowledge, which is a tedious and time-consuming task when done manually. To address the question, we go beyond quantitative measures and aim at qualitatively discuss how much our GUI for presenting the results to domain engineers can assist them in the process of domain analysis based on the automatically extracted results.

4.2 Extraction Process

We performed feature extraction according to our technique, described in Section 3 on all of the 2 389 requirements. Next, we briefly describe each step and its output.

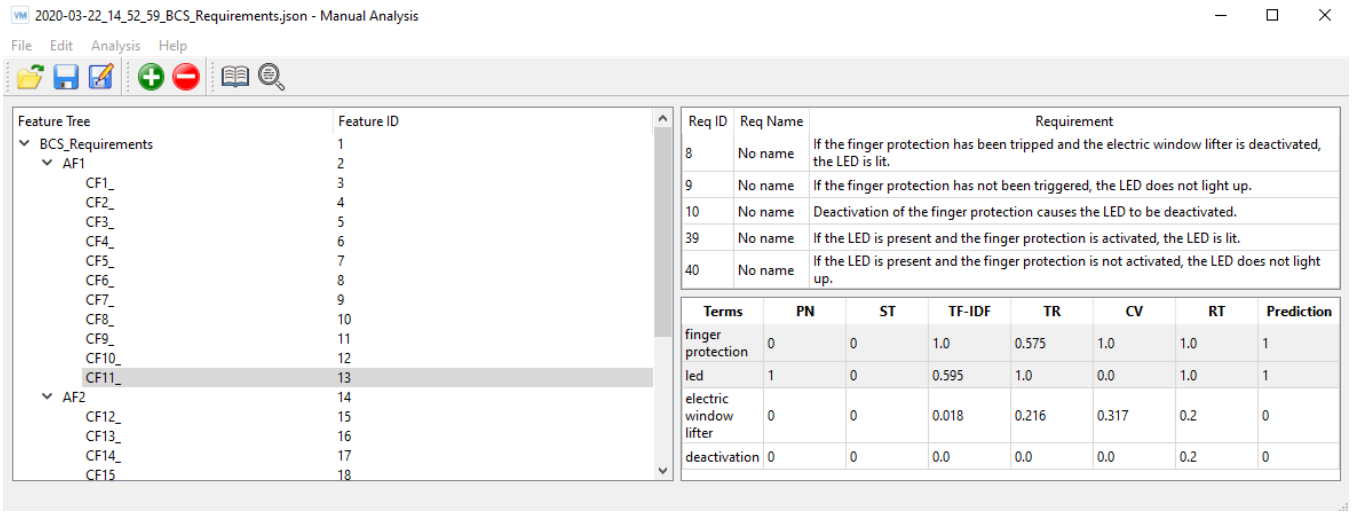


Figure 3: The GUI for manual analysis with a feature tree view, a list of corresponding requirements, and associated feature terms.

(1) As the first step, we applied preprocessing to all raw requirements in order to reduce the complexity. In this process, apart from applying basic NLP techniques, we also removed null requirements and requirements with very few words (i.e., less than five words), eventually resulting in 2231 processed requirements that we could use for the actual feature extraction. The reason for removing the requirements with very few words is that: a) these requirements do not contain specific descriptions of functionality; or b) some requirements are documented through plenty of figures without enough textual information.

(2) Second, a language model was trained by using Doc2Vec on the preprocessed requirements dataset. Based on the pretrained Doc2Vec language model, we obtained the vector representation of each requirement body and name. We then use these vectors to compute the pairwise cosine similarity, according to Section 3.2, and thus, to achieve the distributional semantic similarity of each pair of requirement bodies and names. Moreover, based on the structural correlation (i.e., levels), we preset the overall threshold for the structural similarity of the requirements at the same level, for example, 0 at level 0, 0.3 at level 1, 0.5 at level 2, 0.7 at level 3 and 1.0 at level 4. We used five levels to calculate the structural similarity, since we intended to achieve features of moderate granularity. The specific threshold for each level was determined in order to achieve a proper ratio between structural similarity and distributional semantic similarity. Moreover, in Equation 2, wb, wn and ws are equal to 0.2, 0.2 and 0.6, respectively. According to Equation 3, a 2231×2231 similarity matrix was achieved.

(3) As a third step, the similarity matrix was fed into HAC in which we set the inconsistency threshold to 1.1. The value of inconsistency threshold also affects the granularity of the features. In order to achieve a moderate granularity, we determined this value (i.e., 1.1), also taking the inconsistency thresholds in our previous research [22] into account. After HAC, we achieved 499 features resulting in the initial feature tree.

(4) Finally, one domain engineer used the GUI to visualize the extracted features and check whether the extracted features are applicable in practice and whether the feature terms could assist engineers to identify a feature. This process obeys the evaluation metrics in Section 4.3.

Then, the results were double reviewed by another engineer in order to reduce bias. The final results are used to answer the research questions.

4.3 Evaluation metrics

In order to achieve the quantitative analysis of the results, we use precision to measure the extracted domain knowledge. Since there is no ground truth (i.e., no domain model exists so far), we had to ask domain engineers to evaluate whether the extracted information is meaningful or not. This information about validity includes both, features and feature terms, and the process and metrics are described in the following.

Metric for RQ1. In order to address the accuracy of extracted features, all the extracted features are checked by engineers to determine whether the extracted feature is reasonable and useful. In particular, an extracted feature is considered *meaningful*, if it represents a certain functionality of the system. Moreover, we distinguish between three different kinds of a meaningful feature:

(1) whether a particular meaningful feature exactly represents a particular kind of functionality;

(2) whether a meaningful feature can be merged with another meaningful feature, which means several meaningful features, representing related functionality, can be converted into a single feature.

(3) whether a meaningful feature can be separated into some fine-grained features, which means a meaningful feature with relatively related functions can be separated into several sub-features.

The equation of calculating the precision of meaningful features is as follows:

$$Precision = \frac{The\ number\ of\ meaningful\ features}{The\ number\ of\ extracted\ features} \quad (4)$$

Metric for RQ2. We use a prediction model to extracted feature terms, taking both linguistic and statistical information into account. In order to measure how much useful feedback engineers can achieve from these terms, we evaluate the extracted feature terms following the rules below:

(1) The extracted feature terms are reviewed by domain engineers to determine whether the feature terms are meaningful. Meaningful terms describe a specific functionality and are highly related to the extracted feature, which means that engineers can easily get an idea of what the extracted feature is about based on the meaningful terms.
(2) We only measure the meaningful terms in meaningful features. The reason is that requirements belonging to meaningless features are wrongly grouped, and thus, the terms do not describe the obvious intention of the meaningless feature.

The equation of calculating the precision of meaningful terms in one feature is as follows:

$$Precision = \frac{The\ number\ of\ meaningful\ terms}{The\ number\ of\ extracted\ feature\ terms} \quad (5)$$

Based on the precision of meaningful terms for each feature, we then can compute the precision of all meaningful features.

4.4 Results

RQ1 – Accuracy. In order to present a comparative result, we not only apply the refined approach (Approach I) proposed in this paper, but also use the previous method (Approach II) [22] to process the requirements to extract features, shown in Table 2. As presented above, the information about the body, name and structure of the requirements is analyzed by using the refined approach. Meanwhile, approach II only processes the body of the requirements. According to Table 2, we achieve higher precision (0.783) by using approach I than the precision (0.627) obtained by using approach II. This shows that analyzing different information in the requirements can extract meaningful features more accurately than only taking one type of information into account.

We further analyze the granularity of meaningful features that have been extracted. The MMEF and SMEF (cf. Table 2) that are a subset of the overall meaningful extracted features indicate whether the granularity of meaningful features is appropriate. By using approach I, 12.8% of the meaningful extracted features can be merged with each other and 6.9% of them are able to be separated into more sub-features. Compared with using approach II, the corresponding ratios are 31.2% and 14.2%, respectively. Both ratios of approach I are considerably smaller than the corresponding ratios of approach II, which shows that the results of approach I tend to provide features with appropriate granularity.

We observe that approach II is prone to group the requirements with a similar writing style. However, there may be a slight difference in the functionality they specify. This difference affects the accuracy of feature extraction and the granularity of the extracted features. We illustrate this problem by means of two requirements from Digital Home [11]:

Table 2: The results of extracted features by using two approaches

Approach	No. of All EF	No. of MEF	No. of MMEF	No. of SMEF	Precision
I	499	391	50	27	0.783
II	649	407	127	58	0.627

EF: extracted features; MEF: meaningful extracted features; MMEF: the meaningful extracted features that should be merged with other features; SMEF: the meaningful extracted features that should be separated into several sub-features.

> R1: The *thermostats* shall be used to monitor and regulate the *temperature* of an enclosed space.
> R2: The *humidistats* shall be used to monitor and regulate the *humidity* of an enclosed space.

R1 and R2 specify the functionalities about thermostats and humidistats, respectively. Since the writing style of these two requirements is similar (i.e., the majority of the words in these two requirements are identical), they are prone to be grouped into one feature by using approach II to form a coarse-grained feature "thermostats and humidistats". If the goal is to achieve fine-grained features, manual analysis is required to separate feature "thermostats and humidistats" into feature "thermostats" and feature "humidistats". Certainly, these two requirements are still functionally related to each other to a certain extent. Hence, the obtained coarse-grained feature is also meaningful to identify the main functionality. However, let us assume that if there are two requirements written in a similar style, but the meanings expressed are completely different, it would have a bad impact on the precision of feature extraction.

However, the requirements are clearly structured in the case of Danfoss. Hence, approach I is designed to take requirement name and structure into account to capture the specialties of the requirements in Danfoss. We have re-edited R1 and R2 according to the format of the requirements in Danfoss. We use the re-edited R1 and R2 as an example to illustrate approach I, shown below:

> R1: 1 - Thermostats - The thermostats shall be used to monitor and regulate the temperature of an enclosed space.
> R2: 2 - Humidistats - The humidistats shall be used to monitor and regulate the humidity of an enclosed space.

The first, second, and third parts denote the structure, name, and body of the requirements, respectively. Due to the difference of the structure information (e.g., "1" and "2") of the two requirements, the overall similarity will be reduced, which leads to the fact that these two requirements tend to be split into two features. For the names (e.g., "Thermostats" and "Humidistats") of R1 and R2, if only R1 and R2 are considered and analyzed, the position of the word vectors of "Thermostats" and "Humidistats" may be very close in the vector space because their surrounding words are the same. In other words, the angle between the two vectors is very small, which causes the two names to have high similarity. However, in the entire requirements document, these two words appear not only in these two individual requirements with a very similar writing style

Table 3: The results of extracted features terms

Category	Range	Precision
C1	0 < No. of Terms ⩽ 20	0.705
C2	20 < No. of Terms ⩽ 40	0.657
C3	40 < No. of Terms ⩽ 60	0.639
C4	60 < No. of Terms ⩽ 80	0.606
C5	80 < No. of Terms ⩽ 100	0.596
C6	No. of Terms > 100	0.552

but also in other requirements written with different surrounding words. Therefore, the distribution of these two words in the vector space is different, which reduces the similarity of the two words and further affects the results of feature extraction.

RQ1: While a precision of ≈ 75% for the extracted features is not very high, we consider this as a reliable basis for engineers to further refine the true features. Moreover, the additional information about the structure and name of requirements is beneficial to capturing the specialties of the requirements, thus improving the accuracy.

RQ2 – hints from feature terms. We use the technique presented in Section 3.3 to identify feature-related terms by using the refined approach (approach I). The number of the feature terms extracted from each feature ranges from 2 to 262. Overall, we identified 13078 feature terms from all meaningful features, and 8196 of them are useful, resulting in an overall precision of 0.627.

Since the number of feature terms extracted from each feature is different, the overall precision cannot reflect how this difference affects the accuracy of the results. In order to provide multiple perspectives, we divide the results for extracted feature terms into six categories according to the number of terms extracted from each feature (denoted as *No. of Terms* in the following). The six categories (C1-C6) are shown below:

C1: If 0 < No. of Terms ⩽ 20 , the results belong to C1;
C2: If 20 < No. of Terms ⩽ 40, the results belong to C2;
C3: If 40 < No. of Terms ⩽ 60, the results belong to C3;
C4: If 60 < No. of Terms ⩽ 80, the results belong to C4;
C5: If 80 < No. of Terms ⩽ 100, the results belong to C5;
C6: If No. of Terms > 100, the results belong to C6;

From Table 3, we can see that the precision gradually decreases as the number of feature terms extracted from a feature increases. The potential reason is that the prediction model we used is trained on a small dataset, which limits its ability to process relatively large size of terms. However, the feature terms are intended to help domain engineers to understand the intention of the feature at a glance. Therefore, the feature terms are used as an optional reference when the engineers analyze the extracted feature. In fact, engineers do not need to check whether every feature term is meaningful but should get some hints from the feature terms when dealing with some complex features. Consequently, even in cases where high accuracy is not achieved, the relatively accurate feature terms are still useful for analyzing the intention of features.

RQ2: We achieve relatively accurate feature terms from features, especially for C1, while these meaningful feature terms are capable of providing useful hints to assist domain engineers to have a quick overview of an extracted feature in the process of further analyzing the extraction results.

RQ3 – improving the process of domain analysis. Domain analysis is a form of requirement engineering aiming at identifying features as reusable artifacts in the context of SPL [16]. However, feature extraction from a legacy system by domain engineers is time-consuming from scratch. In order to improve the process of domain analysis, especially for extracting features, we aim at improving the effectiveness by combining automated features and feature terms extraction with the assistance of a dedicated GUI.

According to the specialty of Danfoss requirements, we propose an improved approach, in which we not only analyze the main content of the requirements, but also consider the impact of the structure and name of the requirements on feature extraction. In addition, we design weights for the similarity calculations for the body, name, and structure of the requirements. Domain engineers can adjust the three weights according to the characteristics of the requirements and the specific needs when constructing the feature model to obtain the features of different perspectives (e.g., perspectives of the body, name, and structure). The adjustment of weights requires domain engineers to have a certain understanding of the target requirements, which can make the results of feature extraction in line with expectations, that is, more meaningful features are extracted. For example, in the Danfoss case, we focus on obtaining features from the perspective of requirement structure, that is, the weight (i.e., *ws*) for requirement structure is higher than for the others. As a result, the automated extracted domain knowledge can be used as the first step to generate a domain model, while increasing the efficiency compared with creating a domain model from scratch, especially for processing large size of requirements.

When analyzing features extracted from requirements with a name, the names of the requirements can be used as the first clue to suggest the notion of the extracted feature. However, we observe that the meaning of a feature sometimes can not be obtained intuitively from the requirement names. For example, in an extracted feature, the names of each requirement are so different that we cannot directly comprehend the commonality of the requirements. In this case, the feature terms can provide a reference for identifying the meaning of the feature. In the case that features are extracted from the requirements without names, the feature terms can play an important role in providing the main clue for inferring the intention of features.

Apart from the automatically extracted domain knowledge, we provide a GUI specifically designed for analyzing the requirements and extracted features, and thus, to improve the generated domain models. The majority of previous approaches (cf. Section 2) do not provide a tool for further visualizing and revising the extraction results, resulting in lack of applicability in practice. A part of the previous researches used *FeatureIDE* [17] to visualize the extracted feature tree. Although FeatureIDE is a very applicable tool for domain engineers to manually build a feature model by analyzing the requirements, it lacks the ability to present the key information (i.e., feature terms) which can support engineers to adjust the

automated extraction results. By contrast, our dedicated GUI not only can directly show the extracted feature tree with the mapped requirements, but also integrate the feature terms extraction function for each feature in order to provide hints for engineers to understand the intention of features.

Figure 3 presents the GUI with an example of the extraction results. The feature terms for each feature are obtained by analyzing the six attributes of the candidate terms through a prediction model. However, considering that the extracted feature terms are not accurate enough in some cases, the GUI not only displays the extracted feature terms but also displays the corresponding six attributes for manual analysis. We argue that these attributes also play a role when analyzing the intention of the features. For example, in the case of Danfoss, terms with higher statistical significance values (e.g., TF-IDF, TR and CV) are usually more capable of representing the meaning of the extracted features. Moreover, when the number of feature terms exceeds 80, the terms with the higher RT value or ST marked as 1 can better refer to the notion of features. Besides showing the extraction results, the extracted initial feature tree is editable in the GUI. In detail, 1) the F-R mapping can be easily adjusted by drag and drop operation; 2) features can be deleted and new features can be added; 3) names of the features can be edited; 4) the tree structure can be freely adjusted.

The domain engineers, reviewing results for this study, used our tool and confirmed that this was very useful in understanding the intention of features, thus, being able to assess their semantics and validity. Moreover, having access to the requirements belonging to a feature was also considered useful, as it allows quickly reviewing the scope of a feature. Furthermore, during the process of using our techniques in analyzing real-world requirements, we found that a particular approach cannot accurately analyze different types of requirements documents. When dealing with each specific type of requirements document, the specialty of the requirements should be taken into account. To this end, we conclude the general process of applying feature extraction techniques to identify features from requirements in the following: 1) domain engineers initially check the requirements to identify key special points in the requirements, for example, whether the requirements are structured well; 2) Then, data analysts analyze other characteristics of the requirements, for example, the number of words contained in the requirements, the characteristics of the requirement names and the writing format of the requirements; 3) The data analysts determine the appropriate algorithm and corresponding parameters (e.g., thresholds) to extract features based on the feedback from domain engineers; 4) Domain engineers work on the extracted results to achieve the final feature tree/model via a tool (e.g., the GUI proposed in this paper).

RQ3: Based on the feedback, we argue that only providing a complete framework that combines an automated extraction process together with a guided and graphical presentation for manual adjustment can help domain engineers in practice to recreate domain knowledge from requirements.

5 THREATS TO VALIDITY

Construct validity. We regard that the requirements with related functionality can be grouped into the same feature, which is actually based on the similarity of the requirements. We not only achieve the distributional semantic similarity of the requirement names and bodies by using Doc2Vec, but also take the structure similarity into consideration. Although our similarity-based method might result in a bias for identifying features, the empirical evaluation results reveal that our proposed approach is capable of providing useful features for domain engineers as a reference.

Internal validity. In the process of automated feature extraction, there are several parameters needed to be predefined, such as inconsistency threshold and structure similarity. The inconsistency threshold for clustering affects the granularity of the extracted features. Domain engineers can reduce the inconsistency threshold to achieve fine-grained features and increase it to gain coarse-grained features. We selected a relatively appropriate inconsistency threshold in order to achieve features with moderate granularity, taking the case study of our previous research [22] into account. For the structure similarity, the number of levels also affects the granularity of the extracted features, while the weights for structure, name, and body similarity impact on the ratio of the structure similarity to the distributional semantic similarity in the final similarity matrix. We preset the structure similarity and weights based on our prior experience of requirement analysis and the observations from the Danfoss requirements to achieve features with moderate granularity. The results reveal that the preset parameters are relatively appropriate. We cannot say the fixed parameters in our case study fits all other different datasets only in terms of the results of our empirical evaluation. However, the parameters are flexible to be changed by domain engineers on the demands of different domains.

Conclusion validity. Our evaluation results were conducted by manual analysis, which means bias from engineers exists in this process. In order to reduce the bias, two engineers participated in the analysis, while one engineer finished the overall results and another engineer double checked them to reach a consensus. Moreover, although we used the real-world requirements to verify the applicability of our approach, we are still not sure whether the proposed approach can cope with the real-world requirements in different domains. However, based on our insights, adjustments to algorithms and corresponding parameters are necessary, when dealing with requirements in different domains.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed our approach to improve the process of domain analysis. Especially, 1) we used neural document embedding techniques to gain an accurate language model of the requirements; 2) we achieve the similarity of the requirements taking both distributional semantic similarity and structure similarity into account; 3) HAC was applied to gain the initial feature tree; 4) we developed a dedicated GUI used as a tool to visualize the automated extracted results and support engineers in the process of domain analysis.

This is the first exploration that the technique of feature extraction from requirements is used to improve the process of domain analysis in practice. Our study demonstrated that our approach is capable of assisting domain engineers to extract features.

In the future work, we will use more data from different domains to evaluate our approach and collect more feedback from engineers to improve the approach in order to increase the efficiency of domain analysis to a great extent.

REFERENCES

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. 2012. On Extracting Feature Models From Product Descriptions. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems*. ACM, 45–54.

[2] 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.

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

[4] N. H. Bakar, Z. M. Kasirun, N. Salleh, and H. A. Jalab. 2016. Extracting Features from Online Software Reviews to Aid Requirements Reuse. *J. Applied Soft Computing* 49 (2016), 1297–1315.

[5] 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.

[6] 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.

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

[8] J. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans. 2013. Feature Model Extraction from Large Collections of Informal Product Descriptions. In *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering*. ACM, 290–300.

[9] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. 2011. On-demand Feature Recommendations Derived from Mining Public Product Descriptions. In *Proc. Int'l Conf. Software Engineering*. ACM, 181.

[10] Thomas Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. 2016. Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead. ACM, 252–261.

[11] T. B. Hilburn and M. Towhidnejad. 2007. A Case for Software Engineering. In *Software Engineering Education & Training, 2007. CSEET'07. 20th Conference on*. IEEE, 107–114.

[12] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. 2016. Preprocessor/Based Variability in Open-/Source and Industrial Software Systems: An Empirical Study. 21, 2 (2016), 449–482.

[13] 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.

[14] 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.

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

[16] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature/Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.

[17] 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.

[18] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *Proceedings of the International Workshop on Software Product-Family Engineering*. Springer, 282–293.

[19] 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.

[20] Q. Le and T. Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the International Conference on International Conference on Machine Learning*. JMLR.org, II–1188–II–1196.

[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] Yang Li, Sandro Schulze, and Gunter Saake. 2018. Reverse Engineering Variability from Requirement Documents based on Probabilistic Relevance and Word Embedding. In *Proc. Int'l Conf. Systems and Software Product Line Conference*. ACM, 121–131.

[23] Yang Li, Sandro Schulze, and Jiahua Xu. 2020. Feature Terms Prediction: A Feasible Way to Indicate the Notion of Features in Software Product Line. In *Proc. Int'l Conf. Evaluation and Assessment in Software Engineering*. ACM, to appear.

[24] 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.

[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] N. Salleh N. H. Bakar, Z. M. Kasirun and A. H. Halim. 2017. Extracting Software Features from Online Reviews to Demonstrate Requirements Reuse in Software Engineering. In *Proc. Int'l Conf. Computing and Informatics*. Sintok: School of Computing, 184–190.

[29] 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.

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

[31] 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.

[32] Peter Toft, Derek Coleman, and Joni Ohta. 2000. A Cooperative Model for Cross/Divisional Product Development for a Software Product Line. In *Proc. Int'l Software Product Line Conference*. Springer, 111–132.

[33] 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.

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

[35] 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.

[36] Jiang Zhao, Man Lan, and Junfeng Tian. 2015. ECNU: Using Traditional Similarity Measurements and Word Embedding for Semantic Textual Similarity Estimation. In *The International Workshop on Semantic Evaluation (SemEval)*. Association for Computational Linguistics, 117–122.