Automated Extraction of Feature and Variability Information from Natural Language Requirement Specifications

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M.Sc. Yang Li

geb. am 14.06.1987 in Hebei, China

Gutachterinnen/Gutachter

Prof. Dr. Gunter Saake
Prof. Dr. Andreas Nürnberger
Prof. Dr. Rick Rabiser

Magdeburg, den 30.10.2020

Abstract

Software Product Lines support structured reuse of software artifacts 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. Feature and variability information extraction from different artifacts is an indispensable activity to support the systematic integration of single software systems and software product line. However, for a legacy system, it is non-trivial to gain information about commonalities and differences of the variants. Beyond manually extracting commonalities and variabilities, a variety of approaches, such as feature location in source code and feature extraction in requirements, has been proposed to provide 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 thesis, we provide a systematic literature review, which contains a multi-dimensional overview of feature extraction approaches from natural language documents. Based on the observations from studies, we provide feasible and accurate approaches to improve the efficiency of feature extraction. To achieve this goal, we first explore the application of deep learning technologies in feature extraction. Second, we propose a hybrid approach based on multiple natural language processing and data mining techniques to extract features and variability information. Third, in order to provide understandable notations for features, we propose an approach combining keyword extraction and machine learning methods to predict feature-related terms. Fourth, we apply the proposed feature extraction approaches to analyze the requirements from a real-world scenario in practice, where we adjust the framework and combine other algorithms in terms of the specialities of real-world requirements. We empirically present how our proposed approaches can be used to extract features and variation points, while results show the usage of the proposed approaches can benefit the extraction process.

Zusammenfassung

Software-Produktlinien unterstützen die strukturierte Wiederverwendung von Software Artefakten, um die Wartung und Weiterentwicklung der normalerweise großen Anzahl von Varianten zu realisieren, was die Industrialisierung der Softwareentwicklung insbesondere für softwareintensive Produkte fördert. Die Extraktion von Feature und Variabilitätsinformationen aus verschiedenen Artefakten ist eine unverzichtbare Aktivität, um die systematische Integration einzelner Softwaresysteme und Software-Produktlinie zu unterstützen. Für ein Altsystem ist es jedoch nicht trivial, Informationen über Gemeinsamkeiten und Unterschiede der Varianten zu erhalten. Neben dem manuellen Extrahieren von Gemeinsamkeiten und Variabilitäten wurden vielfältige Ansätze vorgeschlagen, z. B. die Position von Features im Quellcode und die Extraktion von Features in Anforderungen, um Features und ihre Variationspunkte automatisch zu identifizieren. Im Vergleich zum Quellcode enthalten die Anforderungen umfassendere Variabilitätsinformationen und bieten Rückverfolgbarkeitsverknüpfungen zu anderen Artefakten aus frühen Phasen der Softwareentwicklung. In dieser Arbeit bieten wir eine systematische Literaturrecherche, die einen multidimensionalen Überblick über Ansätze zur Feature-Extraktion aus Dokumenten in natürlicher Sprache enthält. Basierend auf den Beobachtungen aus dieser Studie schlagen wir praktikable und genaue Ansätze zur Verbesserung der Effizienz der Feature-Extraktion vor. Um dieses Ziel zu erreichen, untersuchen wir zunächst die Anwendung von Deep-Learning-Technologien bei der Feature-Extraktion. Zweitens schlagen wir einen hybriden Ansatz vor, der auf mehreren Techniken zur Verarbeitung natürlicher Sprache und Data-Mining basiert, um Informationen von Feature und Variabilität zu extrahieren. Darüber hinaus präsentieren wir einen Ansatz, der Schlüsselwortextraktion und Methoden des maschinellen Lernens kombiniert, um feature-bezogene Termini vorherzusagen, damit verständliche Notationen für Features bereitgestellt werden können. Schließlich wenden wir die zuvor präsentierten Ansätze zur Feature-Extraktion an, um die Anforderungen aus einem realen Szenario in der Praxis zu analysieren, wobei wir das Framework anpassen und andere Algorithmen im Hinblick auf die Besonderheiten realer Anforderungen kombinieren. Empirisch präsentieren wir, wie von uns gestellte Ansätze verwendet werden können, um Features und Variationspunkte zu extrahieren. Zugleich zeigen die Ergebnisse, dass die Verwendung dieser Ansätze dem Extraktionsprozess zugutekommen kann.

Acknowledgements

I would like to express my deepest gratitude to Gunter Saake. He gave me the opportunity to pursue my Ph.D. under his supervision, provided an excellent research environment, and gave me the freedom to choose my research direction.

I would like to thank Sandro Schulze for his valuable and constructive suggestions during the planning and development of my research. His attentive guidance, comments, and feedback improved my academic writing skills. During the last four years, we had numerous fruitful discussions that had a major impact on my research. His willingness to give his time so generously has been very much appreciated.

I would like to offer my special thanks to all my colleagues in our team without whom I could not achieve fruitful discussions. The advice given by them has been a great help in my research. I am particularly grateful for the assistance given by Xiao Chen, Sebastian Krieter, Jacob Krüger, Wolfram Fenske, David Broneske, Juliana Alves Pereira, Mustafa Al-Hajjaji, Gabriel Campero Durand, Fabian Benduhn, Jens Meinicke, and Anja Buch. They not only supported me in my research, but also helped me solve the problems encountered in life.

I would also like to thank all the researchers I met on the path of my research. In particular, I thank Thomas Fogdal, Helene Scherrebeck, Jiahua Xu, Stefania Gnesi, and Laura Semini. The collaboration, feedback, and comments from them had a great impact on my research. Moreover, I would like to thank Andreas Nürnberger and Rick Rabiser for being the reviewers of my thesis.

Last but not least, I want to thank my girlfriend Bowen who brings a lot of joy and happiness to our life. I am very grateful to my parents and brother for their selfless support and love.

Contents

List of Figures

List of Tables

List of Acronyms

AA	Average Accuracy
BCS	Body Comfort System
CAO	Clone and Own
CNN	Convolutional Neural Network
CTC	Cross-Tree Constraint
DH	Digital Home
DCNN	Dynamic Convolutional Neural Network
DSM	Distributional Semantic Model
DT	Decision Tree
GNB	Gaussian Naive Bayes
GUI	Graphical User Interface
HAC	Hierarchical Agglomerative Clustering
IDF	Inverse Document Frequency
KNN	K Nearest Neighbors
LE	Laplacian Eigenmaps
LR	Logistic Regression
LSA	Latent Semantic Analysis
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
NMI	Normalized Mutual Information
POS	Part of Speech
RF	Random Forest
RTE	Recognizing Text Entailment
SLR	Systematic Literature Review
SPL	Software Prodcut Line
SPLE	Software Prodcut Line Engineering
SRL	Semantic Role Labeling
SRS	Software Requirements Specifications
SVM	Support Vector Machine
TE	Textual Entailment
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
VSM	Vector Space Model

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 [Kru01], 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 relations 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.

Software Product Line Engineering (SPLE) has been proposed as a large-scale development methodology that enables the efficient development of related software systems from a common set of core assets in a prescribed way [CN01, PBL05]. The resulting *Software Product Line (SPL)* then comprises a set of software-intensive systems that can be distinguished by their commonalities and differences (also known as variabilities) in terms of features. A *feature* in this context is a user-visible increment in functionality [ABKS13]. Based on (de-)selecting features, a particular variant can be tailored and generated based on the developed core assets. While SPLE has been proven to be beneficial in practice [TCO00, FSK+16, HZS+16], it takes significant efforts to introduce SPLE from scratch, as domain features and their relations must be known, requiring a detailed domain analysis [KCH+90]. Also, setting up the whole process usually implies a considerable overhead, as it must be ensured that both domain and implementation artifacts are evolved together and exhibit clearly defined variation points [PBL05]. 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 [Kru01].

Usually, software development does not start with an SPLE approach, because a) it induces high upfront costs (e.g., domain analysis) and b) it is mostly unclear whether a vast amount of variants is needed. Hence, traditional development approaches are preferred with ad-hoc mechanisms used to introduce variability. In particular, it is common practice in industry to apply *Clone and Own (CAO)*, that is, replicating an existing system and adapting it according to the new requirements [RR03, DRB+13]. Although this ad-hoc practice comes with low effort and is easy to use, information about commonalities and differences amongst the cloned systems is lost, thus impeding the maintenance and evolution of the typically large number of variants. At some point, the aforementioned procedure becomes impractical, and thus, an SPLE approach is introduced, using either a *reactive* or *extractive* migration strategy [Kru01]. A crucial point during this transition is to identify features and the variation points among them, i.e., how they relate to each other (e.g., alternative features or exclude/require relation between features). Especially for legacy systems that have evolved over years, it is non-trivial to extract this information, and thus, automation is needed to support this step. Hence, *reverse engineering* techniques such as feature location and extraction are typically used to support the migration process.

While feature location has been subject to intensive research [DRGP13], their applicability for reverse engineering in the context of software product lines has two crucial limitations:

1. Existing feature location techniques predominantly focus on single software systems, while information about variation points is missing. Hence, the extracted information is insufficient for migrating to an SPLE process.
2. Since existing techniques focus solely on source code, additional effort may be required for feature extraction of other artifacts (e.g., requirements, models, documentation) due to missing traceability links from the source code.

We argue that these limitations can be cured by focusing on *Software Requirements Specifications (SRS)* as the primary artifact for feature and variability extraction. Due to considerable progress in *Natural Language Processing (NLP)*, a variety of information, including variation points, can be extracted from such requirements, thus addressing limitation 1. Moreover, requirements are the initial development artifact and usually traceability links to all other artifacts in later development phases, such as source code or test cases, are maintained. Hence, by extracting variability information from SRS documents, we can exploit these links for mapping of feature and variability information to other artifacts, thus resolving limitation 2.

1.1 Goal of the Thesis

The goal of this thesis is to investigate how to extract features and reason about the relationships between them by means of requirements. To this end, we aim at answering the following research questions:

RQ1: *How do the current studies support the process of feature and variability extraction from requirements?*

The goal of this question is to investigate the current state-of-the-art of feature and variability extraction from *natural language* documents by means of a comprehensive

literature survey. According to the result of this question, we can achieve the existing gaps and challenges that can guide our research in this field.

RQ2: *How to identify features and variation points from requirements automatically with high accuracy?*

This is the core question of this thesis. We intend to present a feasible framework to automate the process of identifying features and variability information from requirements. To answer this question, we provide techniques to extract features and their corresponding requirements as well as the relationships between features, resulting in the main contribution of this thesis.

RQ3: *How can we speed up the process of figuring out the intention of a feature?*

Although features with the corresponding requirements can be extracted automatically, the intuition behind the extracted feature is usually unknown. That is to say, it still lacks a brief description of a feature or a name of a feature that is pivotal for domain engineers to achieve a complete view of features. Since features are the indispensable concerns in SPLE, quickly understanding the extracted features can further improve the work efficiency for domain engineers to analyze the extraction results. Hence, we aim at providing an approach to infer the notion of a feature.

RQ4: *How the feature and variability extraction techniques can benefit the software product lines in a real-world scenario?*

For this question, we focus on using the feature and variability extraction techniques in practice. This includes choosing specific methods based on specific application scenarios as well as how to combine different methods to form a user-friendly tool or explicit procedures to improve the efficiency of domain engineers for analyzing the commonalities and variabilities from natural language documents.

1.2 Structure of the Thesis

In order to present our contributions explicitly, we divide the thesis into the following 8 chapters:

In Chapter 2, we briefly introduce the necessary basics of SPLE including domain engineering and application engineering. Although CAO is also targeted software reuse, the development of variant-rich systems with CAO differs considerably from what SPLE proposes. Hence, we make a comparison between software product lines and CAO to further conclude the drawbacks of CAO. Moreover, we also introduce the related NLP concepts and techniques used in this thesis.

In order to address RQ1–RQ4, the thesis presents four main contributions:

RQ1: In Chapter 3, we provide a multi-dimensional overview of approaches for feature and variability extraction from natural language documents by means of a Systematic Literature Review (SLR). We selected 31 primary studies and carefully evaluated them regarding different aspects such as techniques used, tool support, or accuracy of the results. Moreover, we offer key insights and observations that guide us to derive future challenges, arguing that more effort needs to be invested in making such approaches applicable in practice.

RQ2: In Chapter 4, we identify several potential shortcomings of previous research in terms of insights derived from Chapter 3. In order to cope with these shortcomings, we propose an initial approach with a self-learning structure that aims at learning the linguistic characteristics of the requirements to realize extracting features, reducing the usage of external tools for obtaining the semantic and syntactic information. Initial results show that accuracy is still limited, but that our approach allows us to automate the entire process. In Chapter 5, we propose a general framework named *VarMine* to extract features and variability information from requirements, integrating different information retrieval, data mining and NLP technologies. The results reveal that our approach identifies the majority of features correctly and also extracts variability information with reasonable accuracy.

RQ3: In order to better understand the extracted features, we propose an approach that combines machine learning and keyword extraction techniques to predict feature-related terms in Chapter 6. These feature-related terms are used to describe and indicate the notion of features. The results show that the feature-related terms can provide key information for recognizing the meaning of the extracted features.

RQ4: Although we propose approaches for extracting features and variation points from natural language documents in Chapter 4, Chapter 5 and Chapter 6, it is not clear whether these approaches can handle real-world requirements documents from a particular industry. To this end, in Chapter 7, we focus on applying, adjusting, and combining the proposed approaches to automatically extract domain knowledge from requirements specifications from Danfoss to assist domain engineers. We not only propose an improved approach to obtain the feature tree, but also develop a Graphical User Interface (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.

In Chapter 8, we summarize our contributions and discuss the potential directions for future research.

2. Background

In this chapter, we present the basic information needed to be known of in advance. Firstly, we introduce *Software Product Line Engineering*. Afterwards, we present the fundamental concepts regarding *Natural Language Processing*.

2.1 Software Product Line Engineering

Software Product Line (SPL) aims at developing a whole family of related but different software systems by enabling systematic reuse among these systems, called *variants*, and thus enable software customization at large scale. To this end, Software Product Line Engineering (SPLE) has been proposed as a specific development model, consisting of two parts [PBL05]: *domain engineering*, where the domain model is created and reusable artifacts are created (e.g., source code, user cases, requirements and so on); and *application engineering*, where based on a configuration the domain artifacts are composed and complemented in order to derive a concrete variant (e.g., an executable software system). Figure 2.1 presents an overview of domain engineering and application engineering, and we introduce them in detail in the following sub-sections.

2.1.1 Domain Engineering

In the process of domain engineering, the domain of a software product line is first analyzed by domain engineers, and then the reusable artifacts are developed [ABKS13]. The input of domain engineering is the related domain knowledge, and the outputs of it are the reusable artifacts that can be further configured to derive different products or applications of a software product line. Hence, domain engineering aims at the development for reuse [LSR07]. It contains three key goals:

- The scope of the software product line should be outlined by analyzing the collection of domain knowledge.

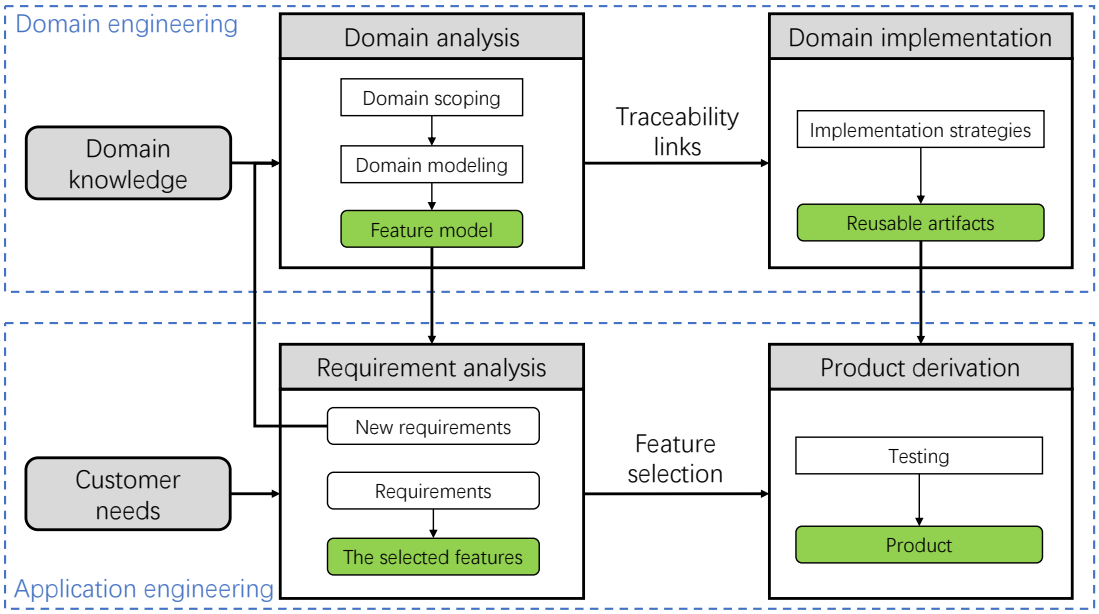


Figure 2.1: An Overview of software product line engineering.

- The commonality and the variability of the software product line should be identified, and all the information is recorded systematically to form a feature model (cf. Section 2.1.3).
- In order to achieve the desired variability, the reusable artifacts should be defined and developed.

In order to achieve the goals above, domain engineering mainly consists of two parts: domain analysis and domain implementation.

Domain analysis

Typically, domain analysis is a process of systematically analyzing the commonalities and differences of related software systems based on the study of all the relevant knowledge of a particular domain collected by domain experts [KCH+90], resulting in a feature model to document and present the information about commonalities and variabilities. In this process, the features that correspond to a particular product line are determined. It first includes a domain scoping step that determines which features should be supported by the software product line and implemented as reusable artifacts. Then, domain engineers identify the relations between features, usually resulting in a feature model presenting features and their relationships. Hence, it comprises two basic tasks: domain scoping and domain modeling.

Domain scoping. In the process of domain scoping, the range of a product line is defined, that is, domain experts decide what should be included and what should be excluded in the software product line based on the goals of the company developing a software product line [PBL05, ABKS13]. The scope describes all the common and variable features that are desired for future products. In particular, taking into account the evolution of the market demand and technology, some functionalities and

related standards may also alter for future applications. Hence, domain experts need to foresee the potential alteration when scoping the domain of the software product line. Collecting the information regarding the target domain is an indispensable task during domain scoping. Typically, the domain experts analyze existing products, competitors' products, handbooks, potential customers and so forth in order to achieve enough domain knowledge [ABKS13].

Domain modeling. In the process of domain modeling, domain engineers identify the information about commonalities and variabilities (i.e., differences) between desired products, and establish a feature model to present all the information in terms of features and the relations and constraints between features. Commonalities constitute the basis of every product from a software product line. Usually, if the number of commonalities are higher than the number of variabilities, it means that there is less effort needed in design for flexibility. The commonalities are usually identified by exploring the common requirements and functionalities that will appear in all the future products of a particular software product line. That is to say, the common requirements and functionalities can be regarded as the proper candidates for commonalities. Variability analysis aims at extracting and defining the variation points by anticipating the potential variants and analyzing different requirements. In particular, requirements from different customers' demands or from the needs of supporting different systems indicate the necessity that variation points and variants should be introduced. Nevertheless, it does not mean that the variation points should be defined for all the differences. In this process, the stakeholders are involved in carefully considering which variation points need to be introduced, since the constituent of the variation points affects the core structure and design of the feature model as well as the products in the software product line. In order to detect all the necessary variability information, domain experts need to achieve enough domain knowledge to analyze what the potential requirements and foreseen functionalities will occur in all desired products of the software product line. For example, they have to analyze which requirements present different functionalities in different products and which requirements only appear in a subset of the products. Finally, all this information about commonalities and variabilities is systematically documented and usually to be used to construct a corresponding feature model (cf. Section 2.1.3).

Although domain analysis can be considered as a type of requirements engineering, it is conducted for an entire software product line [ABKS13]. Hence, there exist obvious differences between domain analysis for the software product line and requirements engineering for single systems:

- Domain experts analyze not only the requirements from a specific customer but also the collection of other existing products, competitors' products, potential customers. This way, the analysis can reveal the common requirements that occur in all the foreseen products of the software product line and which requirements are unique for some products.
- Meanwhile, the prospective changes in requirements should be foreseen, for instance, market demands, technology and standards.
- Finally, all the commonalities and variabilities extracted from requirements are clearly documented in a feature model.

Domain implementation

After achieving the features, the reusable artifacts corresponding to the features identified in the domain analysis can be implemented, which is called domain implementation. The artifacts that can be reused and related to the software product line are various, comprising design, test, documentation and source code. Hence, we can obtain the traceability links between features in the feature model and implementation artifacts.

Domain implementation usually starts with the selection of the implementation strategies, such as language-based variability mechanisms (e.g., parameters, frameworks) and tool-driven variability mechanisms (e.g., preprocessors). Subsequently, in terms of the selected implementation strategy, we need to figure out how to design and implement the common parts and variation points. Three key differences between domain implementation and the single system implementation can be concluded as follows:

- Domain implementation aims at designing and implementing reusable artifacts in different contexts of a particular domain.
- In order to realize the variability, the configuration mechanism is involved in domain implementation.
- Hence, after domain implementation, what we can achieve is the configurable artifacts instead of a product.

2.1.2 Application Engineering

According to the demands of a particular customer, application engineering aims at developing a specific software product line application [ABKS13]. Compared with single application development in traditional software engineering, application engineering can benefit from the reusable artifacts developed in the process of domain engineering. Hence, the goal of application engineering is development with reuse [LSR07]. Here, we introduce the two main tasks in application engineering: requirements analysis and product derivation.

Requirements analysis

Requirements analysis aims at investigating and analyzing the requirements of a specific customer, which seems similar to requirement analysis in traditional software development. However, the key *difference* is that we have already learned domain knowledge during domain analysis, that is, some potential requirements have already been identified and documented in a feature model. Since the corresponding feature model has been already built during the domain analysis process, analysts can map the customer's requirements to existing features. In addition, it is likely that domain analysis might not identify all features that satisfy the specific requirements of a customer. If the analysts find new requirements that do not exist in the current feature model, they can feed the new requirements back into the domain analysis, which may result in modification of the feature model as well as the corresponding implementation artifacts.

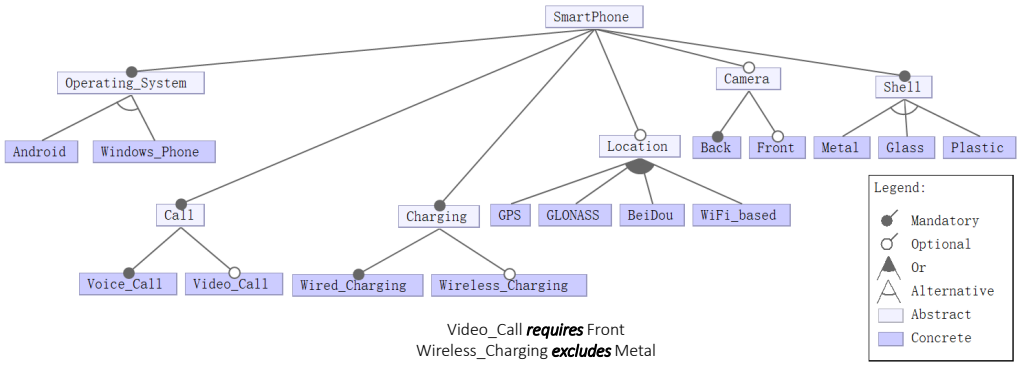


Figure 2.2: An exemplary feature model of smartphone.

Product derivation

After understanding the customer's requirements and selecting the related features, the implementation artifacts are combined to derive the desired product, which is named product derivation. Depending on the selected implementation approach (e.g., parameters, frameworks, preprocessors, etc.), this process can be more or less automated in terms of the selected features and reusable artifacts. Moreover, the activities of testing is necessary to validate and verify the product to meet the specifications before delivering the resulting product to a customer. The key *difference* between product derivation and traditional single system implementation is that the majority of the artifacts are derived from domain implementation artifacts in the process of product derivation rather than being created from scratch.

2.1.3 Feature Model

As part of this SPLE process, a *Feature Model*, as exemplarily shown in Figure 2.2, is commonly used to specify commonalities and differences between the related systems, usually expressed by means of *features* and their relations (i.e, constraints and dependencies) [ABKS13]. A feature usually denotes a unit of functionality of a software system [AK09], namely, it is an end-user visible characteristic of a software system [KCH+90]. Features play an indispensable role in a feature model, since features can specify the commonalities and variabilities of products throughout the entire software life cycle and each feature can meet a requirement and provide a potential choice for configuration. By means of this feature model, variants can be derived by (de)-selecting features that correspond to certain requirements, where the feature selection must satisfy the feature dependencies presented in the corresponding feature model. Hence, feature modeling is a pivotal step in SPLE as part of the domain analysis, in which domain engineers manually analyze the requirements to identify the commonalities and variabilities (i.e., differences) between products in a domain.

Figure 2.2 shows an example of a feature model. If a parent feature is included in a variant, the following types of parental relationships between parent-child features exist:

- *Mandatory* - the sub-feature must be included in each config;

- *Optional* - the sub-feature is possible to be included;
- *OR* - one or more sub-features can be included;
- *Alternative (XOR)* - exactly one sub-feature must be included.

In addition to the above relationships between parent-child features, *cross-tree constraints (CTCs)* express dependencies between features across the whole feature model. The most common are:

- *A requires B* - The presence of feature A implies the presence of feature B;
- *A excludes B* - The presence of feature A implies the absence of feature B.

2.1.4 Gap between SPL and Traditional Software Reuse

Software product lines aim at systematic software reuse. Although the development of a software product line is somewhat similar to the traditional software reuse, software product lines are dedicated to more complicated, detailed, and organized software reuse than traditional approaches. For example, *Clone and Own (CAO)* is a software reuse method. The similarity between CAO and the software product line is that CAO is also applied to a family of related software products. However, their difference is that CAO is more focused on a single product rather than a family of products, which means that information about the commonality and variability of a family of related products has not been systematically retained. Usually, companies build repositories in the process of developing different software products, and the repositories contain some reusable components, modules, and algorithms. Therefore, if CAO is used to develop a new product, the developers only need to find the one most relevant to the new product from the other developed products, reuse all reusable artifacts, adjust and add new functionalities. We can see that CAO can greatly improve development efficiency because developers can start the development with an already existing product. However, it still has three main shortcomings:

- In the process of using CAO, the development and reuse of software products are focused on a particular product, and the commonalities and variabilities of other related products are not taken into account. This shortcoming results in satisfying the demand of customization is a time-consuming, costly and labor-intensive process.
- CAO is mainly used for source code reuse without the ability to reuse other non-code artifacts (e.g. design documents, requirements, use cases, class diagrams, etc.). However, in current software system development process, a large part of the software systems is composed of non-code artifacts, and developers also need to spend considerable effort on non-code artifacts [BLB+14].
- Using CAO approach makes the maintenance of products costly. This is because although the cloned products belong to a product family, there is no direct connection between these products, which results in each product being maintained separately.

In contrast with CAO, SPLE can support the development of all core assets for reuse in a particular domain, which means that it is not necessary to spend considerable efforts to maintain each individual product but to maintain the core assets in a unified manner, since all the products in the same domain share the core assets. Moreover, in this process, the information about commonalities and variabilities is systematically recorded, which makes increasing demand for customization easier to achieve.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a way for computers to analyze, understand, and derive meaning from human language. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, or relationship extraction. Hence, NLP is very indispensable for extracting features and relations from requirements written in natural language. The basic ideas and techniques of NLP used in this thesis are introduced as follows.

2.2.1 Preprocessing

Disregarding the concrete NLP tasks, the indispensable step to initialize the task is to preprocess the input data in order to make them applicable to any subsequent NLP technique. Basically, four parts can be involved: cleaning, tokenization, annotation, normalization.

Cleaning

The original texts usually comprise various types of data, such as words, punctuation, symbols, etc, while not all the data is helpful for a particular task. In order to facilitate further analysis, we can just keep the key information of the texts. Some representative operations are as follows:

Stop words removal. Stop words are usually the most commonly used words in a particular language, such as “the”, “a”, “to”, and so forth. Moreover, any word without enough semantic information to describe a topic in a particular NLP task can be regarded as a stop word. Thus, for different NLP tasks, stop words are usually different.

Lower case. Letters and words are often written in two different types (i.e., upper case and lower case). For example, the letter at the beginning of the sentences is capitalized. Usually, the words in lower case are regarded as the standard form in order to simplify the analysis process.

Punctuation removal. For some specific NLP tasks, punctuation in the texts may be useless. In terms of purposes of different tasks, either all or part of the punctuation can be removed.

Tokenization

Tokenization is generally used in two ways: 1) splitting a text into sentences; 2) splitting a sentence into words, phrases, symbols, or other meaningful elements. An example of tokenizing a sentence is as follows:

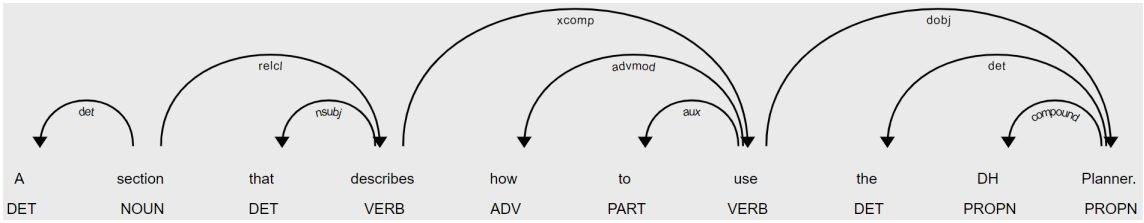


Figure 2.3: An example of a parsed sentence with POS tags and dependencies labels.

Sentence:

"When the alarm system is activated, this is indicated by the light of the LED."

Tokens:

"When", "the", "alarm", "system", "is", "activated", ",", "this", "is", "indicated", "by", "the", "light", "of", "the", "LED", "."

According to the example above, the original sentence is transformed into a list of tokens that can be seen as meaningful units for further analysis process. Moreover, tokens are very significant for NLP, since the majority of NLP tasks are conducted based on the tokens.

Annotation

Annotation is an action to assign a meaningful tag to each word in a sentence, while the tag presents a certain kind of linguistic information of the word. The tag as a sort of a priori knowledge helps the machines to process and understand natural language.

Part-of-Speech tagging. The most popular annotation for preprocessing is Part-of-Speech tagging (POS tagging). By using POS tagging, the part of speech of each word or token can be assigned, for example, noun, verb, adjective, etc. In Figure 2.3, the line under the sentence presents the POS tags of each word.

Dependency parsing. Except for POS, the syntactical information should not be neglected. Dependency parsing plays an important role in achieving the grammatical structure and the relations between words. The information above the sentence in Figure 2.3 shows the corresponding syntactic dependency information of sentence. For example, "nsubj" stands for a nominal subject that is the syntactic subject of a clause and "dobj" denotes the direct object of a verb phrase [MM08].

Normalization

A word can be changed into different forms in terms of the way of being used, such as presenting different tenses (i.e., the past, present and future tense). However, different forms of a word (i.e., the inflected words) may increase the complexity of processing a sentence for some specific NLP tasks, for example, the tasks that can ignore the tense information. In NLP, normalization is usually applied to gain a certain standard form of words in order to simplify the analyzing process.

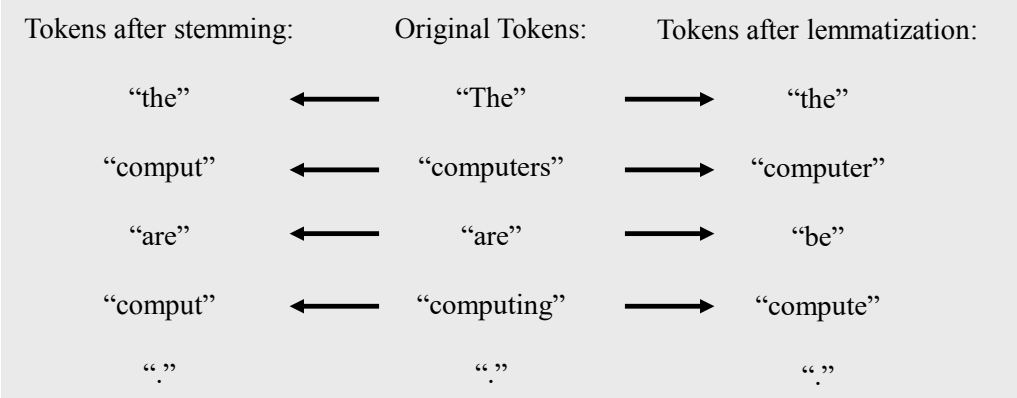


Figure 2.4: An example of tokens after conducting stemming and lemmatization.

Stemming. The inflected words are able to be reduced to their word stems by cutting off the common prefixes and suffixes. It seems that stemming cuts off a word in terms of how it looks into its root form, even if there is no dictionary meaning for this root form.

Lemmatization. The process of resolving the inflected words to their dictionary forms (i.e., lemma) is called Lemmatization, which takes the meaning of the word in the sentence into account.

In Figure 2.4, we use the tokens from an exemplary sentence (i.e., “The computers are computing.”) to present the difference between stemming and lemmatization operation.

After preprocessing, there are various further processing steps for different NLP tasks. We mainly introduce the related techniques for feature and variability extraction from requirements in the following sections.

2.2.2 Word Embedding

In order to compute the semantic similarity of words, capture the context of words in a document and obtain relations among words, the common way is to convert words into vectors (i.e., embeddings). The vector representations of words are commonly achieved in two different ways: traditional distributional semantic models (DSMs) and neural word embedding.

Traditional DSMs

Traditional DSMs can be considered as “count models”, since they operate on co-occurrence matrices to initialize the vector representations of words, such as counting co-occurrences of words appearing in a specific corpus. Vector Space Model (VSM) and Latent Semantic Analysis (LSA) are the common traditional DSMs applied in the research area for feature extraction from requirements [KTWF12, ASB+08, WCR09].

The foundation of VSM is simple, which simplifies the processing of text content to vector operations in vector space, and uses spatial similarity (i.e., the similarity

of vectors) to express semantic similarity of texts, which is intuitive and easy to understand [SWY75]. When a document is represented as a vector, the similarity between documents can be measured by calculating the similarity between the vectors. The most commonly used measure of similarity in text processing is the cosine distance. We briefly introduce how to gain the vector representation of the requirements. Given a set of requirements R and a dictionary D, a requirement is represented as the bag of its words (i.e., the bag-of-words model), and then a value is calculated for each word according to the TF-IDF. Since the size of dictionary D is m, the requirement is converted into an m-dimensional vector. If a word in the dictionary does not appear in a particular requirement, the corresponding element of the word in the vector is 0. If a word appears in the requirement, the corresponding element value of the word in the vector is the TF-IDF value of the word. In this way, the requirement is represented as a vector, and this is the vector space model. Put it differently, the vector space model does not catch the relationship between terms, since it assumes that terms are independent of each other.

LSA also presents the text as a document-term matrix and calculates the similarity between documents by vectors (such as the angle), which is the same as VSM. The difference is that LSA assumes that some latent structures exist in word usage to express a topic, but these latent structures may be partially obscured by the variety of word selections. Hence, the LSA utilizes singular value decomposition to the document-term matrix and takes the first k largest singular values and corresponding singular vectors to form a new matrix [Dum04]. The new matrix reduces the ambiguity of the semantic relationship between terms and text, which is beneficial to figure out the meaning of text.

Neural word embedding

Neural word embedding is a neural-network-based natural language processing architecture which can be seen as prediction models, since the vector representations of words or texts can be gained from a pre-trained language model trained on large text collections. There are also various techniques supporting achieving accurate neural word embedding models, such as word2vec [MSC+13], GloVe [PSM14] and FastText [BGJM16].

Word2vec applies a two-layer neural network to train a large size of corpus, which results in a vector space where the words in the corpus are transformed into vector representation. Its basic idea is that two words with similar context (i.e., surrounding words) should have similar word vectors. According to the difference of using the context, word2vec provides two methods to achieve the representations of languages: the Continuous Bag-of-Words (CBOW) and Skip-gram models. As shown in Figure 2.5, the CBOW predicts the target word based on the context, while given the word, Skip-gram predicts the target context.

Word2vec is trained on the local corpus (i.e., the surrounding words of a word), and its text characteristic extraction is based on sliding windows, while GloVe's sliding windows are used to build a co-occurance matrix, which is based on global corpus. It can be seen that GloVe needs to count the co-occurrence probability in advance. FastText treats each word as a bag of n-grams rather than the word itself. For example, the n-gram representation of word "apple" is "<ap", "app", "ppl", "ple",

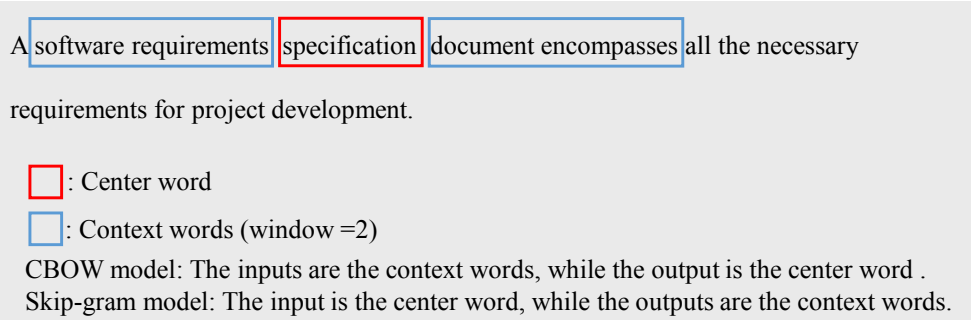


Figure 2.5: An example for sketching the CBOW and Skip-gram models.

"le>" with boundary symbols "<" and ">", if we assume that n is three. Therefore, we can use these tri-grams to represent the word "apple". Subsequently, the sum of these five tri-grams vectors can be used to represent the word vector of "apple". Furthermore, FastText can learn the vector of the character n-grams in a word and sum these vectors to generate the final vector of the word itself, thereby generating a vector for a word that does not appear in the training corpus.

Since the semantic similarities of words, sentences or texts are obtained in terms of a numeric representation learned from the semantic information of large text collections, it is also called *corpus-based similarity*. In our case, the corpora can be any text collections regarding the products or systems in a specific domain, such as requirements specifications.

2.2.3 Recognizing Textual Entailment

Recognizing Textual Entailment is a very challenging task of determining whether a natural language snippet can be inferred from another natural language snippet, which is an essential problem in natural language understanding that needs the capabilities to extract and analyze both syntactic and semantic information in natural language.

Table 2.1: The exemplary pairs for textual entailment.

Premise	Hypothesis	Gold Label
A man is walking away from tents with the word Camden on them.	The man is walking.	entailment
A man is walking away from tents with the word Camden on them.	The tents do not have anything written on them.	contradiction
A man is walking away from tents with the word Camden on them.	The man works for the campy supply company "Camden".	neutral

Textual entailment is defined as a directional relationship between pairs of statements, denoted by "Premise" (P) and "Hypothesis" (H) [DRSZ13]. Given these two

statements P and H, and a human reading and comprehending them, the relationship between them could be one of the following:

- Entailment: If P provides explicit information that can be used to infer that H is most likely true, we can say that P entails H.
- Contradiction: If P provides explicit information that can be used to infer that H is most likely false, we can say that P contradicts H.
- Neutral: If P can not provide explicit information that can be used to infer whether H is true or false, we can say that P is unrelated to H.

Table 2.1 shows the three example pairs with labels for representing relations, taken from the Stanford Natural Language Inference (SNLI) corpus [BAPM15].

2.3 Summary

In this chapter, we introduced the prerequisite knowledge for reading this thesis. Our research is conducted in the context of software product line which allows you to develop a family of systems that share common functionality but are still not identical with less effort by systematically reusing the artifacts of the systems. Feature model is widely used in the process of software product line engineering to be a central place to hold not only the information of commonality and variability but also the information of dependency. However, creating a feature model for a legacy system in the process of domain engineering from scratch needs upfront investment, especially taking a large legacy system and engineers lacking enough domain knowledge of this legacy system into account. To address this, we proposed to analyze the requirements documents of the legacy systems to extract features and variation points. The extracted information is expected to be a starting point for domain engineers to create a feature model in an efficient way. Hence, natural language processing is an indispensable topic in this thesis, and we also introduced the main natural language processing techniques that we used in the following chapters.

3. Current Research on Feature and Variability Extraction

We presented a systematic literature review on feature and variability information extraction from natural language documents at SPLC 2017 [LSS17]. In this chapter, we extend the systematic literature review and present more details on current research in reverse engineering variability from natural language documents.

Identifying features and their relations (i.e., variation points) is crucial in the process of migrating single software systems to software product lines (SPLs). Various approaches have been proposed to perform feature extraction automatically from different artifacts, for instance, feature location in legacy code. Usually such approaches a) omit variability information and b) rely on artifacts that reside in advanced phases of the development process, thus, being only of limited usefulness in the context of SPL. In contrast, feature and variability extraction from natural language documents is more favorable, because a mapping to several other artifacts is usually established from the very beginning.

In this chapter, we investigate the current state-of-the-art of feature and variability extraction from natural language documents by means of a comprehensive literature survey. To this end, we not only focus on existing techniques that have been adopted for SPL, but also on the maturity of these approaches, that is, to what extent they could be applicable in practice. In particular, we make the following contributions in order to answer RQ1:

- A comprehensive study of existing reverse engineering approaches, used input formats, employed NLP techniques and further algorithms for feature & variability extraction.
- A qualitative analysis regarding multiple criteria, such as accuracy, completeness, and their evaluation, which allow to compare the reviewed approaches at a reasonable level of detail.

- We provide key observations, identified within our detailed comparison, and derive shortcomings and challenges that are beneficial to identify future research directions.

3.1 Review Methodology

A Systematic Literature Review (SLR) is an accepted method for evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest [Kit07]. In particular, we apply the proposed guidelines by Kitchenham et al. [Kit07] in order to identify, classify, compare, and evaluate existing techniques for reverse engineering variability from natural language documents. In this section, we provide information about all steps we performed for planning the review.

3.1.1 Need for a Review

In recent years, the application of NLP techniques for aiding the software engineering process by reverse engineering information from several artefacts has been increased considerably. Our literature review aims to complement recent efforts by providing an overview of how NLP techniques are used to infer features from natural language documents [BKS15]. In particular, we extend the review by Bakar et al. [BKS15] by a) focusing also on how variability information is extracted and b) by a comprehensive qualitative evaluation of the approaches regarding aspects such as accuracy, automation, and tool support. As a contribution, we provide detailed insights for both researchers and practitioners, in the current state and maturity of extracting detailed variability information from natural language documents. Moreover, we not only highlight promising approaches, but also identify gaps and formulate derived challenges that have to be addressed in the future, thus, paving the way for more efforts in these research directions.

3.1.2 Research Questions

The focus of this SLR is to identify, compare, and evaluate existing approaches for feature and variability extraction from natural language documents. In order to answer RQ1 mentioned in Chapter 1, we formulate three sub-research questions based on the PICOC method (Population, Intervention, Comparison, Outcome, and Context) [HG11] and present the respective criteria in Table 3.1.

While our overall question targets the applicability of current approaches in practice, we guide our systematic literature review in terms of the following concrete research questions:

RQ1.1: *What approaches of feature and variability extraction from natural language documents have been proposed for SPL?*

With this question, we aim at summarizing all relevant techniques that contribute to the goal of our literature review. Although feature extraction is also subject to research in software engineering, we are mainly interested in approaches focusing on

Table 3.1: Research questions structured by PICOC criteria.

PICOC	Description
Population	Literature in reverse engineering variability from natural language documents.
Intervention	Mechanisms, i.e., techniques, methods, tools, approaches that realize such a reverse engineering process.
Comparison	Techniques together with their performance, evaluation and tool support proposed by each primary study.
Outcome	Several observations regarding applicability and quality of current approaches and major gaps and open challenges in this field.
Context	The SPLE process, in particular the reverse engineering step to enable a systematic product-line development (e.g., in an extractive way).

SPL and how these approaches tackle the challenge of extracting variability information. Finally, we aim at identifying which kind of natural language documents have been used as input for feature extraction.

RQ1.2: *How are the techniques supported regarding tools and automation?*

We are interested, whether the techniques, obtained through RQ1, are supported by robust tools or just implemented as prototypes. This is of special importance in the context of applicability in practice, such as in real-world systems or industry. Similarly, we evaluate to what extent the process of extracting features is automated.

RQ1.3: *How reliable are the approaches proposed for feature extraction in SPL?*

With this question, we focus on the quality of the proposed approaches. In particular, we are interested in two aspects. First, the completeness, that is, to what extent do the approaches also extract variability, thus, providing information for creating a complete picture of the SPL (e.g., by means of a feature model). Second, the accuracy, that is, how precisely do the proposed approaches extract features (and variability) from natural language documents. As a result of this research question, we analyze to what extent the proposed approaches can be applied in practice or need to be revised and evaluated more thoroughly. Moreover, we derive limitations and open challenges from answering these research questions.

3.1.3 Search Strategy

To identify relevant literature and extract the important information, we set up a search strategy that consists of three steps. First, we specify scientific databases to search for our initial set of candidate papers. In particular, we search in the databases of ACM Digital Library, IEEE Xplore, SpringerLink, ScienceDirect, Scopus, dblp, and Google Scholar for studies published in journals, conferences, and workshops between the year 2000 and 2019. We have chosen these libraries as they are renowned scientific databases that index the most important publications in

the field, such as from ACM, IEEE, or Elsevier. As a second step, we implement a review process to exclude duplicate studies or studies that are not relevant for other reasons (cf. Section 3.1.4). Afterwards, as a third step, we apply *snowballing* to complement the initial search [WRH+12]. In particular, we analyze references and citations of retrieved studies and secondary studies (i.e., existing surveys), thus, identifying relevant literature not covered by the aforementioned databases. Finally, we merge the results from our initial searching and snowballing to obtain the final set of *primary studies*.

3.1.4 Conducting the Review

Basically, we conduct our systematic review based on the protocol defined in the previous subsection. For instantiating this protocol, we have to take concrete actions as follows; *(i)* define the concrete search term, *(ii)* define inclusion and exclusion criteria for identifying relevant literature, and *(iii)* specify a concrete and systematic process for extracting the data needed to answer our RQs.

Search Criteria: To construct our search string, we derived keywords from our research questions based on the population, intervention, and outcome. Additionally, we checked for possible synonyms, related terms and alternative spellings. Finally, we used boolean logic; an "OR" to combine alternative terms/spellings and an "AND" to connect the major terms in our string. The resulting search string is as follows:

("feature extraction" OR "feature identification" OR "feature mining" OR "feature detection" OR "variability extraction" OR "variability identification" OR "variability mining" OR "variability detection") AND ("natural language" OR "requirement specification" OR "textual requirement" OR "product specification" OR "product description" OR "product review") AND ("software product lines" OR "product family" OR "software family" OR "feature-oriented software development")

Our search string comprises three parts: 1) the extraction and analysis of the particular information or artifacts; 2) the types of documents analyzed; and 3) the context of the research being surveyed. In spite of the fact that all the relevant keywords within each part are of similar meanings, they differ somewhat. For the first part, "feature extraction" is the most frequent term in this topic used to describe the process of extracting features from natural language documents. Although the search terms such as "feature identification", "feature mining" and "feature detection" are all used as synonyms for feature extraction, "feature detection" is highly relevant to detect dead features. We regard "feature detection" as one of the search terms, since feature extraction related techniques may also be applied in the process of detecting dead features by means of analyzing requirements. There also exist several search terms that seem to be related to the topic of feature extraction but are not included in the search string nevertheless, for example, "feature selection" and "feature location". "Feature selection" usually means selecting a good feature set to achieve customer requirements, which focuses on the problem of optimization, while "feature location" predominantly concentrates on locating features in source code. We find

that although the papers retrieved by using these terms may also contain the analysis of the textual comments in the source code, it lacks a systematic method that focuses on extracting features from requirements, thereby resulting in lots of useless search results. Likewise, for variability extraction, we use the same wording as the search terms of feature extraction. For the second part, "natural language" is used to describe the types of the documents. And, "requirement specification", "textual requirement" and "product specification" are all applied as synonyms for a detailed and formal description of a system or product to be developed with its functional and non-functional requirements. "Product description" can be regarded as a kind of informal description of a product that can be collected on the internet, while "product review" denotes the review of a particular product from their customers. Obviously, for the third part, the search terms that are either related or similar to each other are used for finding the papers that are focused on research on software product lines. Understanding the small difference among these keywords, the study selection can be conducted more efficiently and comprehensively.

Inclusion and Exclusion Criteria: We created a set of *inclusion (IC) and exclusion (EC)* criteria to identify potential primary studies. While initially intended to be applied on the title and abstract, it turned out that this is insufficient for most of the papers to decide on their relevance. Hence, we also scanned introduction and conclusion to make a more reliable decision. The inclusion criteria we finally created, based on the analysis scope, are as follows:

IC01 Articles matching the search string mentioned above and within the scope of our analysis, i.e., they propose a technique or mechanism for feature & variability extraction from natural language documents.

IC02 Articles published between January 1st 2000 to December 31st 2019, since research on automatic feature & variability extraction from natural language documents in SPL began in 21st century.

Moreover, we consider studies irrelevant if they meet at least one of the following exclusion criteria:

EC01 Articles not focusing on feature and variability extraction from natural language documents in SPL, i.e., feature extraction from legacy code, approaches improving feature modeling, functional requirements extraction, etc;

EC02 Articles not written in English;

EC03 Articles not belonging to research papers, i.e., proposals, summaries of conferences, lecture notes, etc;

EC04 Articles not pertaining to firsthand research, namely, related literature review or survey papers.

Data Extraction: First of all, we applied our search string to scientific databases. Afterwards, we applied inclusion and exclusion criteria to the result. Along with this process, we extracted and stored the following information in a spreadsheet:

- Date of search, scientific database, study identifier
- Publication information, author, title, publication year, source, publication type (Journal/Conference)
- Inclusion criteria *IC01–IC02* (and which one applies)
- Exclusion criteria *EC01–EC04* (and which one applies)

The author of this thesis initially applied the data extraction process. For all excluded papers, a brief reason was specified and double-checked by the second author of the original paper [LSS17]. In the case of non-agreement, the paper has been discussed by both of us to find a consensus. Once we finally decided on the primary studies, we performed a further retrieval by applying the snowballing method [WRH+12]. In particular, we took all primary studies and papers excluded due to EC04 into account. For papers, found by snowballing, the same extraction process as above was applied and if we decided on their relevance, it was added to the list of primary studies.

For the final list of primary studies, we read the full paper and captured and extracted additional data, required to perform our analysis and answer our research questions. In particular, we extracted the following data:

- *Input* and *Output* of the corresponding approach.
- *Methodology*, i.e., which NLP techniques are used and which further techniques are possibly applied in a post-processing step. Moreover, we noted the degree of automation and how much of the variability (i.e., the relation between features) can be recovered by a particular approach.
- *Evaluation:* to what extent a particular approach has been evaluated and what is the result of the evaluation, especially regarding accuracy
- *Tools:* whether there is any tool available, implementing the proposed approach and NLP techniques.

3.2 Results

In this section, we present the results of our literature review. First, we briefly report on the results of our systematic literature search, described in Section 3.1. Then, we provide detailed answers to our formulated research questions.

3.2.1 Results of Studies Search

Table 3.2: Overview of all reviewed papers, ordered by year of appearance.

ID	Title	Year	Input	Acc	Output
P01	An Approach to Constructing Feature Models Based on Requirements Clustering [CZZM05]	2005	SRS	Yes	FM

P02	An Exploratory Study of Information Retrieval Techniques in Domain Analysis [ASB+08]	2008	SRS	No	FM
P03	A Framework for Constructing Semantically Composable Feature Models from Natural Language Requirements [WCR09]	2009	SRS	No	FM
P04	On-demand Feature Recommendations Derived from Mining Public Product Descriptions [DGH+11]	2011	PD	Yes	FM
P05	Supporting Commonality and Variability Analysis of Requirements and Structural Models [KTWF12]	2012	SRS	No	RTL
P06	On Extracting Feature Models From Product Descriptions [ACP+12]	2012	PD	Yes	FM
P07	Decision Support for the Software Product Line Domain Engineering Lifecycle [BEG12]	2012	PD	No	FM
P08	Mining Commonalities and Variabilities from Natural Language Documents [FSD13]	2013	PB	No	FL
P09	Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings [HCHM+13]	2013	PD	Yes	FM
P10	Mining and Recommending Software Features across Multiple Web Repositories [YWYL13]	2013	PD	Yes	FL
P11	Feature Model Extraction from Large Collections of Informal Product Descriptions [DDH+13]	2013	PD	Yes	FM
P12	A Systems Approach to Product Line Requirements Reuse [NSN+14]	2014	SRS	No	OVM
P13	Analyzing Variability of Software Product Lines Using Semantic and Ontological Considerations [RBIW14]	2014	SRS	No	SS
P14	Generating Feature Models from Requirements : Structural vs . Functional Perspectives [IRB14]	2014	SRS	Yes	FM
P15	Detecting Feature Duplication in Natural Language Specifications when Evolving Software Product Lines [KBA15]	2015	SRS	No	DF
P16	Improving the Management of Product Lines by Performing Domain Knowledge Extraction and Cross Product Line Analysis [RBWH15]	2015	FD	Yes	FM
P17	Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines [HW15]	2015	SRS	Yes	FM

P18	Semantic Information Extraction for Software Requirements using Semantic Role Labeling [Wan15]	2015	SRS	No	SI
P19	CMT and FDE: Tools to Bridge the Gap between Natural Language Documents and Feature Diagrams [FSGD15]	2015	PB	No	FM
P20	Automatic Semantic Analysis of Software Requirements Through Machine Learning and Ontology Approach [Wan16]	2016	SRS	No	SI
P21	Extracting Features from Online Software Reviews to Aid Requirements Reuse [BKSJ16]	2016	OR	Yes	FL
P22	Variability Analysis of Requirements: Considering Behavioral Differences and Reflecting Stakeholders' Perspectives [IRBW16]	2016	SRS	No	FM
P23	Mining Feature Models from Functional Requirements [MBBA16]	2016	SRS	No	FM
P24	Automated Extraction of Product Comparison Matrices from Informal Product Descriptions [NBA+17]	2017	PD	Yes	PCM
P25	Extracting Software Features from Online Reviews to Demonstrate Requirements Reuse in Software Engineering [BKSH17]	2017	OR	Yes	FL
P26	Ambiguity Defects as Variation Points in Requirements [FGS17]	2017	SRS	Yes	VI
P27	Extracting software product line feature models from natural language specifications [SKPC18]	2018	SRS	Yes	FM
P28	Requirement Engineering of Software Product Lines: Extracting Variability Using NLP [FFGS18]	2018	SRS	Yes	VP
P29	Behavior-Derived Variability Analysis: Mining Views for Comparison and Evaluation [RBSA19]	2019	SRS	Yes	VV
P30	Towards Complex Product Line Variability Modelling: Mining Relationships from Non-Boolean Descriptions [CHN19b]	2019	PD	Yes	FM
P31	Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions [CHN19a]	2019	PD	Yes	FM

Acc: Accessibility.

SRS: software requirements specifications; PD/PB: product description/brochure; OR: online software reviews; FD: feature diagrams.

FM: feature model; OVM: Orthogonal Variability Model; FL: feature list; RTL: recommendation traceability links; SS: SRS similarity; DF: duplicated features; SI: semantic information; PCM: product comparison matrices; VI: variability indicators; VP: variation points; VV: variability views.

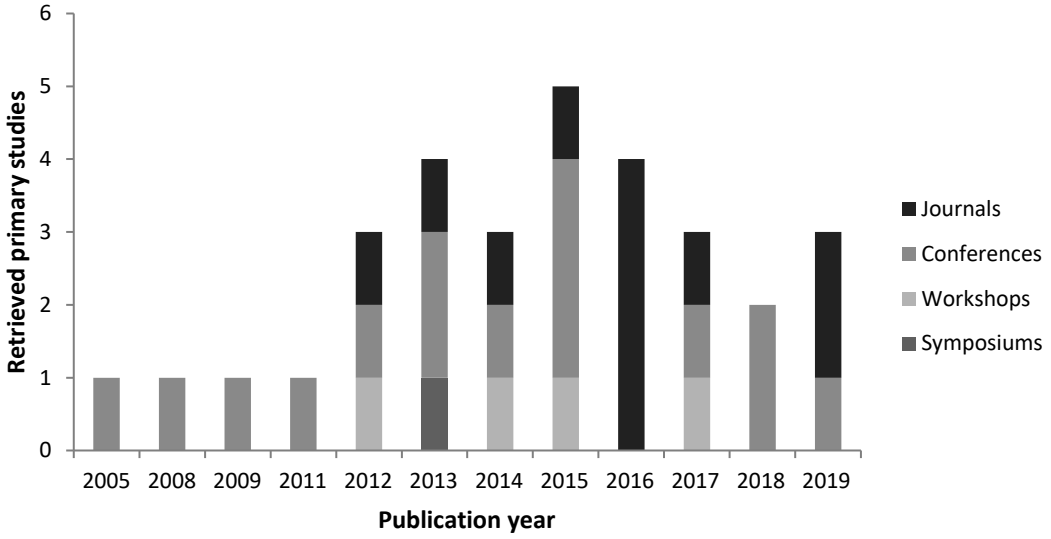


Figure 3.1: The distribution of primary studies from 2005 to 2019 by year from each venue: symposium, workshop, journal, conference.

In order to obtain the primary studies, we initially used the predefined search string on the selected databases mentioned (cf. Section 3.1.3). As a result, we retrieved an initial list of relevant studies comprising 251 papers (ACM: 43, IEEE: 6, SpringerLink: 94, ScienceDirect: 69, Scopus: 17, dblp: 22). Note that in this step, we already applied our inclusion criteria to the papers found, thus, all non-relevant papers have already been filtered out. Next, we applied our exclusion criteria to this initial list. After applying EC01, a majority of the papers could be discarded, as they propose relevant approaches, but on different artifacts than we are interested in. Further papers have been discarded, because they adhere to EC03 and EC04. Finally, we removed duplicated papers from our list, which eventually results in a list of 12 papers.

Based on these selected papers and on the papers excluded due to EC04 (i.e., constituting secondary studies), we then performed *snowballing* as an additional step to retrieve relevant papers we may have missed so far. In particular, we performed three iterations of *backward* (i.e., analyzing the reference list of selected papers) and *forward* (i.e., screening papers on Google Scholar that cite our selected papers) snowballing [WRH+12]. In the first iteration, we found 522 papers (backward:341, forward:181), from which we excluded 504 papers due to our exclusion criteria and removal of duplicates. Hence, 18 papers have been considered to be relevant, and thus, added to our list of primary studies. For these papers, we again applied the snowballing technique, resulting in 1280 papers, from which one paper remained after applying exclusion criteria and duplicate elimination. Finally, we also applied snowballing for the paper retrieved in the previous iteration, but from initially 58 potential papers all have been discarded (excluded or duplicated).

As a result, given our list of 12 papers from the initial search, we were able to identify 19 further papers with snowballing that we found worth being added to our primary studies. Hence, we found 31 papers as primary studies that are subject to further analysis in order to answer our research questions. Figure 3.1 presents that 19% of

the studies were found in workshops (4) and symposiums (1), whereas the majority ($\sim 81\%$) were published in conferences (15) and journals (11). Moreover, we provide an overview of all papers, together with the extracted information, in Table 3.2 and answer our research questions in the following.

3.2.2 Answering Research Questions

The main goal of this SLR is to provide detailed insights of techniques used, degree of maturity achieved, shortcomings, and challenges ahead for extracting feature and variability information from natural language documents. To this end, we formulated three research questions in Section 3.1.2, which we answer in the following.

RQ1.1: What approaches of feature extraction from natural language documents have been proposed for SPL?

With this research question, we shed light on techniques used for the extraction process and which kind of natural language documents are considered as input for the respective approaches.

Techniques used: Natural language processing techniques are widely used and indispensable for extracting features and variability information from natural language documents. Besides natural language processing techniques, various techniques of information retrieval and machine learning are considered to be helpful in this research direction. We briefly introduce how these techniques are generally used, with a specific focus on feature identification and variability extraction in SPL. The general process is shown in Figure 3.2.

As a first step, natural language documents are transformed into types of words that can be identified and analyzed easily by computers. This step is called *Text Pre-processing*. In particular, natural language documents are divided into words, phrases, symbols, or other meaningful elements (e.g., using tokenization). Additionally, these elements can be tagged with their type of word (e.g., noun, verb, object, etc.) using Part-of-Speech tagging (POS tagging). In addition, stop words which usually refer to the most common words in vocabulary (e.g., "the", "at") are removed in this phase, as they lack any linguistic information.

A second (optional) step is *Term weighting* which can be adopted to estimate the significance of terms in natural language documents by calculating the frequency of their occurrence in different natural language documents. For instance, Term Frequency-Inverse Document Frequency (TF-IDF) and C-NC value are two commonly used techniques. With TF-IDF, the term is considered important if it appears frequently in a document, but infrequently in other documents. C-NC value is a more sophisticated statistical measure that combines linguistic and statistical information.

Another optional step is *Semantic Analysis* that is typically used to gain semantic information. Several techniques can be employed in this step, for example, Vector Space Model (VSM) and Latent Semantic Analysis (LSA) are widely used to conduct a semantic analysis. With VSM, preprocessed documents are represented as

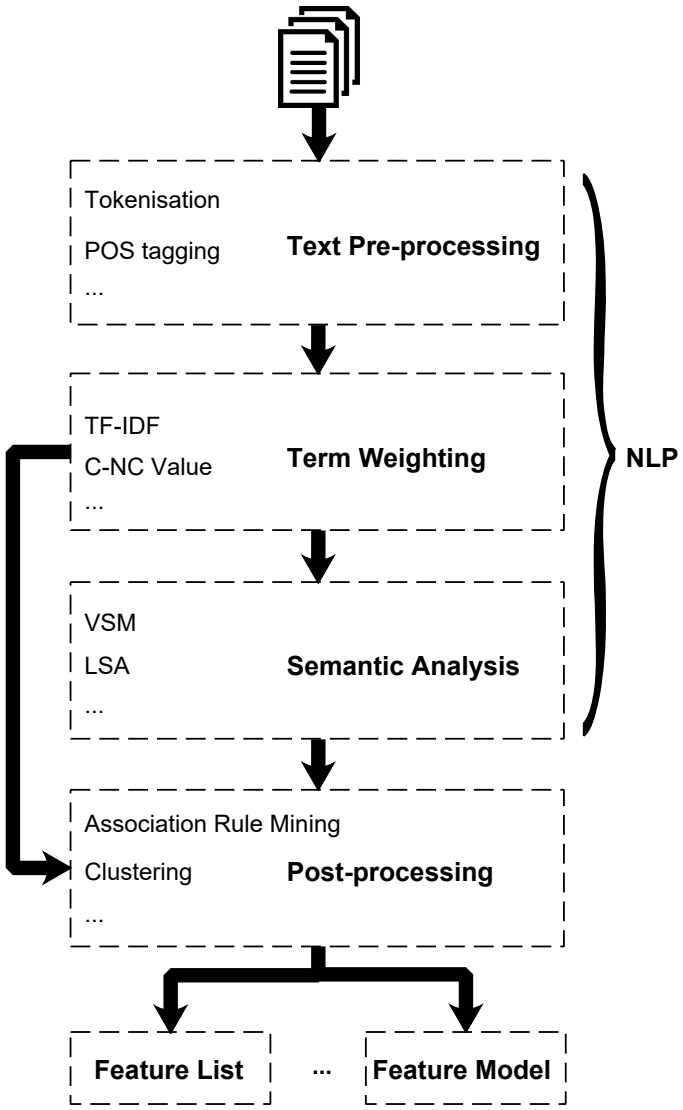


Figure 3.2: General process for applying techniques in reverse engineering variability from natural language documents.

vectors. Through calculating the cosine between the vectors, the similarity between documents can be determined. LSA utilizes a term-document matrix to analyze the similarity of documents and can be combined with Singular Value Decomposition to reduce the dimension of the textual documents.

Additionally to the NLP process, the transformed data can be further analyzed by a *post-processing* step in order to identify features and extract variability information. Certainly, various methods can be used in this step, such as Clustering Approaches and Association Rule Mining. Cluster approaches are adopted to group similar features with a feature being a cluster of tight-related requirements. Association rule mining is used to discover affinities among features across products, and to augment and enrich the initial product profile. After post-processing, various outputs can be obtained, such as a feature list or feature model.

Table 3.3: Overview of the NLP techniques used by the reviewed approaches.

NLP Techniques		ID
Text Pre-processing	Tokenisation	P07 P09 P12 P15 P18 P19 P20 P24 P25 P27
	Part of Speech Tagging	P02 P03 P05 P07 P08 P11 P12 P13 P14 P15 P17 P18 P19 P20 P21 P22 P24 P25 P27 P29
	Lemmatization	P18 P20 P21 P24 P25
	Stemming	P04 P09 P10 P11 P25
Term Weighting	TF-IDF	P04 P09 P10 P11 P13 P14 P22 P23 P25 P27 P29
	C-NC Value	P08 P19 P24
Semantic Analysis	Vector Space Model	P02 P05
	Latent Semantic Analysis	P02 P03 P13 P14 P21 P22 P25
	Contrastive Analysis	P08 P21 P24
	Syntactical Heuristics	P24
	Latent Dirichlet Allocation	P09 P10
	Semantic Role Labeling	P13 P14 P18 P20 P22 P29
	Semantic Model	P23
Lexical Database	WordNet	P07 P13 P14 P16 P20 P21 P22 P23 P29
	SemCor	P20
	PropBank	P18 P20
	FrameNet	P18 P20

First of all, our analysis reveals that all the phases, outlined in Figure 3.2, have been employed by the considered approaches. In Table 3.3, we provide a detailed overview of NLP techniques used by the particular approaches. For text pre-processing, the majority performs POS tagging ($\sim$ 65%) to decompose the documents in their building blocks, shown in Figure 3.3 (a). Moreover, tokenization was applied frequently ($\sim$ 32%), yielding to a similar result. Beyond preprocessing, Figure 3.3 (b) presents that term weighting with different techniques is quite common amongst the approaches ($\sim$ 45%). Our inspection reveals that this technique is mainly used to identify meaningful terms that resemble possible features. Meanwhile, TF-IDF is a very popular method to achieve this goal. An interesting observation we made, is that most approaches employ more than one technique, either from preprocessing only or in combination with term weighting.

Observation 1. *For extracting features and variability, a diverse selection of NLP techniques is employed, indicating that a) different approaches may lead to the desired result and b) applying multiple NLP techniques increases the quality of the result.*

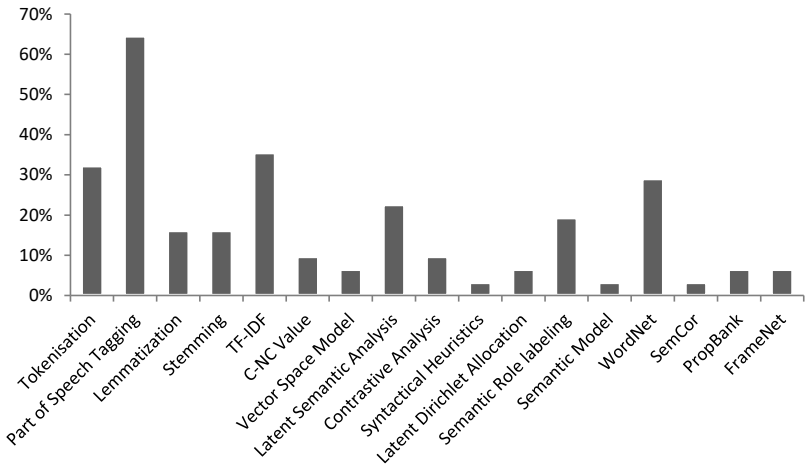
While the above-mentioned approaches focus mainly on syntactical aspects, many approaches also make use of the (optional) built-in mechanisms of NLP for semantic analysis ($\sim$ 52%). In particular, latent semantic analysis (LSA) and semantic role labeling (SRL) are preferred techniques, where the latter is combined with using a proper lexical database. If we take quality criteria such as accuracy or completeness into account (cf. Table 3.5), we observe a slight tendency that an additional semantic analysis improves the overall result. In addition, the usage of a lexical database ($\sim$ 32%) benefits automatically analyzing and understanding the natural language documents to extract features and variation points.

Observation 2. *Semantically understanding the natural language documents is a crucial aspect for successfully reverse engineering features and variability.*

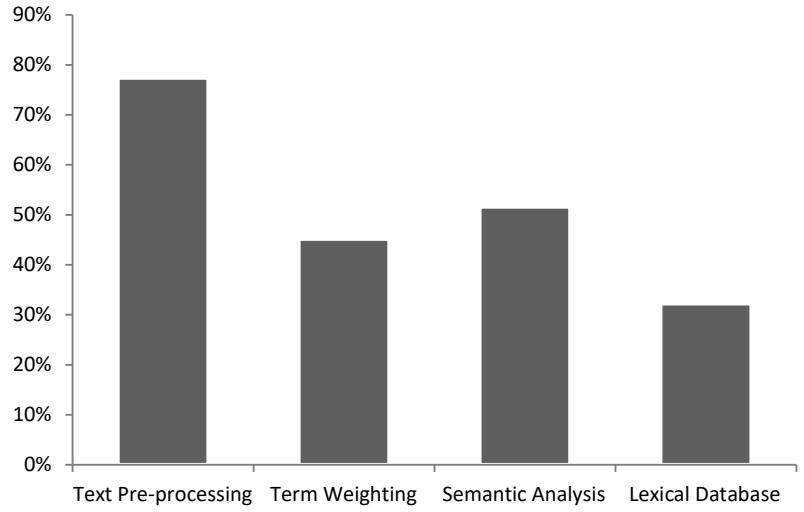
Finally, many approaches ($\sim$ 81%) perform an additional post-processing step on the top of previously mentioned NLP mechanisms (cf. Figure 3.2), while Table 3.4 shows a detailed overview of post-processing techniques used by the particular approaches. Our analysis reveals that especially clustering algorithms are preferably used in this stage of the extraction process ($\sim$ 45%). The reason is that clustering is able to find relations between the previously identified concepts or terms. Hence, such algorithms are especially beneficial to finding related and unrelated features, and thus, enable the search for groups of features and even other relationships among them such as hierarchies.

Observation 3. *Clustering algorithms are well-suited to complement the NLP process, as they facilitate the detection of the specific relationships between features such as groups or parent-child relations.*

Besides clustering, a variety of other techniques are employed for post-processing ($\sim$ 61%), mainly from the domain of heuristics and machine learning. The particular algorithms we identified are especially able to learn specific patterns in the



(a)



(b)

Figure 3.3: Histograms of the NLP techniques used by the reviewed approaches.

preprocessed and enriched data from the NLP process. In this way, even complex dependencies can be inferred, which is hardly possible with NLP mechanisms only.
Types of documents: As natural language documents may occur in various forms, we are also curious about which kind of documents are subject to feature/variability extraction. According to Table 3.2, our review indicates that software requirements specifications (SRS) are used predominantly ($> 54\%$) as input, usually based on standards such as IEEE-STD-830. Interestingly, the structure (e.g., hierarchies) of such documents is only rarely employed, though it may contain valuable information such as for grouping features or establishing parent-child relationships. Moreover, only a few data sets for SRS are provided, thus, preventing from replicating most of the studies. Next frequently, product descriptions (PD) are used ($\sim 29\%$), mainly due to their availability (e.g., via web pages such as softpedia.com) and the rich information, they contain. For instance, features are likely to appear more explicit in such documents and sometimes even bullet lists are provided. Finally, product brochures (PB), i.e., documents used for marketing reasons, are sometimes used.

Table 3.4: Overview of the post-processing techniques used by the reviewed approaches.

Post-processing Techniques		ID
Clustering	K-Means	P09 P10 P11 P21 P25
	K-Medoids	P10
	SPK-Means	P04 P09 P11
	Fuzzy C-Means	P09 P21
	Self Organizing Map Clustering	P21
	Hierarchical Agglomerative Clustering	P01 P02 P03 P10 P14 P16 P22 P29
	Incremental Diffusive Clustering	P04 P09 P10
	Graph Clustering	P01 P24
Miscellaneous	Formal Concept Analysis	P23 P30 P31
	Association Rule Mining	P04 P09 P10 P11
	Decision Tree	P18
	K-nearest Neighbors	P04 P09 P19 P20
	Maximum Entropy Method	P20
	Propositional Logic	P06
	Heuristics	P02 P06 P17 P23 P24 P26 P27 P28 P29 P30
	Edmonds' Algorithm	P11 P16
	Implication Graph	P06 P11

Usually, they highlight main features of a product, thus, making it easy to extract them. However, due to their limited purpose (e.q, acquiring new customers), they are usually incomplete and contain very little or no variability information.

Observation 4. *Software requirements specifications are frequently used as input for feature and variability extraction from natural language documents. However, the reviewed papers provide no or only limited (i.e., small, artificial) SRS documents, thus, impeding replicability and making applicability in practice questionable.*

RQ1.2: How are the techniques supported regarding tools and automation?

Since the process of extracting features & variability is pretty complex and tedious, sophisticated tool support and a high degree of automation are inevitable. We use both aspects as quality criteria for the reviewed papers, and thus, evaluate them using a 3-point scale. The results are presented in Table 3.5 (last two columns), with tool support further divided into tools for NLP which is indispensable on the basis of Figure 3.2 and tools for feature & variability extraction (FVE) developed based on the proposed approaches.
For tool support, ● means that comprehensive tool support exists *and* is available.

Accordingly, ◐ indicates that tool support is described, but not available. Finally, if tool support is neither provided nor described, the approach is rated with ○. We use the same symbols for the degree of automation, indicating that an approach is fully automated, semi-automated, or only minor/not automated.

Tool support: Our analysis reveals that some approaches do not provide any tool support ($\sim$ 20% for NLP & $\sim$ 45% for FVE). In most of these cases, only algorithms used are mentioned, but neither their origin nor how multiple algorithms are put together is elaborated. This, in turn, makes it hard to reason about the respective approaches, thus, mitigating the trust in the applicability of them. For another minority of approaches ($\sim$ 13%), the NLP tools are not accessible. Besides, in a fairly large amount of approaches ($\sim$ 42%), the FVE tools are described, but not provided or not available anymore (e.g., [WCR09]). This is especially surprising, as in some of these cases researchers invested considerable effort in building whole frameworks to automate the extraction process. Even worse, the non-availability also questions the sustainability of such approaches, that is, whether they have been created for practical usage or just for theoretical evaluations. Finally, only for a minor amount ($\sim$ 13%), the FVE tools are available online, for instance on Github, usually complemented by examples and further material. In some cases, the tools appear to be mature, stable, and designed to be used by a wide range of researchers and practitioners (e.g., [FSGD15, NBA+17]). By contrast, the third-party NLP tools are usually available online which support the majority of the studies ($\sim$ 67%).

Observation 5. *When tools are available, they make a stable, maintained, and reliable impression, and thus, are likely to be applicable in practice. However, in most cases, no FVE tools are described or even exist, thus, mitigating trust in applicability and reliability of the corresponding approaches.*

Automation: The vast majority of approaches ($>$ 90%) foster a semi-automated approach, where at least some manual adjustment is required. Most commonly, parameters and thresholds need to be specified (e.g., [WCR09, DGH+11, ACP+12]) or domain analysts have to interact with the described tool in order to correct or validate the information. Approaches that provide full automation refrain from taking user input into account, although this may influence the results, e.g., regarding feature names. Moreover, our analysis reveals that these approaches usually exhibit a rather low accuracy (cf. Table 3.5).

Observation 6. *Most approaches provide a high degree of automation, whereas full automation is an exception (but possible in general). This is mainly due to the complex extraction process, which requires manual assessment and domain knowledge to achieve a satisfactory result.*

RQ1.3: How reliable are the approaches, proposed for feature extraction in SPL?

With this research question, we investigate the result quality of the extraction process. In particular, we focus on two quality criteria; *accuracy*, that is, to what

extent the identified feature & variability information is correct; and *completeness*, that is, whether and to what extent feature & variability information is extracted. Moreover, we assess how comprehensive the approaches have been evaluated.

We present the results in Table 3.5, using the same 3-point scale as in the previous RQ, having the following meaning. For *accuracy*, the ● means that the approach is very accurate, i.e., features (and variability) are reverse engineered correctly. Accordingly, the ◐ means that the approach is sufficiently accurate, that is, the main information is correct but contains minor inconsistencies (e.g., wrong/missing features or variability). Finally, the ○ means that the approach is inaccurate, thus, only capable of providing a very high-level overview of separated concerns. For *completeness*, the ● means that the complete information (i.e., features & variability) is extracted explicitly, most likely in form of a feature model. In contrast, the ◐ indicates that variability information is only partially extracted. Consequently, the ○ means that no variability information can be extracted by the approach, thus, the result is only a list of features. For the *evaluation*, ● indicates that a comprehensive and also *reproducible* revaluation is provided that allows for detailed reasoning about the proposed approach. Likewise, the ◐ means that the evaluation has some limitations, for instance, only a small or artificial case study is used or the case study is not reproducible, thus, the results of the study can not be verified. Finally, the ○ refers to approaches that provide no or only a weak evaluation.

Accuracy: Our analysis reveals that many approaches ($\sim$ 55%) are inaccurate, however, for various reasons. While some of them perform rather bad in corresponding metrics (e.g., precision, recall; [CZZM05, KBA15, HW15]), other approaches simply do not provide any information about it. For the latter, the reason is that they focus on other aspects such as usefulness for guiding developers, thus, the evaluation does not provide any quantitative measures (e.g., [IRB14, IRBW16]). Next, some approaches ($\sim$ 42%) provide relatively accurate results, which can be seen as a starting point for further, manual refinement. Finally, only one approach ($\sim$ 3%) is highly accurate, and thus, can be used out-of-the-box (i.e., without manual adjustment).

Observation 7. *The accuracy is one of the most critical problems that needs to be improved for achieving practical applicability. However, it seems that even with less or unknown accuracy, several approaches perform well in supporting developers in manual domain analysis or even extraction processes.*

Completeness: Generally, many approaches extract partial variability information ($\sim$ 45%). However, in most of these cases, only some relations (e.g., mandatory, optional) are extracted, and thus, important information is missing. For a similar amount of approaches, only features are identified, but not their relationships ($\sim$ 35%), which means that they are of limited usefulness for the SPLE process. Finally, only a minor proportion ($\sim$ 20%) provides complete and explicit variability information that makes it possible to generate a complete feature model with detailed relationships.

Table 3.5: Result of the qualitative analysis of primary studies.

ID	Accuracy	Completeness	Evaluation	Tool Support		Automation
				NLP	FVE	
P01	○	◐	◐	○	○	◐
P02	○/◐	○	○	●	○	◐
P03	○/◐	◐	○	●	◐	◐
P04	◐	◐	◐	○	○	◐
P05	○	○	○	○	○	◐
P06	◐	◐	◐/●	○	●	◐
P07	◐	◐	◐	●	◐	◐
P08	○	◐	○	◐	○	◐
P09	◐	◐	◐/●	●	○	◐
P10	◐	○	○/◐	●	◐	◐
P11	◐	◐/●	○/◐	●	○	●
P12	◐	◐	◐	●	○	◐
P13	●	○	◐	●	◐	◐
P14	○	●	○/◐	●	◐	◐/●
P15	○	○	○	●	○	◐
P16*	○	○	○	●	○	●
P17	○	◐	○	●	◐	◐
P18	○	○	○	●	○	◐
P19	○	◐	○	◐	●	◐
P20	○	○	◐	●	○	◐
P21	◐	○/◐	◐	●	◐	◐
P22	○	●	◐	●	◐	◐/●
P23	◐	○/◐	◐	●	◐	◐
P24	◐	○	◐/●	●	○	◐
P25	○	◐	○	●	◐	◐
P26	○	●	○	◐	◐	◐
P27	◐	●	●	●	●	●
P28	○	●	○	◐	◐	◐
P29	◐	◐	◐	●	◐	◐
P30	◐	◐	●	○	●	◐
P31	○	●	○	○	○	◐

FVE: feature & variability extraction; *this approaches has been negatively evaluated, because it takes already existing feautre diagrams as input, thus, being partially out of scope.

Observation 8. *When features and variability information are complete and explicit, they constitute a feature model with detailed relationships. However, in most cases, the approaches are incomplete wrt. variability information, thus, requiring increased manual effort for recreating this information.*

Evaluation: Surprisingly, many of the reviewed approaches ($\sim$ 55%) provide a weak or even no evaluation. In some cases, the approaches are just sketched, but fail to evaluate it in any sense, while others lack important information such as study design or sound evaluation criteria. Furthermore, certain approaches ($\sim$ 35%) provide a basic evaluation, but lack reproducibility due to missing access to tools and/or data sets. Finally, only a minor proportion ($\sim$ 10%) presents a comprehensive and reproducible evaluation that gives valuable insights into benefits and limitations of the respective approaches, thus, making their claimed contribution convincing.

Observation 9. *A comprehensive and reproducible evaluation makes an approach more reliable and allows for reproducibility. However, in most cases, evaluations are performed either in a weak and unsound manner or lack important resources to reproduce them. Especially the latter aspect impedes an objective comparison.*

3.3 Discussion

Based on our detailed analysis, we summarize and discuss our observations. In particular, we elaborate on aspects that constitute challenges and need to be improved in future research.

Input format matters. In our survey, we found SRS and product descriptions to be the most common input for feature & variability extraction. While SRS provide the most detailed information, representing the domain, we identified a lack of access to SRS, which impedes the progress in developing approaches for information extraction from such documents. On the other hand, product descriptions are freely available, but only reflect an incomplete overview of a domain. For future research, we see two challenges. First, a detailed comparison (by means of a sound evaluation) to what extent product descriptions can be used for domain analysis as an alternative for SRS. Second, to design reverse engineering approaches so that they can be used flexibly with different input formats, in particular, supporting SRS and product descriptions.

Extracting variability is challenging. Extracting variability information is by far the most challenging task, indicated by the rather low proportion of accurate and complete approaches. Depending on the kind of input documents, different approaches are used to extract variation points. However, most of them need manual intervention, i.e., the result of the automated extraction is not accurate and complete enough to get rid of domain analysts' correcting. The reason is that variability extraction from natural language documents is a process of understanding natural language, which is usually full of ambiguities. Thus, the challenge is to improve existing approaches, either by new combinations of existing techniques or by developing new techniques. Also, we see great potential in taking additional information

(e.g., domain knowledge) into account, which poses the challenge of integrating it in an automated extraction process.

Rethinking sustainability. Tool support and a coherent evaluation are mostly missing, due to several reasons. For making progress in extracting features & variability, especially regarding its applicability in practice, these aspects need to be addressed in the future. First, establishing a ground truth (e.g., a gold standard) is inevitable for assessing future approaches. This not only allows us to compare approaches with each other, but also to draw a conclusion about their performance (in terms of accuracy) and robustness. Second, reusing existing approaches for reproducibility and improvement is of superior importance for making the next step. This, in turn, requires a common sense of tool building, which may go beyond the scope of pure research. Nevertheless, we argue that one of the challenges is that fundamental ideas are backed by tools that can be accessed by others. This way, the researcher can join forces, and thus, push the boundaries for extracting features & variability.

3.4 Threats to Validity

Construct Validity: Our search string may be incomplete, and thus, limit the diversity of information from digital engines. We addressed this issue by diversifying search terms, extending to their synonyms, and elaborating on the different meanings between them. In addition, we carefully derived the search term based on our research questions.

Internal Validity: First, we may have overseen important papers to be included in the survey, due to bias of primary study selection or negligence. We addressed this issue by an independent repetition of the literature search by another researcher, according to the presented methodology (cf. Section 3.1). Second, the assessment of the approaches, regarding our proposed quality criteria, is prone to be subjective, thus, introducing bias in the overall evaluation. We addressed this issue as follows: Two researchers *independently* assessed all primary studies regarding the criteria given in Table 3.5. Afterwards, we compared our results; in case our opinion diverged, we discussed the reasons for our assessment and, finally, made a common decision.

External Validity: We did not expand the primary study selection to books, which possibly affects the generalizability of our study.

Conclusion Validity: The data extraction may be of bias. To tackle it, the one researcher initially extracted data complying with the predefined data extraction form, then double-checked by the second researcher and discussed whether the data was accurate and appropriate to answer the research questions.

3.5 Related Work

In this section, we discuss related literature reviews addressing reverse engineering techniques for extracting features and analyzing the commonality and variability of products.

Dit et al. conducted a systematic review of feature location techniques in source code, including case studies and tools, and then, presented a taxonomy for classification along nine key dimensions [DRGP13]. However, although encompassing NLP and Information Retrieval techniques, the considered techniques address source code comments, identifiers, etc., rather than textual requirements.

Khurum et al. presented a systematic review of domain analysis solutions to analyze the level of industrial application and/or empirical validation of the proposed solutions [KG09]. Moreover, they investigate the usability and usefulness of the proposed approaches. However, this review does not present the specific approaches, used to identify features and their relationships from the primary studies, especially targeting natural language documents.

Lisboa et al. conducted a systematic review of domain analysis tools that support the domain analysis process to identify and document common and variable characteristics of systems in a specific domain [LGL+10]. This review covers large-scale tools analyzing different sources, instead of only focusing on natural language documents, thus being less comprehensive.

Berger et al. presented a systematic review of variability modeling practices in industrial SPL to provide insights into application scenarios and perceived benefits of variability modeling, notations and tools used, the scale of industrial models, and experienced challenges and mitigation strategies [BRN+13]. However, this review focuses on notations and related tools employed in variability modeling rather than on approaches for extracting features & variability.

Alves et al. conducted a systematic review of requirements engineering for SPL to suggest important implications for practice and identifying research trends, open problems, and areas for improvement [ANAV10]. This review also provides a survey of semi-automatic or automatic tools used. However, it was conducted in 2009, leading to a lack of the latest tools. The types of surveyed requirements artifacts include not only natural language documents, but also requirements in various forms, e.g., features and orthogonal variability models. In addition, it does not focus on approaches for feature & variability extraction.

Bakar et al. presented a systematic review of feature extraction approaches from natural language requirements for reuse in SPL [BKS15]. This review provides a detailed survey of approaches used for identifying features and analyzing their relationships from textual requirements, e.g., NLP techniques and clustering approaches, and also specifies the evaluation approaches. However, this review just contains 13 studies, which can not offer a comprehensive survey and does not provide sufficient evidence of the available tools supporting the variability information extraction.

3.6 Summary

Feature and variability information extraction from natural language documents can obtain an explicit mapping of feature and variability information to other artifacts. However, there are few systematic literature reviews providing a comprehensive and detailed survey of approaches and tools used for extracting features and their relationships from natural language documents. In this chapter, we present the results

of our systematic literature review of 31 papers that propose approaches to extract features and variability and which we compared and analyzed qualitatively based on several criteria.

Based on our review, we made several observations and derived implications and challenges for current but also future research in this field. Among others, our key findings are that: a) Software requirements and product descriptions are the most common natural language documents, but exhibit differences with respect to the information used for extraction; b) Many approaches are neither accurate nor complete, and thus, of limited use in practice due to increased manual effort or simply wrong information; c) Tool support is rather sparse, which impedes reproducibility, and thus, makes a fair comparison and reasoning impossible; d) Full automation is hard to achieve and, based on several evaluations, may even not be wanted or possible, because some amount of domain knowledge is only manually available or expressible.

Based on our findings, we suggested several actions to overcome current limitations, but also to provide more solid foundations for future research in this direction. In the following chapters, we intend to tackle some of the mentioned challenges by ourselves.

4. An Initial Self-Learning Structure for Feature Extraction

This chapter is based on and shares material with the SANER'18 paper "Extracting Features From Requirements: Achieving Accuracy and Automation with Neural Networks" [LSS18a] and the SPLC'18 doctoral symposium paper "Feature and Variability Extraction from Natural Language Software Requirements Specifications" [Li18].

As introduced in Chapter 3, reverse engineering techniques are common means to support automatic extraction of features and, in some cases, even their variability information. Feature extraction techniques play an important role to automatically identify features by analyzing natural language documents. Recently, various approaches that focus on requirements to recover features have been proposed (cf. Chapter 3), because:

1. Requirements contain more comprehensive information about commonalities and variability;

2. Requirements establish traceability links to other artifacts of later development phases (e.g., source code).

In previous research, diverse natural language processing and data mining techniques have been applied to analyze semantic information and cluster similar features. However, the majority of the existing approaches lack either accuracy of extracting feature and variability information or a sufficient degree of automation, thus, requiring extensive manual intervention. And, we observe that the syntactic and semantic information of the requirements is analyzed and applied to further achieve features and variation points. For example, some of these approaches highly depend on the syntactic information of sentences (i.e., parse trees) [RBIW14,

IRB14, Wan15, Wan16, IRBW16, HW15], which requires manual intervention at the very beginning to preprocess the original requirements according to some specific rules (e.g., requirement parsing), even though external tools such as Stanford Parser [MSB+14] 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 [Mil95] to identify synonyms and compute the similarity [IRB14, Wan15, Wan16, IRBW16]. Meanwhile, another part utilizes traditional distributional semantic models (DSMs), for instance Vector Space Model (VSM) [KTWF12] and Latent Semantic Analysis (LSA) [ASB+08, WCR09], to calculate the similarity.

The potential disadvantages of the aforementioned techniques used for feature and variability information extraction can be summarized in the followings:

D1 If external tools for achieving syntactic information are used, the accuracy of results from external tools highly affect the accuracy of feature and variability information extraction. That is to say, the error from external tools will be introduced into the subsequent analysis process for feature extraction. Moreover, this kind of external error is unpredictable and uncontrollable.

D2 For requirement parsing, it not only takes a significant amount of time for preprocessing, but also requires experienced domain engineers who should know well about the syntactic-related rules specially designed for requirement parsing.

D3 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.

D4 Moreover, traditional DSMs can be considered as "*count* models" [BDK14], since 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 *prediction* models [BDK14].

Since features provide a general overview of the software product line in terms of user-visible functionalities, extracting features is usually the first step for analyzing the commonalities and variabilities in the software product line. To overcome these limitations and achieve accurate features, we propose a technique based on an unsupervised learning structure to extract features. Basically, our technique consists of three parts, 1. a Convolutional Neural Network (CNN), 2. an unsupervised dimensionality reduction function, and 3. a clustering algorithm. Using this technique allows us to make use of a compressed binary representation of the requirements and to apply word embedding models as prediction models, which have shown to outperform common "count models" [BDK14]. And, we also avoid applying external tools for achieving syntactic information and external structured database for obtaining semantic knowledge.

In this chapter, we focus on the above mentioned the combination of CNN and the unsupervised dimensionality reduction function, thus, making the following contributions:

- We propose a technique to extract features from requirements that 1. utilizes an unsupervised learning structure *without* relying on syntax information and external structured knowledge resources, and 2. integrates a prediction model to process raw text requirements and generate word vectors instead of using traditional distributional semantic models (DSMs).

- We provide insights in a preliminary feasibility study, discuss current results and possible improvements, and find key techniques that can be used in the following research.

4.1 Methodology

In this section, we introduce the proposed self-learning structure in detail, including an overview of the approach, Laplacian Eigenmaps, Convolutional Neural Network and Clustering.

4.1.1 Overview

We provide an overview of our approach and introduce the particular techniques and how they are applied in our context. We provide an overview of the entire process in Figure 4.1.

Initially, the input documents (i.e., textual requirements specifications) are pre-processed, that is, we decompose the documents into sentences and then remove stop words in these sentences.

Subsequently, the pre-processed requirements are further processed in two ways: First, by applying *Laplacian Eigenmaps* algorithm to obtain a low dimensional representation, which is then converted into binary codes. And by applying a word embedding model that creates a word vector for each word in the requirements. Next, requirements are projected into a matrix representation, by employing the pre-trained word embedding set, and fed into a CNN.

Finally, the output of CNN is compared with the binary codes to evaluate the goodness of CNN. This process is executed repeatedly and eventually, the result can be used to utilize clustering algorithms, such as k-means, to extract features based on the characteristic representation. 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 explain each step in more detail, with the clustering being out of scope for this subsection.

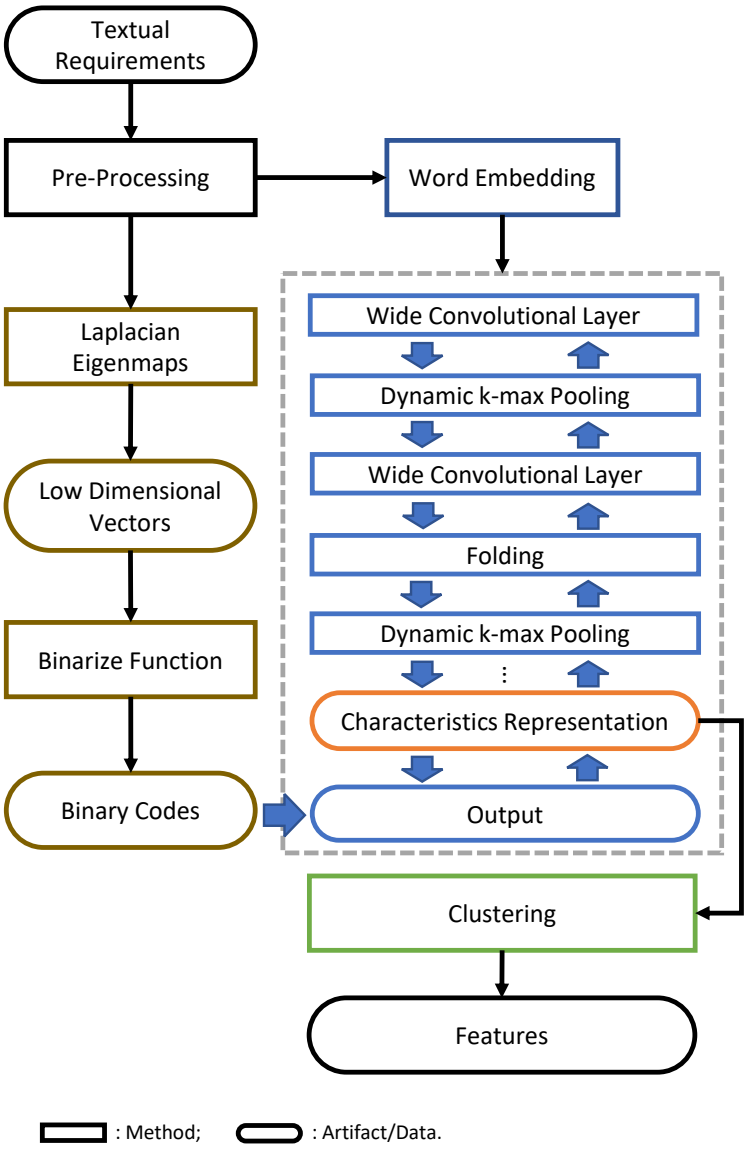


Figure 4.1: The flow chart of feature extraction.

4.1.2 Laplacian Eigenmaps

Laplacian Eigenmaps (LE) is applied to compute a low-dimensional representation of the input dataset (i.e., requirements) that optimally preserves local neighborhood information in a certain sense [BN03]. The foundation of this technique is the *laplacian matrix*, which resembles a graph, based on the input data. Hence, LE constitute a dimensionality reduction function based on graphs with nodes being data points and edges connecting nodes that exhibit a particular similarity (for a given similarity definition). Moreover, the edges have associated weights, indicating the degree of similarity between the connected nodes, and the similarity of any two nodes can be measured by the distance between them. Given these properties, the goal of applying LE is to reduce dimensionality by keeping similar nodes with a short distance, as they are likely to exhibit a relation as well in the original data.

To achieve a graph representation, we first create a vector of each sentence by TF-IDF and then use K Nearest Neighbors (KNN) algorithm to link each vector to its k nearest vectors. The similarity matrix A is computed by applying heat kernel algorithm, that is, A is the adjacency matrix of the KNN graph for the raw requirements. Given a diagonal $n \times n$ matrix D (i.e., $D_{ii} = \sum_{j=1}^{n} A_{ij}$), the graph Laplacian L can be calculated by $D - A$. The matrix D denotes the requirement significance: the bigger the value of D_{ii} is, the more significant is the i-th requirement which is strongly connected by neighbours. The optimal low dimensional matrix Y can be obtained by solving the following objective function:

$$\begin{aligned} \underset{Y}{arg\ min} \quad & trace(YLY^T) \\ subject\ to \quad & YDY^T = I \\ & YD1 = 0 \end{aligned} \tag{4.1}$$

arg min denotes arguments of the minimum which are elements that let a function get its smallest value. In order to match the output of neural network in the training process, we convert the low dimensional matrix Y into a set of binary codes B [ZWCL10].

4.1.3 Convolutional Neural Network

For our purposes. we utilize a specific kind of CNN, that is, a *Dynamic Convolutional Neural Network* (DCNN), which has been proposed for modeling sentences [KGB14]. By using this model, the order of words in the sentences is preserved and word relations of varying size can be captured. In particular, the DCNN captures word relations independent of their distance in a sentence without any prior knowledge.

In Figure 4.1 (right part), we illustrate how we employ the DCNN model for feature extraction from requirements. Initially, we transform our requirements into a word vector representation using the already mentioned word embedding. As a result, we obtain a set of word vectors E, each one characterizing a word in our requirements (i.e., the similarity wrt. all other words).

Next, we employ the word vectors E to transform each sentence s_i from the requirements into a matrix $S \in \mathbb{R}^{d_w \times s}$, where d_w is the dimensionality of the word vector and s is the length of a sentence. We then feed this matrix S into our DCNN, where it is processed consecutively by different layers (e.g., in Figure 4.1 we show two exemplary convolutional layers). Finally, the output of one iteration of our DCNN is compared with the binary codes from our LE reduction. The result is then propagated back to adjust weight matrices inside our DCNN model.This process is repeated several times in order to learn an accurate characteristic representation of the requirements. The particular layers of our DCNN model work are as follows.

Wide Convolutional Layer

In our DCNN, the convolution is a matrix computation between a *weight matrix*, called *filter*, and our input matrix S. We apply a *one-dimensional* convolution to each row of matrix S, that is, we use a one-dimensional filter to scan each sentence in order to detect its latent semantic and structure information.

There are two types of convolution: *narrow* convolution and *wide* convolution. For instance, given a sentence with seven words and a filter of length five, we can execute the narrow convolution operation only three times, because the filter must be *entirely* inside the sentence. In contrast, for wide convolution, the filter is allowed to be partially outside the sentence with all elements out of range being zero (i.e., zero padding). As a result, we can execute the wide convolution operation eleven times. Consequently, wide convolution is capable of gaining more information at the margins of the sentence, because it make sure that each weight in the filter scans all the words in the sentence.

More formally, in our wide convolutional layer(s), a filter $\mathbf{f} \in \mathbb{R}^f$ slides over each row of the requirement matrix $S \in \mathbb{R}^{d_w \times s}$, eventually resulting into a matrix $C \in \mathbb{R}^{d_w \times (s+f-1)}$, where f is the width of the filter and s is the length of the sentence. The matrix C is called the *characteristic map* of the requirement matrix S.

Folding

So far, the filter is only applied to each row of the requirement matrix S independently, thus, neglecting the relation between different rows. To overcome this drawback, *Folding* is a method to detect latent relationships between adjacent rows: At each folding layer, every two rows in a characteristic map are summarized columnwise. Consequently, for a characteristic map with d_w rows, folding returns a map $\hat{C} \in \mathbb{R}^{(d_w/2) \times (s+f-1)}$ with only $d_w/2$ rows.

Dynamic K-max Pooling

K-max pooling is a downsampling strategy in CNN to reduce the dimensionality of the intermediate layer output matrix (i.e., our characteristic map). This technique helps to avoid over-fitting by providing an abstracted form of our charateristic map. As a consequence, it reduces the computational costs since it reduces the parameters that are subject to computation (i.e., the entries of the map). The traditional k-max pooling operation is to pool the k most active characteristics (i.e., the most important characteristics) in a given sequence. Given a preset k, the sub-matrix $\bar{C} \in \mathbb{R}^{(d_w/2) \times k}$ of the k highest values in each row of the matrix $\hat{C}$ are chosen by k-max pooling. However, we apply *dynamic* k-max pooling, where the pooling parameter k is dynamically selected. As a result, we achieve a smooth extraction of higher-order and longer-range characteristics. Given a fixed, predefined pooling parameter k_{top} for the topmost convolutional layer, the parameter k of k-max pooling in the l-th convolutional layer can be computed as a function:

$$k_l = max(k_{top}, [\frac{L-l}{L}s]) \tag{4.2}$$

where L is the total number of convolutional layers in the network.

Output

The output layer is a key step to complete the learning process by fitting the binary codes B, and thus,realizing the unsupervised learning process. The output layer of DCNN is a function:

$$O = W_O h \tag{4.3}$$

with (i) h being the characteristic representation, (ii) $O \in \mathbb{R}^q$ being the output vector, and (iii) $W_O \in \mathbb{R}^{q \times r}$ being the weight matrix. Finally, we train the DCNN model by back-propagation and speed up training by using Adam optimization algorithm [KB14].

4.1.4 Clustering

Given the linguistic characteristic representation of the requirements, learned by the DCNN, the result can be used with clustering algorithms, such as traditional K-means algorithm, to group sentences which describe similar functionality into a cluster based on the characteristic representation. In particular, our intuition is that the requirements, represented by a particular cluster together, belong to the same feature.

4.2 Preliminary Result

To demonstrate the general feasibility of our approach, we implemented a prototype and applied it to a small set of requirements. In this section, we briefly state on the dataset used, the setting for our study, elaborate on initial results, and discuss them regarding accuracy and automation.

Dataset: We use the requirements from the *Body Comfort System (BCS)* [LLLS13]. BCS is a case study from a real-world scenario that takes variability into account, and it includes 98 concrete functional requirements.

Experiment setting and evaluation metrics: We use an API from gensim [ŘS10] that implements the word2vec technique, to implement word embedding, and keep all default parameters. In order to obtain pre-trained word vectors of high quality, we apply Wikipedia dumps (https://dumps.wikimedia.org/), an open source corpus with three billion words, to train word embedding. We apply cross-entropy loss function to evaluate the training process to gain the expected DCNN model [GDN13].

Preliminary result: Up to now, we don not use clustering algorithm to cluster features in terms of the characteristic representation from current DCNN model. The key problem is that the output of DCNN does not match the binary code very accurately. The loss value, which is used to evaluate DCNN model's performance, remains around 0.5 instead of continuously decreasing after approximately 1500 training steps. Clustering features based on a bad performance model makes no sense for the accuracy of extracted features.

4.2.1 Discussion

We discuss the results from above with respect to the second goal (RQ2), i.e., to achieve high accuracy and full automation.

1) For the accuracy part, we obtain a DCNN model with high loss, which means that we initially fail to achieve this goal. However, even if this is not the desired result, it was somewhat expected at this stage of our research. Basically, we identified two reasons for the rather low accuracy. First of all, our DCNN model has several parameters in the particular layers, such as weights or size of the convolution filter,

that need to be tuned. While there are numerous other approaches that elaborate on these parameters, none of these approaches focus on requirements or feature extraction therein. Hence, these parameters need to be determined experimentally and based on a sufficiently large data set. Consequently, our initial parameters serve only as a baseline (i.e., lower bound), and thus, need to be refined by applying it to more requirements and gain insights on characteristics that may affect the parameter setting. In order to optimize the model, we need to adjust some preset parameters, for example, the dimensionality of word vectors in word embedding training, size of filter in DCNN, or even the number of layers in DCNN. Another reason leading to the high loss is that the self-learning structure we designed is somewhat irrational so that it cannot learn the semantic characteristics existing in the requirements. For example, word2vec is capable of learning the meaning of a word in terms of its surrounding words. By contrast, our approach might lack the ability to capture the semantic information of requirement documents in the learning process. Moreover, LE is more likely to provide a sparse matrix representing the original requirements rather than semantic or syntactic information. Usually, CNN requires a sufficiently large dataset for the training phase to achieve reasonable results. Although the relatively small size of the dataset we used may have a bad effect on the results, we deem that it is not the main reason. The interaction of DCNN and LE in our approach does not achieve our expectations, and the structure needs to be further adjusted. Nevertheless, we argue that even with the current accuracy our approach has the potential to perform better than some other approaches with the same objective, as stated in Chapter 3.

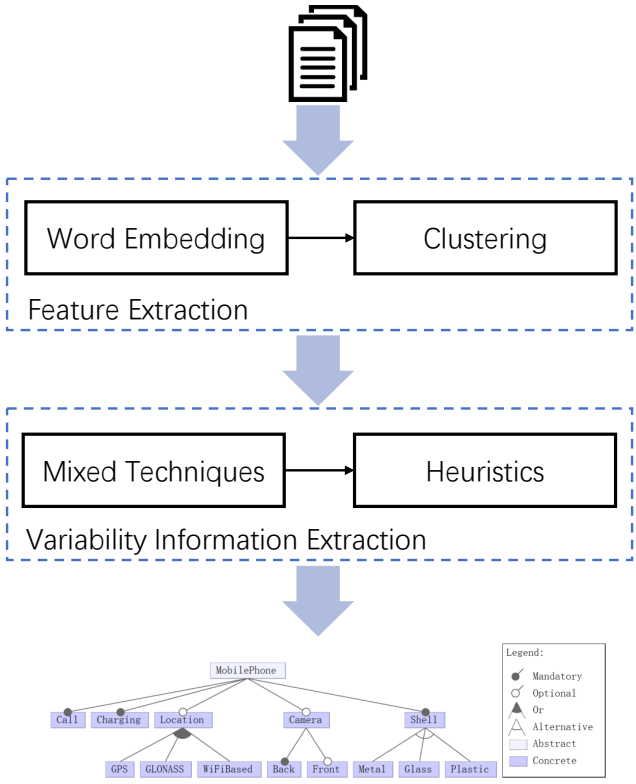


Figure 4.2: An overview of the key techniques.

2) For the automation part, we propose an unsupervised learning structure without syntactic information of sentences, because this often imposes manual analysis and correction. Neglecting the setup of parameters, we argue that the entire learning process is automatically conducted by DCNN model. Although we currently fail to obtain sufficient accuracy by our DCNN model, and thus, do not apply a clustering algorithm, the process of feature clustering can be fully automated as well. Nevertheless, some manual semantic analyses may still be needed to gain the final features in some cases. For instance, this may be necessary when the sentences in a cluster belong to non-functional requirements or even are not related to a certain software (e.g., sentences w.r.t. stakeholders).

Although we do not give a complete answer for RQ2 in this chapter, the first exploration inspires our following research. We find the key techniques that can be used to form a general framework, shown in Figure 4.2. First, we focus on using software requirements specifications as the input, since software requirements specifications contain complete information about functionality. Second, the neural-network-based word embedding techniques is capable of capturing the semantic information of words, and we can also obtain a model trained on a dataset from a particular software product line to achieve the domain knowledge of some domain-specific terms. The accurate extracted knowledge can benefit the clustering process to enhance the accuracy of feature extraction. Finally, we use heuristics to identify the variation points based on different NLP techniques, resulting in a feature model. In Chapter 5, we use these key techniques we identified during the first exploration of this initial self-learning structure to offer a complete framework for extracting features and variability information.

4.3 Related Work

Dependency on syntax and external knowledge database: Reinhartz-Berger et al. proposed an approach for analyzing features and variability by combining semantic similarity with similarity of software behavior as manifested in requirement statements [RBIW14]. To this end, they applied Semantic Role Labeling (SRL) technique that highly relies on syntactic information. Itzik et al. also used SRL to transform each sentence into six roles and computed similarity of each pair of semantic roles by applying WordNet [IRB14]. Afterwards, Hierarchical Agglomerative Clustering (HAC) is applied to these roles to cluster features. Based on the papers above [RBIW14, IRB14], Itzik et al. proposed an approach named semantic and ontological variability analysis (SOVA) to analyze variability of functional requirements [IRBW16]. 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 [IRBW16]. 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 [Wan15, Wan16]. However, Wang's research only extracted semantic information of requirements and did not extract features in the context of SPL.

All of these approaches require syntactic information and external knowledge resources to obtain accurate semantic information of requirements. In contrast, we

utilize an unsupervised learning structure (i.e., a CNN) to gain linguistic characteristic representation *without* assistance of syntactic information and external structured knowledge database.

Traditional DSMs: Alves et al. conducted an exploratory study on leveraging information retrieval techniques for feature and variability extraction [ASB+08]. They presented a framework by employing LSA and VSM techniques 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 [WCR09]. They applied LSA to measure similarity and HAC to cluster similar texts into the 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 [KTWF12]. They used VSM to determine the 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. Also, we apply CNN to learn the linguistic characteristic representation rather than directly computing the similarity of each pair of sentences.

4.4 Summary

In this chapter, we proposed an unsupervised learning structure to extract features from requirements. To this end, we combine CNN and LE to detect the linguistic information of requirements. Then, features can be extracted from the learned linguistic information by applying a clustering algorithm. In particular, we aim at improving current limitations regarding accuracy and automation without any dependency on syntactic information and external knowledge database. To this end, we apply a prediction model to build word vectors, which outperforms traditional distributional semantic models, and introduce a CNN technique for modeling sentences and to learn their linguistic characteristics. While we are struggling with the accuracy of our approach, we achieve a mostly automated process, thus, minimizing manual intervention for extracting features from requirements. Moreover, the first exploration also inspires our following research.

5. VarMine: Reverse Engineering Variability in A Hybrid Way

This chapter is based on and shares material with the SPLC'18 paper "Reverse Engineering Variability from Requirement Documents based on Probabilistic Relevance and Word Embedding" [LSS18b], and the SPLC'18 doctoral symposium paper "Feature and Variability Extraction from Natural Language Software Requirements Specifications" [Li18].

In Chapter 3, we observe 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 relatively high degree of automation (i.e., they need a lot of manual interventions), which hinders the applicability in practice. As introduced in Chapter 4, we also explore an initial approach to automatically learn the latent characteristics of requirements in order to solve the existing disadvantages (i.e., D1–D4) for traditional methods. Although the usage of external NLP tools may introduce unexpected errors in subsequent analysis, stable and available NLP tools can provide indispensable support for feature extraction tasks in terms of observation 5 in Chapter 3, which is also the reason that most studies have used external NLP tools. Therefore, in this chapter, we do not exclude the use of external tools, and we use the tools implemented by using the state-of-art techniques in order to minimize errors from external tools.

Hence, we propose an approach named *VarMine*, integrated with different machine learning and natural language processing techniques to extract features and infer the variability information. Our approach is mainly comprised of : 1) a neural word embedding model as a prediction model [BDK14] to obtain word level semantic information; 2) a probabilistic relevance framework of information retrieval to obtain requirement level semantic similarity; 3) a clustering algorithm to extract the initial feature tree; 4) a mixed method including heuristics and Recognizing Text Entailment (RTE) techniques to detect variability information. We aim to create a generic framework to identify features and extract variation points from

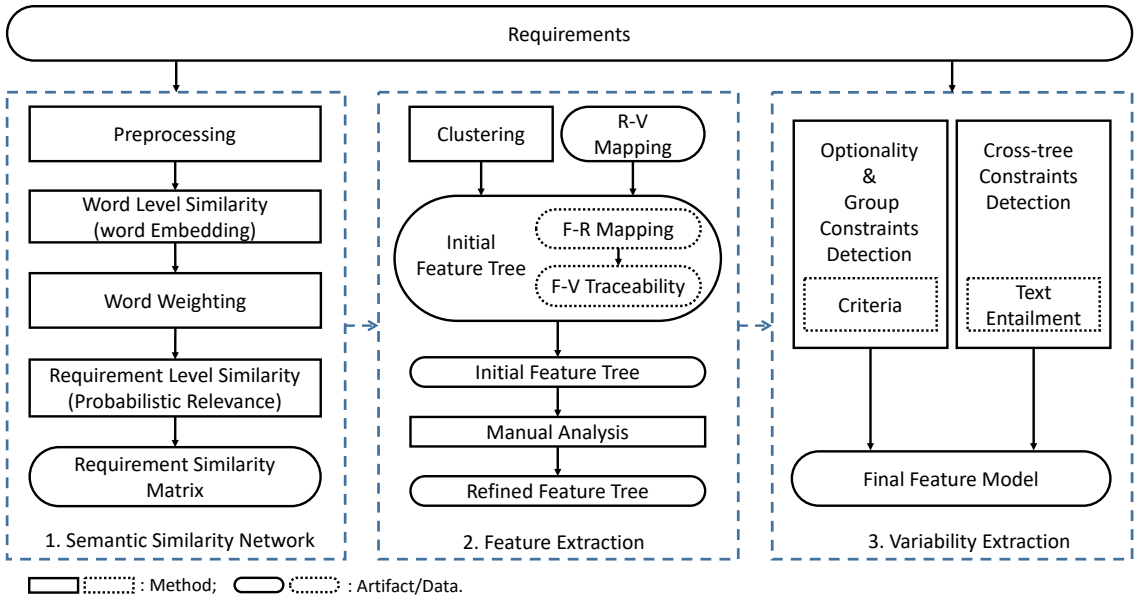


Figure 5.1: The flow chart of feature and variability extraction using VarMine.

textual requirements in order to improve the work efficiency for domain analysis. In particular, we make the following contributions to answer RQ2:

- We implement a semi-automated approach which is capable of extracting features and variability information.To this end, we integrate a prediction model into a probabilistic relevance framework to process the raw requirements instead of using DSMs.
- Meanwhile we firstly introduce recognizing text entailment technique into SPL for variation points detection.
- 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.

5.1 VarMine in a Nutshell

Our proposed approach is able to convert natural language requirements specifications into features and mine the variability information. In this section, we briefly introduce the particular techniques we use in VarMine and how they are applied in our context. We provide an overview of the entire process in Figure 5.1.

Initially, we utilize a semantic similarity network to obtain the similarity for each pair of requirements by three steps: First, 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. Moreover,

note that, the requirement specifications we used as input not only contains the textual information about functionalities, but also includes the information regarding which variants the requirements belong to. That is to say, the input documents includes a *mapping* between *requirements* and *variants (R-V Mapping)*. Second, we utilize a pre-trained word embedding model (*word2vec* in our current approach) to obtain the word level similarity of requirements (cf. Section 5.2.1). Third, 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 5.2.2). The output of this step is a requirement similarity matrix.

Subsequently, 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 (cf. Section 5.3.1). Moreover, we obtain a *mapping* 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)*. We don not directly extract the variability information from the initial feature tree. The reason is that the automated extracted feature tree cannot achieve 100% accuracy, and the variability information among the false positive features makes no sense. Hence, the initial feature tree will be analyzed by domain engineers to achieve the *refined feature tree* without false positives. Subsequently, we use our technique to extract the variation points from the refined feature tree.

Finally, after achieving the refined feature tree, based on four pre-defined criteria that take advantage of F-V Traceability link, the parental relationship (i.e., mandatory, optional, OR, XOR) between features can be extracted (cf. Section 5.3.2). Moreover, we are able to infer cross-tree constraints by applying recognizing textual entailment techniques (cf. Section 5.3.3). The final feature model is formed by both the extracted features and variability information.

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 5.2 and Section 5.3).

5.2 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.

5.2.1 Word Level Similarity

Inspired by the work of Mikolov et al. [MSC+13], we use WORD2VEC, a neural-network-based technique 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

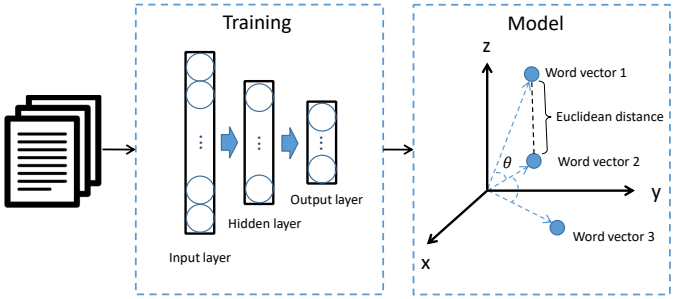


Figure 5.2: Exemplary and simplified process of obtaining word vectors.

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 words are grouped together in a semantic vector space. Hence, by using WORD2VEC, we can achieve a neural word embedding model (i.e., prediction model) to gain the more accurate vector representations of words, compared with using LSA and VSM. We measure cosine distance (also used in [MSC+13]) between two word vectors to evaluate the similarity of each pair of words, which means the *word similarity* is actually measured by the cosine value of angle between two vectors. The smaller the angle, the higher is the similarity. In Figure 5.2, we show an example of the simplified process of obtaining 3-dimensional word vectors, and we assume that the word vectors are three-dimensional vectors, because it is convenient to visualize them in a three-dimensional coordinate system. More formally, the word similarity is defined by:

$$wordSim(w_1, w_2) = cosine(\vec{w_1}, \vec{w_2}) \tag{5.1}$$

There are different metrics that can be used to compute the similarity of requirements. For example, Euclidean distance measures the distance between two points in the vector space. However, using Euclidean distance might lead to a problem, that is, even if two requirements are really similar, they may be of a large Euclidean distance, especially for the large size of requirements. In this case, Euclidean distance is not able to represent the similarity of requirements accurately. However, if we use cosine similarity, there is still a chance that the angle between the two vectors is small (e.g., "word vector 1" and "word vector 2" in Figure 5.2). As a result, cosine similarity is capable of properly calculating the similarity of different sizes of requirements.

5.2.2 Requirement Level Similarity

After obtaining the word level similarity, we extend it into requirement level similarity based on BM25+, an extension of Best Match 25 (BM25)[RZ09]. BM25+ is a so called probabilistic relevance framework for document retrieval, which improves the accuracy of processing very long requirements compared with BM25 [LZ11] 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

Table 5.1: Example of topic words.

ID	Requirements	Topic Words
r_1	If the finger protection has not been triggered, the LED does not light up.	finger, protection, LED
r_2	When the alarm system is activated, this is indicated by the light of the LED.	alarm, system, this, light, LED

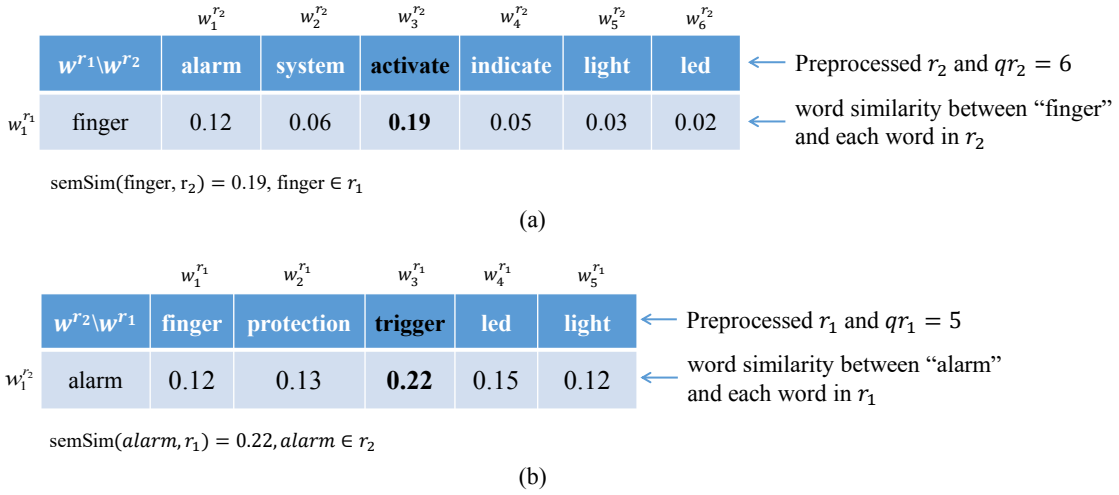


Figure 5.3: An example for computing $semSim(w, r)$.

of two short texts, taking the quantity of words in the text into account. However, the technique exhibits two limitations: 1. Term Frequency (TF) lacks the semantic information of words, since it just records the occurrence of a particular word in a document. 2. Inverse Document Frequency (IDF) can not entirely represent the significance of words in a text, since it is a statistic index without linguistic information.

Basic function

In order to improve the original BM25+, we first replace TF by using WORD2VEC to gain the word level similarity, thus, resolving limitation (1). In detail, using the two requirements r_1 and r_2 in Table 5.1 as an example, we compute the *semantic similarity* (represented by $semSim(w, r)$ in Equation 5.2) between each word in r_1 and the requirement r_2 rather than the TF of each word in r_1 appearing in r_2. For achieving these semantic similarity values, we search for the maximum word similarity between a word in r_1 and each word in r_2, as indicated by the example in Figure 5.3. This word level similarity is introduced to replace term frequency used in original BM25+, and thus, contributes to gain more semantic information.

Moreover, we establish a combination of topic words in the requirement documents and IDF in order to measure the *significance* of words (represented by $Weight(w)$ in Equation 5.2) more precisely, thus, addressing limitation (2). Specifically, 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 software product line. Here, we argue that the *subject* and *object* in the sentences of requirements are of more topic information and domain knowledge. To this end, we use dependency parser [HJ15] to automatically extract all subjects and objects of each sentence and clause in the requirement documents as topic words. For example, in Table 5.1, 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.

Consequently, the basic function used to compute the similarity of each pair of requirements based on BM25+ is as follows:

$$bmSim(r_1, r_2) = \sum_{i=1}^{qr_1} Weight(w_i^{r_1}) \times \left(\frac{semSim(w_i^{r_1}, r_2) \times (k_1 + 1)}{semSim(w_i^{r_1}, r_2) + k_1 \times (1 - b + b \times \frac{qr_2}{avgqr})} + \delta\right) \tag{5.2}$$

with

- qr_1 and qr_2 are the quantity of words in r_1 and r_2 respectively;
- $semSim(w_i^{r_1}, r_2) = max(wordSim(w_i^{r_1}, w_1^{r_2}),\ ...\ , wordSim(w_i^{r_1}, w_{qr_2}^{r_2}))$, where w^{r_1} and w^{r_2} denote each word in r_1 and r_2 respectively;
- $Weight(w_1) = \begin{cases} Preset\ Value, & w_1 \in Topic\ words; \\ IDF(w_1), & otherwise. \end{cases}$
- $b \in [0, 1], k_1 \in [1.2, 2], \delta \in [0, 1.5]$;
- $avgqr$ is the average quantity of words in all individual requirements. For example, we assume that there is a requirement document only containing two individual requirements shown in Table 5.1. r_1 includes 14 words, while r_1 contains 15 words, thereby resulting in the fact that $avgqr$ is 14.5.

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 $avgqr$ to perform requirement length normalization (i.e., the normalization of the quantity of the words in the requirements), and use δ to avoid very long documents being overly penalized [RZ09, LZ11]. However, the key differences are that we introduce word level similarity based on WORD2VEC and topic words for weighting into BM25+.

Ultimate function

With Equation 5.2, we can observe that the results of $bmSim(r_1, r_2)$ and $bmSim(r_2, r_1)$ are different. The reasons are that:

(1) When computing $bmSim(r_1, r_2)$, it would not take weight values of (topic) words in r_2 into account (as indicated by Equation 5.2), and vice versa.

(2) And, each $semSim(w_i^{r_1}, r_2)$ in $bmSim(r_1, r_2)$ is obviously different from each $semSim(w_j^{r_2}, r_1)$ in $bmSim(r_2, r_1)$ (cf. Figure 5.3).

(3) Moreover, the quantity of the words in the two requirements is also different (cf. Figure 5.3).

Hence, we conclude that if we just use $bmSim(r_1, r_2)$ to present the final similarity of r_1 and r_2, the result is inaccurate due to a loss of semantic information and significance information of words in r_2. Thus, to mitigate the impact which requirement is chosen as reference and to obtain an unique and accurate similarity value, we calculate the average of $bmSim(r_1, r_2)$ and $bmSim(r_2, r_1)$, taking the weights of words into account. As a result, the final function for requirement similarity calculation is as follows:

$$reqSim(r_1, r_2) = \frac{1}{2} \times \left(\frac{bmSim(r_1, r_2)}{\sum_{i=1}^{qr_1} Weight(w_i^{r_1})} + \frac{bmSim(r_2, r_1)}{\sum_{j=1}^{qr_2} Weight(w_j^{r_2})}\right) \tag{5.3}$$

By applying Equation 5.3, we achieve a semantic similarity network (i.e., a symmetric matrix) which encompasses the similarity values for each pair of requirements.

5.3 Feature and Variability Extraction

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

5.3.1 Feature Extraction

To extract the tree-like structure of a feature model, we apply Hierarchical Agglomerative Clustering (HAC) [RM05]. In a nutshell, similar requirements (i.e., requirements with similar functionality) are grouped into the same cluster that is considered 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* [JD88]. 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, 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 candidate inconsistency thresholds. To this end, firstly we use an arithmetic sequence to obtain the candidate inconsistency thresholds, as shown in Equation 5.4.

$$IT_n = IT_1 + (n-1) \times d \tag{5.4}$$

with

- IT_1 is the first candidate inconsistency thresholds;
- IT_n is the n-th candidate inconsistency thresholds;
- d is the common difference.

After determining the IT_1 and d, the other successive candidate inconsistency thresholds can be computed. And then, we apply every candidate inconsistency threshold in HAC to extract features. In this process, the number of features extracted by HAC decreases, when the value of the inconsistency threshold increases. Hence, the biggest inconsistency threshold is the one which results into the number of extracted features being two, since just one feature being extracted in a certain software product line is unreasonable. After obtaining all the candidate inconsistency thresholds and the corresponding features, we use two internal validity indices, *Dunn index* [Dun74] and *Calinski Harabaz index* [CH74], 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

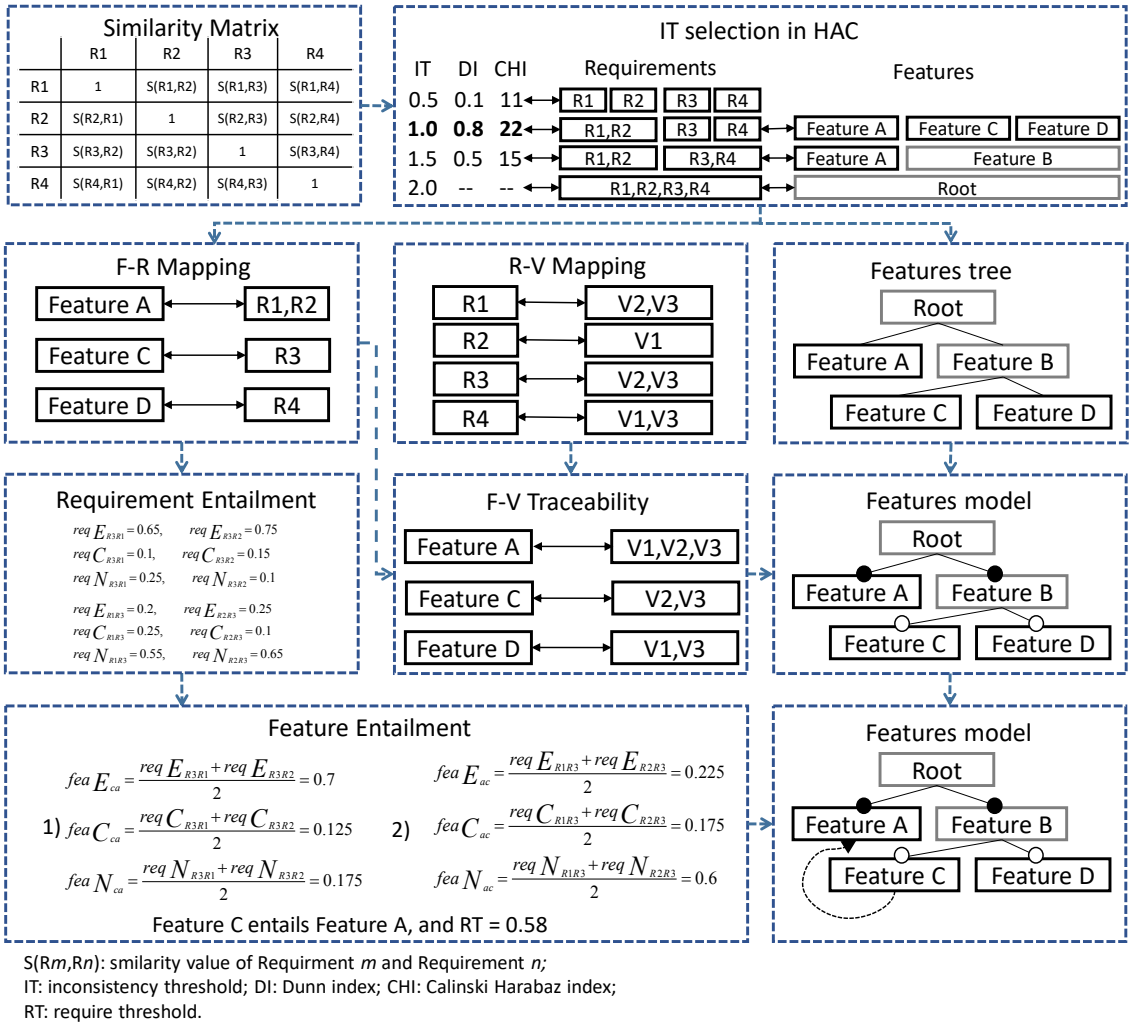


Figure 5.4: An example of feature and variability information extraction.

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 5.4 using four requirements (R1–R4) and three variants (V1–V3). The similarity matrix of requirements is firstly fed into HAC to cluster similar requirements. And then, the optimal inconsistency threshold is achieved (i.e., $IT = 1$), since the value of the corresponding Dunn index is the largest one (i.e., $DI = 0.8$). Meanwhile, as there is just one largest Dunn index, we do not take the Calinski Harabaz index into account. After clustering, 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 mapping between concrete features and requirements *(i.e., F-R Mapping)*. The concrete features are directly extracted from functional require-

ments, 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 optionality and group constraints (cf. Section 5.3.2) can be inferred from the aforementioned two mappings. Additionally, in terms of F-R Mapping, the cross-tree constraints can be detected by using textual entailment technique (cf. Section 5.3.3). Note that, this example is used to simply illustrate the proposed methodology with few features, which ignores the redundant constraints. In general, if removing one constraint does not have any influence on commonalities and variabilities presented in a feature model, this constraint is redundant. That is to say, the removal of redundant constraints does not change the validity of configurations [KAT16]. For example, the cross-tree constraint (i.e., require) between Feature A and Feature C is actually redundant, since Feature A is mandatory for Root.

5.3.2 Optionality and Group Constraints 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 5.4) 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 N_{mo} percent of all the variants from input requirement documents; or (b) all its sub-features covers N_{mo} percent of the variants. N_{mo} is a threshold, while $0 < N_{mo} \leqslant 100$.

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 N_{or} percent of the variants from input requirement documents. N_{or} is a threshold, while $0 < N_{or} \leqslant 100$.

4. Features under a same parent node form an *XOR* group, if the union of these features covers N_{xor} percent of the variants, and these features never occur in the same variant simultaneously. N_{xor} is a threshold, while $0 < N_{xor} \leqslant 100$.

The four criteria are created mainly based on the proportion of the variants covered by certain features to all the variants appearing in the documents. Next, we illustrate the application of our criteria by means of two abstract examples, denoted in Figure 5.4 and Table 5.2, where we assume $N_{mo} = N_{or} = N_{xor} = 100$. In Figure 5.4, 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

Table 5.2: Example of OR and XOR group relation extraction.

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

Table 5.3: Mapping between TE and CTC.

TE Relation	Probability	CTC
Entailment	71.9%	Require
Contradiction	1.4%	Exclude
Neutral	26.7%	Non-CTC

of feature C and D covers all three variants, feature B is mandatory. Obviously, Feature C and D are optional.

In Table 5.2, we assume that 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.

5.3.3 Cross-Tree Constraints Detection

Inspired by the research on Recognizing Text Entailment (RTE) [AM09, PTDU16], we propose a mixed method by using RTE technique with pre-defined rules to detect the cross-tree constraints. RTE technique is used to predict Textual Entailment (TE) that is defined as a directional relation between two statements named *premise* and *hypothesis*, which is able to present the probability of whether the facts in the premise implies the facts in the hypothesis. There are three different relations - *entailment, contradiction, neural* — between premise and hypothesis in TE for the general NLP purpose. Although cross-tree constraints are used to present the dedicated relations between features in SPL context, we can observe that these relations constitute a similar meaning compared with TE. Hence, we make mapping between these two kinds of relations and illustrate the general idea of using this technique with the example below.

Premise: Violent opening of a door causes an alarm to be triggered when the alarm is active.
Hypothesis: When an alarm is triggered, this is signaled by the light of the LED.

In the above example, there are two requirements which are regarded as premise and hypothesis respectively. Based on Parikh et al. research [PTDU16], we achieve three different TE relations (i.e., entailment, contradiction and neutral) with different

probabilities, shown in Table 5.3. For the sake of the latent relevance of TE and Cross-Tree Constraints (CTCs), we regard entailment as require, contradiction as exclude and neutral as non-ctc, inheriting the probability from TE. Hence, we can observe that the requirement in the hypothesis requires the requirement in premise, because of entailment with the highest probability.

Feature entailment. However, in our case, a feature may contain several requirements. Hence, we need to analyze all the TE relations between requirements belonging to features in order to achieve feature entailment relations. For example, we assume that there are two features, F_a and F_b. In terms of the *F-R Mapping*, we can achieve two sets of requirements, $F_a = \{r_{a1}, r_{a2}, r_{a3}, ..., r_{am}\}$ and $F_b = \{r_{b1}, r_{b2}, r_{b3}, ..., r_{bn}\}$. The feature entailment relations between F_a and F_b are computed by the following equation:

$$
\begin{aligned}
feaE_{ab} &= \frac{\sum_{r_{bj}=r_{b1}}^{r_{bm}} \sum_{r_{ai}=r_{a1}}^{r_{an}} reqE_{r_{ai}r_{bj}}}{m \times n},\ r_{ai} \in F_a \ and \ r_{bi} \in F_b \\
feaC_{ab} &= \frac{\sum_{r_{bj}=r_{b1}}^{r_{bm}} \sum_{r_{ai}=r_{a1}}^{r_{an}} reqC_{r_{ai}r_{bj}}}{m \times n},\ r_{ai} \in F_a \ and \ r_{bi} \in F_b \\
feaN_{ab} &= \frac{\sum_{r_{bj}=r_{b1}}^{r_{bm}} \sum_{r_{ai}=r_{a1}}^{r_{an}} reqN_{r_{ai}r_{bj}}}{m \times n},\ r_{ai} \in F_a \ and \ r_{bi} \in F_b
\end{aligned}
\tag{5.5}
$$

with

- $feaE_{ab}$, $feaC_{ab}$ and $feaN_{ab}$ are the entailment, contradiction and neutral probability respectively between feature F_a (premise) and feature F_b (hypothesis);
- $reqE_{r_{ai}r_{bj}}$, $reqC_{r_{ai}r_{bj}}$ and $reqN_{r_{ai}r_{bj}}$ are the entailment, contradiction and neutral probability respectively between requirement r_{ai} (premise) and requirement r_{bi} (hypothesis).

Then, we define two criteria to determine the reasonable entailment relations between features:

(1) If $\begin{cases} feaE_{ab} = max(feaE_{ab}, feaC_{ab}, feaN_{ab}) \\ feaN_{ba} = max(feaE_{ba}, feaC_{ba}, feaN_{ba}) \end{cases}$, F_a entails F_b.

Explanation: $feaE_{ab} = max(feaE_{ab}, feaC_{ab}, feaN_{ab})$ means when F_a is the premise and feature F_b is the hypothesis, entailment is the main relation between F_a and feature F_b, since $feaE_{ab}$ obtains maximum value compared with $feaC_{ab}$ and $feaN_{ab}$. However, we cannot directly determine that F_a entails F_b just in terms of the fact above. It is necessary to double check the relation after we switch the role of the features. That is to say, we should know what the main relation is, if F_b is the premise and feature F_a is the hypothesis. Moreover, if the two main relations are not logically contradictory to each other, we consider that the two relations are meaningful and valid to be used to further infer the final relation between the two features. In this case, if F_a

entails F_b, it must satisfy two conditions simultaneously: (a) $feaE_{ab}$ should be the maximum, when F_a is the premise and feature F_b is the hypothesis; (b) $feaN_{ba}$ should be the maximum (i.e., the main relation is neutral), when F_b is the premise and feature F_a is the hypothesis.

(2) If $\begin{cases} feaC_{ab} = max(feaE_{ab}, feaC_{ab}, feaN_{ab}) \\ feaC_{ba} = max(feaE_{ba}, feaC_{ba}, feaN_{ba}) \end{cases}$, F_a and F_b contradict each other.

Explanation: Likewise, in order to identify whether F_a and F_b contradict each other, we have to take the two main relations into account. That is to say, if we determine that F_a and F_b contradict each other, it must satisfy two conditions simultaneously: (a) $feaC_{ab}$ should be the maximum (i.e., the main relation is contradiction), when F_a is the premise and feature F_b is the hypothesis; (b) $feaC_{ba}$ should be also the maximum (i.e., the main relation is also contradiction), when F_b is the premise and feature F_a is the hypothesis.

Finally, we utilize two thresholds to filter the results of feature entailment in order to identify the reasonable cross-tree constraints for different SPLs.

(3) If F_a entails F_b and ($feaE_{ab}$ OR $feaN_{ba}$) $\geqslant$ *require threshold*, F_b requires F_a.

Explanation: The magnitude of $feaE_{ab}$ and $feaN_{ba}$ indicates the strength of the entailment relationship between F_a and F_b. Hence, we use a *require threshold* to detect the require relation between features. If (a) F_a entails F_b (i.e., criterion (1) is satisfied.), and (b) either $feaE_{ab}$ or $feaN_{ba}$ is larger than the require threshold, we say that F_b requires F_a.

(4) If F_a contradicts F_b and ($feaC_{ab}$ OR $feaC_{ba}$) $\geqslant$ *exclude threshold*, F_a excludes F_b each other.

Explanation: Likewise, the magnitude of $feaC_{ab}$ and $feaC_{ba}$ indicates the strength of the contradiction relationship between F_a and F_b. Thus, we also use an *exclude threshold* to identify the exclude relation between features. If (a) F_a contradicts F_b (i.e., criterion (2) is satisfied.), and (b) either $feaC_{ab}$ or $feaC_{ba}$ is larger than the exclude threshold, we say that F_b and F_a exclude each other.

Figure 5.4 contains a running example of CTCs extraction between Feature A and Feature C.

5.4 Evaluation

To demonstrate applicability and benefits of VarMine, we conducted two case studies. 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].

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

5.4.1 Research Questions

The RQ2 mentioned in Chapter 1 is focused on the improvement of the accuracy and automation for the feature and variability information extraction, which affects applicability of the proposed approaches in practice. For evaluating the quality of our feature and variability extraction process, we formulate the following three sub-research questions:

RQ2.1: *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.

RQ2.2: *In what ways can our approach improve the work efficiency of domain analysis?*

For this research question, we put emphasis on how our technique can decrease manual intervention in the process of identifying features and extracting variability information in order to solve time-consuming and labor cost problem.

RQ2.3: *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.

5.4.2 Case Study Description

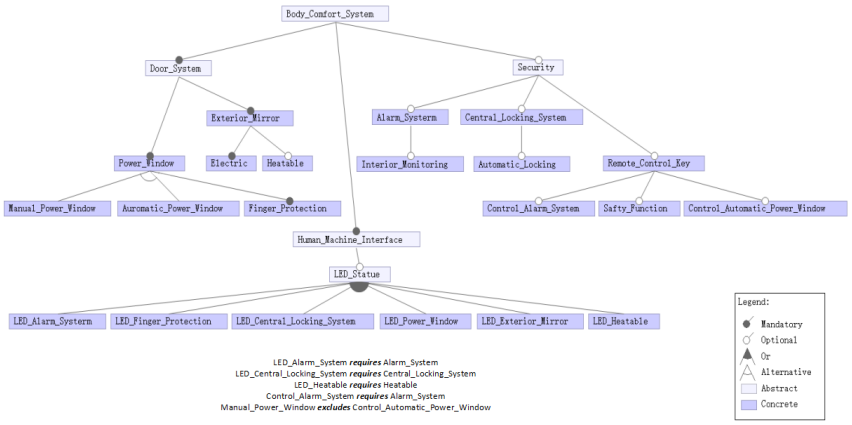
In this section, we provide details about the subject system and the general process for conducting our case study.

Dataset

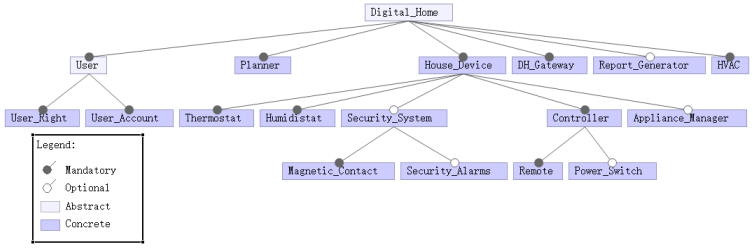
In order to evaluate our approach, two datasets from different domain are used: 1) the requirements (i.e., software requirement specifications) from the *Body Comfort System (BCS)* [LLLS13], a case study from a real-world scenario that takes variability into account, comprising 98 concrete functional requirements; 2) The *Digital Home (DH)* [HT07] requirements including 38 individual requirements. The common characteristics of these two datasets are that a) they both provide the R-V Mapping and b) each requirement in the datasets is short text with one or two sentences specifying the corresponding functionality of the system. Figure 5.5 presents the feature models of BCS and DH used as ground truth. The BCS feature model comes from the paper [LLLS13]. For the DH feature model, two senior researchers and the author of this thesis independently analyzed the DH requirements document and then reached a consensus to form the ground truth before the evaluation.

Experimental setting

Since all the data and implementation are already reachable via the aforementioned link, we just briefly introduce the key parameters setting here. For word level similarity, we are confronted with two limitations in this case study: 1) A word embedding



(a) BCS



(b) DH

Figure 5.5: The ground truth of BCS and DH feature model.

model trained with a large corpus is supposed to be of high quality. However, the BCS and DH datasets are both small datasets, which leads to a lack of enough data for training a word embedding model with high quality. 2) Although we can achieve an accurate word similarity with the word embedding model, 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. That is to say, 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. The common method to handle out-of-vocabulary words is to assign a random vector for out-of-vocabulary words [Kim14]. Unfortunately, a random vector not only lacks the ability to vectorize a word accurately but also leads to an uncertainty of the results. To overcome these limitations, we use two different models for computing word level similarity. First, we use the pre-trained model from Mikolov et al. [MSC+13] trained on Google News dataset[2] including 100 billion words as our *main model*. Second, we also train a *complementary model* on the requirements of each software product line that is subject to our variability mining analysis. The main model is utilized to gain the general meaning of a word. If a word is absent in the main model, the word probably belongs to domain-specific terms and the vector representation of this word is obtained by applying the complementary model.

[2]https://code.google.com/archive/p/word2vec/

For requirement level similarity, according to parameter setting in [LZ11], we denote $k1 = 1.2$, $b = 0.75$ and $\delta = 1$ when computing Equation 5.2 in our experiments. The preset weight value for topic words is 1, which means the topic words are more important than the common words, when computing the similarity. 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. For feature extraction, the IT_1 (i.e., the first candidate inconsistency threshold) and d in Equation 5.4 are both preset as 0.01, in order to achieve a sequence of fine-grained candidate inconsistency thresholds. For optionality and group constraints extraction, N_{mo} and N_{xor} both equals 100, while N_{or} is 70. For cross-tree constraints extraction, the require threshold equals 0.58, while exclude threshold is preset to 0.8. Moreover, our implementation depends on spaCy [HM18], gensim [ŘS10], SciPy [VGO+20], scikit-learn [PVG+11] and AllenNLP [GGN+18]. After gaining the features and variability information by our proposed technique, we employ *FeatureIDE* [TKB+09] as graphical and textual editor to visualize the extracted feature model, which we show in Figure 5.6 and Figure 5.7.

Evaluation processes.

In order to achieve a comprehensive evaluation of our technique, we have to consider three processes in our technique to be evaluated:

- 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. Moreover, it also presents the accuracy of the *F-R Mapping*.
- Second, we implement *feature model evaluation* to measure how accurate we extracted the features and variability information, namely, whether the clusters can be regarded as features and whether the extracted variability information (e.g., optionality, group constraints, and cross-tree constraints) is meaningful. To this end, we measure how well our the extracted feature model complies with the original feature model (i.e., ground truth).
- Third, we make a comparison between our approach and the state of the art for feature extraction from requirement specification.

5.4.3 Clustering Evaluation

We use two external validation techniques to estimate the clustering performance: *Average Accuracy (AA)* [LKYW05] and *Normalized Mutual Information (NMI)* [SG03]. 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 requirements clusters. NMI is an information theoretic measure to scale the mutual information of requirements between clusters [SG03]. 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 5.4: Results of Clustering evaluation.

Dataset	AA	NMI
BCS	0.80	0.83
DH	0.81	0.88

We evaluate the clustering results of our approach by comparing them with the ground truth. In Table 5.4, 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. Namely, the *F-R Mapping* is relatively accurate.

5.4.4 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.

In order to achieve a quality analysis of the results, we mainly utilize the common measures *precision*, *recall*, and *F1 score* to evaluate the accuracy of the extracted features and variability information, shown in Equation 5.6, Equation 5.7 and Equation 5.8, respectively. In this context, precision is used to determine how many features and variation points have been extracted correctly. Consequently, recall is employed to measure the proportion of correctly identified and extracted features and variation points compared with all features and variability information 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 and variation points extracted by our approach as *true positives*, 2) features and variation points generated by our approach but absent in the ground truth set as *false positives*, and 3) the pertinent features and variation points in truth set which are absent in the feature models generated by our process as *false negatives*.

$$\begin{aligned} Precision &= \frac{Extracted\ Information \cap Ground\ Truth}{Extracted\ Information} \\ &= \frac{True\ Positive}{True\ Positive + False\ Positive} \end{aligned} \tag{5.6}$$

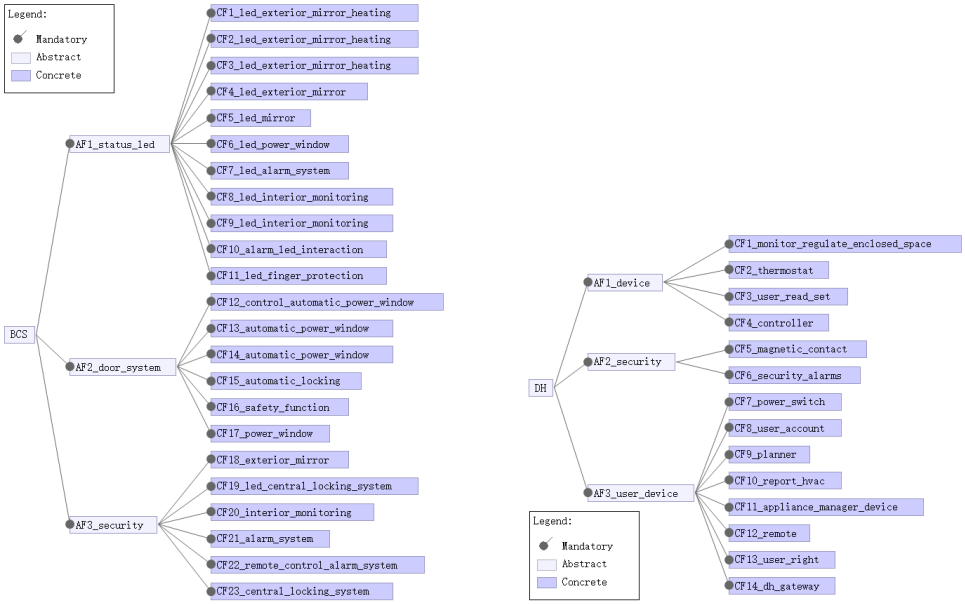
$$Recall = \frac{Extracted\ Information \cap Ground\ Truth}{Ground\ Truth} = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{5.7}$$

$$F1\ score = 2 \times \frac{Precison \times Recall}{Precison + Recall} \tag{5.8}$$

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) The extracted features and variation points are analyzed, taking the uncertainty and bias of manual analysis into account. If some extracted features are recognized as relevant properties and represent variability, we also regard them as true positives.



(a) BCS (b) DH

Figure 5.6: Initial feature tree of BCS and DH.

Feature.

Figure 5.6 presents the results of our feature extraction method on BCS and DH datasets, while Table 5.5 shows the evaluation results in both situation (i) and (ii).

In BCS dataset, 26 features including 23 concrete features and three abstract features are extracted automatically, while in situation (i) 19 of them belong to true positives (e.g., AF1, CF1, CF4, CF6, CF7, CF11, AF2, CF12, CF13, CF15, CF16, CF17,

AF3, CF18, CF19, CF20, CF21, CF22, CF23). In situation (ii), another six features also presenting variability are recognized as true positives (e.g., CF5, CF8, C10). For example, CF8 is the feature presenting the functionality of status led of interior monitoring, while the Status LED of interior monitoring is included in the LED Alarm System in ground truth (cf. Figure 5.5a). Moreover, in some cases, if there are multiple features that denote the similar functionality, we just take one of them into account. For example, although CF1, CF2, and CF3 are somewhat different with each other in functionality, they are all related to "led exterior mirror heating". CF1 and CF2 describe the activation of Status LED of exterior mirror heating, while CF3 specifies the deactivation of Status LED of exterior mirror heating. Compared with ground truth, the granularity of these three features is too fine, which is unnecessary. Therefore, in this case, we just take CF1 into account in the calculation of precision and recall.

In DH dataset, there are 17 extracted features containing 14 concrete features and three abstract features. In situation (i), 12 of the 17 features are true positive (e.g., AF1, CF2, CF4, AF2, CF5, CF6, CF7, CF8, CF9, CF12, CF13, CF14), while another one feature, CF3, is also regarded as true feature in situation (ii). In the DH case, the granularity of several features is too coarse, such as CF11, resulting in the fact that requirements with different functionality are tangled up. Hence, the intention of this kind of feature is inexplicit. We find that although some requirements describe different functionality, they contain many similar words. This is the main factor that has a bad effect on the accuracy of feature extraction in the DH case.

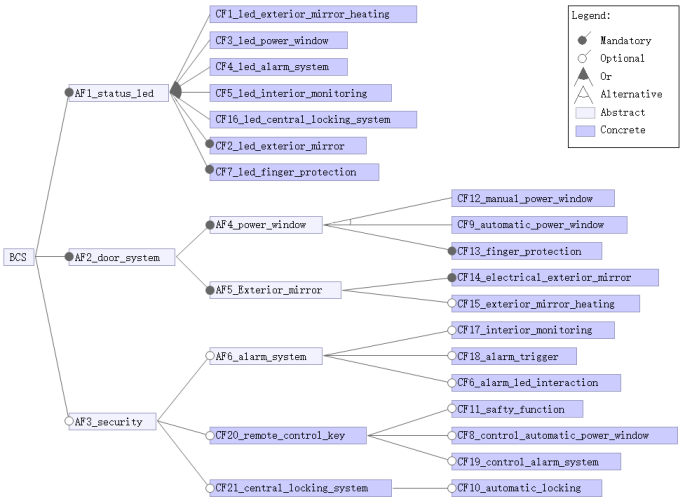
Table 5.5: Feature extraction evaluation results.

Dataset	Situation	Precision	Recall	F-measure
BCS	(i)	0.73	0.76	0.75
	(ii)	0.85	0.82	0.84
DH	(i)	0.71	0.71	0.71
	(ii)	0.76	0.72	0.74

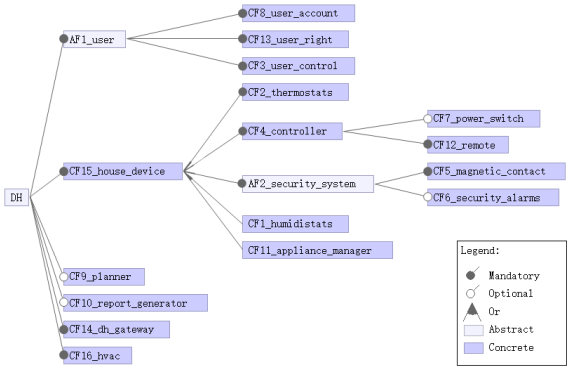
Table 5.5 shows we have achieved relatively accurate feature extraction results compared with ground truth. In situation (i), the values of precision and recall for both BCS and DH are close to each other, reaching 70%. But, in situation (ii), more potential true features also representing variability are mined from BCS than DH.

Variability information.

Variability information is subject to features, which means the accuracy of variability information extraction is prone to be affected by the accuracy of feature extraction. The errors from feature extraction will be accumulated in the process of extracting variation points, which will reduce the accuracy of the extracted variability information to a great extent. Moreover, it is meaningless to extract the relationship between false features. Hence, the features extracted automatically are further analyzed and refined by manual work. This process includes three main steps: 1)



(a) BCS



(b) DH

Figure 5.7: Refined features and extracted parental relationships.

Feature merging, which merges several features with similar functionality into one feature; 2) Feature partition, which divides one feature up into several sub-features; 3) F-R Mapping correction, which puts the wrongly grouped requirements into the correct feature; 4) Renaming features, which adjusts the names of features to fit the revised feature model. All these analysis steps are simplified and assisted by a graphical user interface (GUI) of manual analysis via drag and drop operations, introduced in Chapter 7.

Afterwards, the variability information is detected in terms of the refined features presenting in Figure 5.7, while the evaluation results is shown in Table 5.6.

Optionality and group constraints. Mandatory, optional, or-group and xor-group can be regarded as the parental relationships between features, shown in Figure 5.7. Hence, each feature is of only one parental relation, which means the number of parental relationships is equal to the number of features. We also take both situation (i) and (ii) into consideration.

In the refined BCS feature model, compared with ground truth (cf. Figure 5.5a), we find 22 true positives (e.g., AF1, CF1, CF3, CF4, CF16, AF2, AF3, AF5, CF12, CF9, CF13, CF14, CF15, AF3, AF6, CF17, CF20, CF11, CF8, CF19, CF21, CF10).

In situation (ii), the features absent in the ground truth are analyzed to make sure whether the parental relationships of these features are correct or not. Then, another three features are recognized as true positives with the right parental relationships (e.g., CF5, CF18, CF6).

In the refined DH feature model, there are 14 features with right parental relationships (e.g., AF1, CF8, CF13, CF15, CF2, CF4, CF7, CF12, AF2, CF5, CF6, CF10, CF14, CF16) in situation (i) compared directly with ground truth (cf. Figure 5.5b). Moreover, another feature CF3 is regarded as a true positive with correct parental relationship in situation (ii).

Table 5.6: Optionality and group constraints evaluation results.

Dataset	Situation	Precision	Recall	F-measure
BCS	(i)	0.81	0.88	0.84
	(ii)	0.96	0.93	0.94
DH	(i)	0.78	0.82	0.80
	(ii)	0.83	0.83	0.83

Table 5.6 gives a positive answer that the extraction of optionality and group constraints between features is doable by analyzing the F-V Traceability. It shows that the distribution of features in different variants reflects the potential relations between features. Certainly, we still cannot achieve all the optionality and group constraints correctly by only analyzing the F-V Traceability. For example, in the ground truth of BCS feature model, CF2 and CF7 should be or-grouped with other sub-features of AF1. However, by using our approach, CF2 and CF7 are both identified as mandatory features, since these two features appear in all variants. Moreover, in the ground truth of DH feature model, CF1 is mandatory and CF11 is optional, while they are or-grouped by using our approach. However, the extracted relations with high accuracy provide a very good starting point to improve the process of further manually analyzing the relations between features.

Cross-tree constraints. In order to evaluate to what extent the extracted CTCs can assist domain analysis, we divide each detected CTC into four parts: *pair, relation, direction, redundancy.*

For pair, we intend to analyze whether the pair of features in an extracted CTC is meaningful. If the two features in an extracted CTC also belong to the CTCs ground truth, they are *matched* each other. In some cases, although the two features are not totally matched with the CTCs ground truth, they are correlated with each other and still able to provide hints to finally find the real matched pair. For example, feature A and feature C belong to the pair in the extracted CTCs, while there actually exists a real constraint between feature A and B. Moreover, feature C is a sub-feature of feature B with high relatedness. Hence, the feature A and feature C are marked as *related*. If the two features are neither matched nor related, they are marked as *irrelated*.

For relation, we plan to check whether the relation between the extracted pair of features is correct or not. Since there are only two constraint keywords, require

and exclude, in each CTC, the relation is either require or exclude. if the identified constraint keyword is correct, the relation is marked as *correct*; otherwise, it is marked as *reverse*.

For direction, we consider to make sure that the direction between the two features with require relation is correct, since require is a directional relation. For example, if the detected relation between feature D and feature E is that feature D require feature E and the true relation between is the same, the direction is marked as correct. But, if the true relation is that feature E require feature D, the direction is marked as reverse.

For redundancy, we intend to identify whether the extracted CTCs are redundant constraints compared with the extracted parental relationships shown in Figure 5.7. For example, there is an extracted CTC between CF6 and CF15 (i.e., CF6 require CF15) from DH dataset. However, CF15 is a mandatory sub-feature of the root feature, which means CF15 must be selected. Hence, "CF6 require CF15" is a redundant constraint.

Table 5.7: The extracted cross-tree constraints.

Dataset	CTCs	Pair	Relation	Direction	Redundancy
BCS	CF2 require CF14	Matched	Correct	Correct	Yes
	CF3 require CF12	Related	Correct	Correct	No
	CF4 require CF18	Related	Correct	Correct	No
	CF4 require CF19	Irrelated	-	-	-
	CF5 require CF18	Related	Correct	Correct	No
	CF10 require CF20	Related	Correct	Reverse	No
	CF18 require CF19	Irrelated	-	-	-
DH	CF4 require CF12	Related	Reverse	Correct	Yes
	CF6 require CF5	Related	Correct	Correct	Yes
	CF6 require CF12	Related	Correct	Correct	Yes
	CF6 require CF15	Related	Correct	Correct	Yes
	CF7 require CF12	Related	Correct	Correct	Yes
	CF8 require CF9	Related	Correct	Reverse	Yes
	CF8 exclude CF16	Irrelated	-	-	-
	CF13 require CF9	Related	Correct	Reverse	Yes
	CF13 require CF10	Irrelated	-	-	-
	CF13 require CF11	Related	Correct	Reverse	Yes
	CF13 require CF12	Related	Correct	Reverse	Yes
	CF14 require CF12	Related	Correct	Reverse	Yes
	CF12 exclude CF16	Related	Reverse	Correct	Yes
	CF15 exclude CF16	Irrelated	-	-	-

Table 5.7 presents the extracted CTCs results from BCS and DH. Although only one CTC with perfectly matched features, correct relation and direction is identified in BCS, we observe that the majority of the CTCs is with related pairs (71%). Meanwhile, 67% and 48% of the extracted CTCs are of correct relation and direction,

respectively. Our approach does not extract CTCs with high accuracy, which means RTE is not suitable for every pair of requirements. For example, in BCS ground truth, "Manual_Power_Window" and "Control_Automatic_Power_Window" exclude each other. But, the requirements belonging to them are not of textual entailment relations. The requirements below is a typical example:

> *Manual_Power_Window:* Pressing the button to open the window will cause the window to open as long as the print is running or the window has reached the bottom.
> *Control_Automatic_Power_Window:* Pressing the remote control button to open the window causes the window to open for opening.

Even if we read them directly, it is very difficult to get the textual entailment relation between them. In this case, domain knowledge plays a vital role in the process of extracting CTCs. Furthermore, in DH case, all the extracted CTCs are redundant compared with the extracted parental relationship. This result indicates that additional CTCs are unnecessary for presenting the variation points in DH feature model, which is consistent with its ground truth. Although our approach cannot provide CTCs with high accuracy for every case, the extracted information can assist domain engineers to carefully further adjust the structure of feature model on demand, forming the final feature model.

5.4.5 Comparison with SOVA and ArborCraft

In this subsection, we make a comparison between our approach and the approaches described in [IRB14]. Itzik et al. proposed an approach named SOVA [IRB14] 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 [WCR09]. 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 [IRB14], while we apply the same experiment setting described in Section 5.4.2. Then, we implement qualitative analysis, since the ground truth is not provided in paper [IRB14].

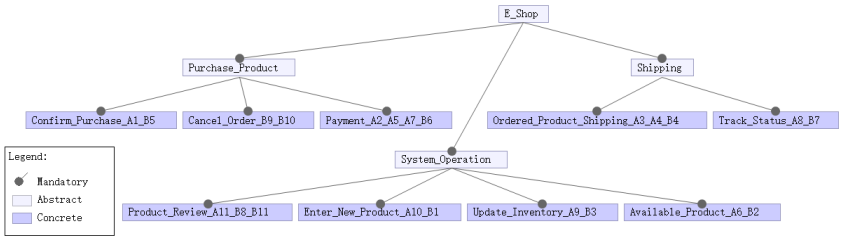


Figure 5.8: The initial E-Shop feature tree.

Table 5.8: The extracted features compared with SOVA and ArborCraft.

VarMine	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
Ordered_Product_Shipping	ShippingDocuments-A3, ShippingDocuments-A4, ProductDeliveryStatus-B4	ShipOrderedProduct	feature26
Track_Status	ProductDeliveryStatus-A8, ShipmentDetails-B7	TrackStatus	A8, B7
Product_Review	ProductReview	ReviewProduct, ReturnDamagedProduct-B8	A11, B8, B11
Enter_New_Product	Catalog	EnterNewProduct	feature22
Update_Inventory	Inventory-A9, Inventory-B3	ReturnProduct-A9, PurchaseProduct-B3	feature24
Available_Product	Inventory-A6, ShippingCart-B2	BuyProduct-A6,PurchaseProduct-B2	A6,B2

Features.

Figure 5.8 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 5.8) 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_Review 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 leading to updating inventory. Available_Product feature shows system's operation which is caused by customers' buying behavior, when available products are displayed.

We argue that the feature tree generated by our approach is more applicable than the feature tree 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 in the process of extracting initial feature trees.

Variability information.

One analyst refined the extracted features and structure, shown in Figure 5.9. Then, based on the four criteria we proposed, we found eight mandatory features and three optional features, while three features are or-grouped. Moreover, one extracted CTC (CF3 require CF12) is further identified to have related pair of features, which can be finally revised to "CF12 require CF3". This potential CTC indicates If a customer returns a damaged product, the system sends a negative review on the product to the supplier.

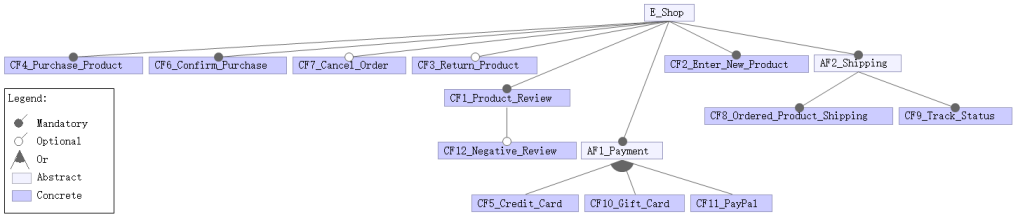


Figure 5.9: The refined E-Shop features and the extracted parental relationships.

5.4.6 Answering RQs

For RQ2.1, firstly, in terms of quantitative analysis of the BCS case study, our approach (VarMine) provides relatively high precision and recall of extracted features, especially taking situation (ii) into account. Although the precision and recall of feature extraction in the DH dataset are a little bit lower than the values in the BCS case study, we can observe that VarMine is capable of extracting applicable features from legacy requirements documents. For variability information, we achieved relatively high precision and recall of the extracted parental relationships in both BCS and DH dataset. Although the extracted CTCs do not match the ground truth perfectly, we can observe that these potential constraints are able to be used to further detect the true CTCs between features from the natural language inference perspective.

For RQ2.2, VarMine has been used for, both BCS and DH case study and the comparison with SOVA and ArborCraft. In this process, we can see that: 1) the original textual requirements can be directly processed by using VarMine, which means there is no need to manually parse the requirements with a specific rule; 2) we utilize the same preset parameters and experimental settings for processing all the data in three domains and the key parameter for clustering (i.e., inconsistency threshold) can be determined automatically in terms of different datasets, which means VarMine is capable of reducing the workload for selecting suitable parameters. Hence, in the process of achieving the initial feature tree, we successfully avoid manual intervention not only in BCS and DH case study, but also in the comparison with two related research. The manual analysis is mainly focused on refining the initial feature tree, since extracting variability information among false positive features makes no sense. Thus, 3) for feature extraction, VarMine is capable of providing potential features for domain engineers, which means they do not need to analyze the requirement from scratch. Meanwhile 4) the extracted F-R Mapping provides the opportunities and hints for domain engineers to refine the feature tree in terms of some specific needs. After feature extraction, 5) VarMine can also assist domain engineers to detect the optionality, group, and cross-tree constraints among features. Consequently, although manual analysis is essential especially to determine the final feature model, domain engineers do not necessarily spend plenty of time figuring out the features and variation points from scratch, assisted by VarMine.

For RQ2.3, comparing the results of three different SPLs, we evaluate whether our approach works properly for different domains. We first conduct the BCS and DH case study with both quantitative and qualitative analyses which reveal 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 to assist domain engineers based on the comparison of results between the other two tools. Taking both qualitative and quantitative analysis of the results in three SPLs into account, VarMine is of relative universality for different SPLs. Although we obtain both relatively reasonable results in three different SPLs, there are still many challenges needed to be overcome to make the generated feature model more applicable.

5.5 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, there are many preset parameters in VarMine affecting the result of feature and variability information extraction. For example, the weight values 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 three domains, BCS, DH 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 three SPLs. However, the results from just three domains may be not enough to support the conclusion.

5.6 Related Work

With advances in NLP, information retrieval, and data mining techniques, the development of feature and variability extraction from textual requirements is flourishing. The research of Chen et al. belongs to one of the first explorations of pursuing automated techniques for feature identification and variability modeling. They applied clustering to identify and organize features in an undirected graph presenting the requirements relation [CZZM05]. Although variability identification highly depends on manual analysis of basic relationships of requirements, the clustering part is one of the foundations for feature identification from requirements also inspiring our research.

Reinhartz-Berger et al. proposed an approach to extract features and variability information based on combining semantic similarity of requirements with the similarity of behavioral aspects of requirement statements [RBIW14]. They applied Semantic Role Labeling (SRL) technique that highly relies on syntactic information to extract different roles which are of special importance to functional requirements and then classify these roles into the initial state, external events, and final state of software behavior. However, both SRL technique and software behaviors highly rely on accurate syntactic information, which leads to substantial manual interventions

to parse the requirements in order to achieve the accurate behavioral vectors of each requirement.

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 [IRB14]. They also used SRL to transform each sentence into six roles and computed similarity of each pair of semantic roles by applying WordNet [IRB14], 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. Besides the limitation of SRL and HAC they used, WordNet as an external knowledge database is not capable of containing all word meanings of high quality. Moreover, it suffers from out of vocabulary words, especially for domain-specific words, which results in a bad effect on feature and variability extraction. In contrast, we applied the neural word embedding techniques to achieve a word vector space of the words appearing in the requirements, which is able to tackle the problem regarding out of vocabulary words.

Based on the papers above [RBIW14, IRB14], Itzik et al. proposed an approach named semantic and ontological variability analysis (SOVA) to analyze the variability of functional requirements [IRBW16]. 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 that reflect stakeholders' needs and preferences [IRBW16]. 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 a weighted calculation of words regardless of structure information of requirements, for example, the length of the requirements.

Wang proposed a method to gain semantic information of requirements by applying machine learning and SRL techniques. In this process, the semantic frames for frequent verbs appearing in requirements were built by applying SRL with the assistance of Stanford Parser and WordNet [Wan15, Wan16]. Subsequently, in terms of the frames, some sentences from requirements specifications were labeled manually, while the labeled sentences were fed into the decision tree or maximum entropy algorithm for training. However, Wang's research only extracted semantic information of requirements and did not extract features in the context of SPL. Moreover, besides the cost of manual annotation, the training set just comes from one domain (i.e., E-commerce), which affects the generalization ability for different domains.

Other approaches focus on traditional DSMs techniques. Alves et al. conducted an exploratory study on leveraging information retrieval techniques for feature and variability extraction [ASB+08]. They presented a framework by employing LSA and VSM techniques 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 [WCR09]. They applied LSA to measure similarity and HAC to cluster similar texts into the 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 [KTWF12]. They used VSM to determine the 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 [FSD13, NBA+17]. However, in Ferrari et al.'s research, they only detected mandatory and optional features, while Sana 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 the occurrence of the word with different meanings in requirements [FGS17].

5.7 Summary

In this chapter, 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 computing the similarity in order to improve the accuracy of feature extraction. Moreover, we take topic words regarding the requirements of a domain into account, while topic words can be seen as the smallest unit of domain knowledge. We use this knowledge to assign weight values to our similarity measures, and thus, improve the accuracy. We then apply hierarchical clustering to extracted initial features which will be refined by manual analysis to remove the false positives. Finally, we make use of predefined criteria to identify the optionality and group constraints and utilize the RTE based method to infer the CTCs. To evaluate the 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 the other two related approaches in a different domain. Our results reveal that we gain relatively accurate features. Although our approach is not capable of detecting all the variability information, the majority of the extracted relationships between features are reasonable.

6. The Inference of the Notions of Features

This chapter is based on and shares material with the EASE'20 paper "Feature Terms Prediction: A Feasible Way to Indicate the Notion of Features in Software Product Line" [LSX20].

Software Requirements Specifications (SRS), as the initial software development artifacts, are a suitable source for exploring domain knowledge (here: features) as they capture *what* the system should do and also provide traceability links to other artifacts. According to Chapter 3, a series of research on feature extraction from requirements has been introduced, and we also proposed approaches to extract features from requirements, resulting in mappings between features and requirements in Chapter 5.

However, there still exists a "last mile" problem to be solved in order to achieve a more comprehensive view on the extracted features, that is, the intuition behind the feature by means of functionality or some other kind of semantics. While this could be solved by analyzing the corresponding requirements manually, this is a time-consuming task that does not scale for dozens or even hundreds of requirements per feature. In order to tackle this problem, we propose to use terms that occur in the requirements and are highly related to the intention of the feature. We argue that such *feature terms* can support domain engineers to understand what a feature is about as well as serve as suggestions for feature names.

Different approaches have been proposed for identifying feature terms from textual documents. For example, in some research [SKPC18, BKSJ16, BKSH17], various heuristics are predefined in terms of the linguistic information, such as, lexical and syntactic information of the requirements, to find all the candidate terms from requirements matching some specific rules. After obtaining the candidate terms, several techniques can be used to further analyze these terms. For instance, word weighting methods can be applied to measure the importance of each term by assigning a weight value to terms [SKPC18]. In terms of the weights, feature terms can be filtered by certain thresholds or fed in a clustering algorithm to obtain groups

of similar terms [BKSJ16, BKSH17]. Although the methods based on predefined rules are useful for certain cases, not all the terms extracted by using these rules can correctly indicate the intention of features even with the assistance of certain word weighting methods. Among others, the amount of requirements that are used for feature term extraction or the domain may lead to terms that are incorrectly identified. A different approach that has been proposed is to label the true feature terms in the raw requirements and then use this information to train a model to predict feature terms [BEG12]. Even though this avoids defining dedicated rules for different scenarios, it may suffer from insufficient data in the training process, and thus, result in a rather poor prediction model. The reason is that simply labeling the terms is not enough to provide characteristic information of terms that can be learned in the training process.

Hence, for extracting feature terms, there exist two main limitations in current research:

1. The limited heuristics and rules based on linguistic information lack the ability of generalization for different scenarios.

2. With supervised Machine Learning (ML), the labeled data is of low quality, thus leading to a rather inaccurate (learned) model.

In order to overcome these limitations, we propose a supervised machine learning based approach to identify feature terms from requirements, integrated with basic rules used to extract candidate terms. In particular, we analyze different attributes of each term, where an attribute actually represents a certain characteristic of a term, such as the linguistic characteristic, statistical characteristic and other potential characteristics that may affect whether a candidate term is a feature term (cf. Attributes extraction in Section 6.1.1). As a result, we can train the prediction model on these attributes rather than on the terms directly, which allows to abstract away specialties that are, for instance, domain-dependent.

In this chapter, we make the following contributions to answer RQ3:

- We identify six representative attributes of terms to create the labeled data with relatively high quality for supervised machine learning.

- We propose an approach that combines rule-based and learning-based feature terms extraction in order to increase accuracy.

- We utilize seven different machine learning algorithms within our approach to obtain the best prediction model for each algorithm trained on the labeled data, and thus, to assess the influence of the ML algorithm on the accuracy.

- We present an empirical study to evaluate our approach including the comparison of the performance of different prediction models across datasets from different domains.

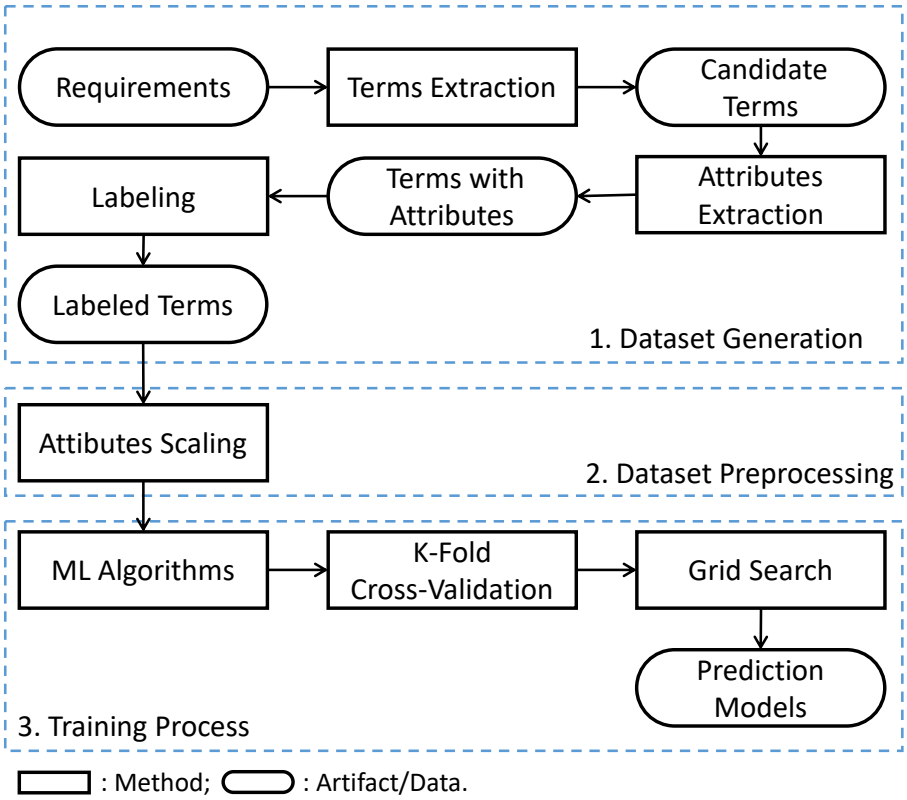


Figure 6.1: The Overall workflow of our proposed approach.

6.1 Methodology

In this section, we explain our proposed approach for identifying feature terms. In Figure 6.1, we show an overview of the methodology. Initially, we need to create a labeled dataset for training the prediction model. To this end, we must extract the candidate terms from requirements, analyze the attributes of each candidate term and assign labels to these terms. Subsequently, the raw data need to be preprocessed. Finally, the preprocessed data are fed into seven different machine learning algorithms to train prediction models. In particular, we use Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), K Nearest Neighbors (KNN), Random Forest (RF), Gaussian Naive Bayes (GNB) and Multi-Layer Perceptron (MLP). Based on the obtained prediction models, we then can predict feature terms and evaluate their accuracy.

6.1.1 Dataset Generation

The labeled dataset is indispensable for supervised learning methods. Here, we introduce how the labeled dataset for training is created in detail.

Running example

In order to specify how the labeled dataset is created, we use two features and the corresponding requirements from the Body Comfort System (BCS) as a running example, shown in Table 6.1. BCS is a case study from a real-world scenario [LLLS13]. It encompasses 98 concrete functional requirements and also contains a complete

Table 6.1: An example of requirements.

F-ID	R-ID	Requirements
1	1	When the alarm system is activated, this is indicated by the light of the LED.
	2	If the alarm is deactivated, the LED does not light up.
	3	When an alarm is triggered, this is signaled by the light of the LED.
	4	If no alarm is triggered, the LED does not light up.
	5	When the interior monitoring is activated, this is signaled by the light of the LED.
	6	If the interior monitoring is not activated, the LED does not light up.
	7	The LED, which signals the triggered alarm, can only be reset by pressing the reset button, which is next to the LED. The key has only one effect when the ignition key is in the ignition.
2	8	Two temperatures are defined: a minimum temperature and a maximum temperature.
	9	If the temperature measurement on the exterior mirror falls below the minimum temperature, the exterior mirror heating is activated.
	10	Switching off the vehicle leads to the switch-off of the exterior mirror heater.
	11	When the exterior mirror heating is activated and the maximum temperature is exceeded, the exterior mirror heating is deactivated.
	12	If the LED is present and the exterior mirror heating is activated, the LED is lit.
	13	If the LED is present and the exterior mirror heater is not activated, the LED does not light up.

F-ID: Feature ID; R-ID: Requirement ID.

feature model generated by domain engineers. Hence, we can obtain the mappings between features and requirements, and thus, we know which feature an individual requirement belongs to.

Terms extraction

We define three rules to extract all the *candidate terms* in the requirements belonging to a particular feature based on *Part-of-Speech (POS) patterns* that take the linguistic information into account:

1. We just focus on nouns and phrases that end with nouns which are identified by POS patterns. The rationale behind this idea is that nouns and phrases that end with nouns are usually used to indicate functionalities in the requirements.

2. We remove single-word terms that have already occurred in a phrase (i.e., multi-word terms) from terms extracted by the predefined POS patterns, since

we argue that such phrases convey more explicit information than single-word terms.

3. Similarly, we also remove the multi-word terms that have never been used independently, i.e., that always occur as part of larger multi-word terms, in order to filter out the redundant terms, which can improve the quality of the extracted terms.

Table 6.2: An example of how to use the three rules.

POS Pattern	Exemplary Terms	Candidate Terms
[PROPN]	LED	Yes
[Noun]	system	No
[Noun+Noun]	alarm system	Yes
[Noun+Noun]	mirror heating	No
[ADJ+Noun]	exterior mirror	Yes
[ADJ+Noun+Noun]	exterior mirror heating	Yes

PROPN: Proper noun. ADJ: Adjective.

Table 6.2 shows partial POS patterns together with the corresponding exemplary terms extracted from the requirements in Table 6.1 based on rule (1). Note that *proper noun* is a noun starting with a capitalized letter, no matter where it appears in a sentence. Moreover, the table contains information of whether an exemplary term is a candidate term or not, based on rule (2) and (3). For instance, the single-term "system" has already occurred in the multi-word term – "alarm system" in feature 1, and thus, "system" is not a candidate term based on rule (2). Moreover, based on rule (3), "mirror heating" cannot be regarded as a candidate term, since it is always used in the phrase – "exterior mirror heating". Meanwhile, "exterior mirror" is applied independently in requirement 9 of feature 2. So, "exterior mirror" can be regarded as a candidate term.

In summary, we use the three predefined rules to obtain meaningful candidate terms and filter out redundant terms, such as "system" and "mirror heating".

Attributes extraction

After obtaining all the candidate terms, we identify six different attributes of terms belonging to a particular feature: *Proper Noun (PN), Sub-Term (ST), Term Frequency – Inverse Document Frequency (TF-IDF), TextRank (TR), C-Value (CV)* and *Ratio of Terms (RT)*. We explain the rationale of each term in the following.

PN: Proper Noun is a kind of special noun and it is usually utilized to describe a specific thing in natural language. We regard proper noun as one of the attributes, since proper nouns widely occur in different requirements and they might represent specific functionalities. If the term contains a proper noun, we regard its attribute of PN as true (i.e., $PN = 1$). Otherwise, it is false (i.e., $PN = 0$).

ST: Sub-Term is the term that is part of another, longer term. This kind of sub-term information provides meaningful hints about how terms are used in the requirements

for describing a feature. For instance, either the terms might occur independently to specify a concrete functionality, or they appear together with other words to describe the details of a functionality. In order to take this information into consideration for feature term extraction, we use the attribute of ST to record this information for each term in a feature. If a candidate term from a particular feature is used as a sub-term of another candidate term of this feature, its attribute of ST is true (i.e., $ST = 1$). Otherwise, it is false (i.e., $ST = 0$).

TF-IDF, TR and CV: TF-IDF, *TextRank* and *C-Value* are different techniques that can be used to measure the statistical significance of the words appearing in documents. However, each technique computes the *importance value of words* (i.e., the weight of each word in documents) from different perspectives.

TF-IDF not only focuses on the frequency of each word appearing in one document, but also analyzes the information that how many times the words appear in all other documents simultaneously. TextRank builds a graph to rank the importance of words, where nodes represent words and edges represents the co-occurrence of the linked words [MT04]. Finally, C-Value is designed to measure the significance of multi-word terms (i.e., phrases), taking the frequency of both, a phrase and its sub-phrase, into account [FAT98]. In order to take all these three different methods for measuring the significance into account, we apply all the attributes of TF-IDF, TR and CV to specify the importance of each candidate term, respectively. The obtained values of these three attributes for each term are actually the corresponding weights measured by using TD-IDF, TextRank and C-Value.

Since TF-IDF and TextRank are initially used for measuring the significance of single-word term only, we have to adapt them for multi-word terms. To this end, we calculate the mean value of the importance values of all words appearing in a term for both, TF-IDF and TextRank, as follows.

$$TFIDF(t) = \frac{\sum_{i=1}^{n} TFIDF(w_i)}{n} \tag{6.1}$$

$$TR(t) = \frac{\sum_{i=1}^{n} TR(w_i)}{n} \tag{6.2}$$

where,

- $TFIDF(t)$ is the mean TF-IDF value of the multi-word term t, while $TR(t)$ denotes the mean TextRank value of the multi-word term t;
- n denotes the number of words in term t;
- $TFIDF(w_i)$ is the TF-IDF value of the word w_i in the requirements belonging to a particular feature; Note that $TFIDF(w_i)$ not only counts the term frequency of w_i appearing in all requirements of a particular feature, but also takes the occurrence of w_i in the requirements of other features into account;
- $TR(w_i)$ is the TextRank value of the word w_i in the requirements belonging to a particular feature.

Finally, CV is just applied to measure the weights of multi-word terms. Consequently, the CV value for a single-word term is 0.

RT: We use *Ratio of Terms* to indicate how many requirements belonging to a particular feature contain a certain candidate term, and we intend to know whether a candidate term occurs in the majority of the requirements in a feature or not. To this end, we use the equation below to calculate the RT value for each term:

$$RT(t) = \frac{m}{n} \tag{6.3}$$

where,

- m denotes the number of the requirements containing a specific term t of a particular feature;
- n denotes the number of all the requirements of this particular feature.

Labeling

After obtaining the attribute information for each term in a feature, we obtain the complete dataset by assigning *label* to each candidate term. Labels are used to denote whether a candidate term is a feature term or not. In order to provide accurate label information, we do the following steps:

a) We firstly take the feature name into account. If a candidate term is similar/related to the feature name, this candidate term can be regarded as a feature term labeled as 1.

b) Moreover, we further analyze the requirements belonging to the features. If we find that a candidate term can also provide pivotal information of the intention of a particular feature, we regard this candidate term as a feature term labeled as 1.

c) The rest of the candidate terms that do not satisfy a) and b) are not feature terms labeled as 0.

Table 6.3 shows the labeled candidate terms mined from Table 6.1. In the BCS feature model, the feature name of feature 1 is *LED Alarm System*, while the name of feature 2 is *Exterior Mirror Heating*.

6.1.2 Dataset Preprocessing

The original data need to be further processed before training in order to improve the performance of the trained model. In what follows, we will introduce why the data must be preprocessed and how to do the preprocessing.

Table 6.3: An example of the labeled data.

F-ID	Terms	PN	TF-IDF	TR	CV	ST	RT	Label
1	interior monitoring	0	0.2107	0.0353	2.0000	0	0.2857	1
	triggered alarm	0	0.3372	0.0653	1.0000	0	0.1429	1
	alarm system	0	0.2518	0.0567	1.0000	0	0.1429	1
	ignition key	0	0.2150	0.0528	1.0000	0	0.1429	0
	reset button	0	0.1492	0.0544	1.0000	0	0.1429	0
	LED	1	0.4396	0.1104	0.0000	0	1.0000	1
	light	0	0.3487	0.0519	0.0000	0	0.4286	0
	effect	0	0.0745	0.0419	0.0000	0	0.1429	0
2	exterior mirror heating	0	0.3985	0.0768	6.3399	0	0.5000	1
	exterior mirror	0	0.4438	0.0864	4.0000	1	0.8333	1
	exterior mirror heater	0	0.3544	0.0704	3.1699	0	0.3333	1
	minimum temperature	0	0.3510	0.0872	2.0000	0	0.3333	0
	maximum temperature	0	0.3402	0.0794	2.0000	0	0.3333	0
	temperature measurement	0	0.3071	0.0782	1.0000	0	0.1667	0
	LED	1	0.1606	0.0581	0.0000	0	0.3333	1
	switch	0	0.1539	0.0550	0.0000	0	0.1667	0
	vehicle	0	0.0634	0.0407	0.0000	0	0.1667	0

F-ID: Feature ID.

Attribute scaling

From Table 6.3, we can observe that the values of different attributes vary considerably in range and magnitudes. For some machine learning algorithms (e.g., KNN) using Euclidean distance to compute the distance of two points, the attributes with high magnitudes will have higher weights in the distance calculations than attributes with low magnitudes. This makes these machine learning algorithms pay more attention to the attributes with high magnitudes (e.g., CV), neglecting the attributes with low magnitudes (e.g., TR). In order to reduce this bias, we have to normalize these values to the same level of magnitudes.

To this end, we utilize Min-Max Scaling to normalize the range and magnitudes of three attributes (e.g., TF-IDF, TR and CV). By using Equation 6.4 below, each original value is converted into the corresponding normalized value between 0 and 1.

$$v' = \frac{v - min(v)}{max(v) - min(v)} \tag{6.4}$$

where,

- v' denotes the normalized value;
- v denotes the original value of a specific attribute in a particular feature;
- $min(v)$ and $max(v)$ denotes the minimum and maximum values of a specific attribute in a particular feature, respectively.

We use the candidate term – "interior monitoring" in Table 6.3 as an example to specify how v' can be computed, such as, we calculate the v' of its TF-IDF. The original TF-IDF value of "interior monitoring" is 0.2107. The minimum TF-IDF value in feature 1 is 0.0745, while the maximum TF-IDF value in feature 1 is 0.4396. Hence, the normalized value of TF-IDF for "interior monitoring" is 0.3730 (keep four decimal places).

Afterwards, the normalized data are fed into different machine learning algorithms for training.

6.1.3 Training Process

We utilize seven common and established ML algorithms to train the prediction models. This allows us to eventually compare the results among all of these algorithms, and thus, to identify the most suitable one(s). In order to achieve a convincing comparison, we use *K-Fold Cross-Validation* for performance evaluation and apply *Grid Search* for parameter selection for each machine learning algorithm.

K-Fold Cross-Validation

Usually, the subject dataset is separated into the training set and test set in the traditional machine learning process, where the former is applied to train the model and the latter is used to evaluate the performance of the trained model. However, the performance of the trained model is highly affected by how the dataset is split. As a result, the performance of the model may exhibit considerable differences if training and test datasets differ. This situation becomes even worse in case of a relatively small dataset. Moreover, splitting a dataset into just two subsets (for training and test, respectively) is also prone to lead to overfitting problem. In order to overcome these problems, we use *K-Fold Cross-Validation* that divides a dataset into k folds (i.e, k sets). Afterwards, the training and testing process is repeated k times, while at each time $k-1$ sets are used for training and the remaining set is applied for testing [RTL09]. Additionally, each set must be used at least once for training and once for testing. Finally, the performance of the trained model is the *mean value* of all k results.

Grid Search

For each machine learning algorithm, there are different hyperparameters that have to be specified. This step is crucial as these hyperparameters have a huge impact on the performance of the algorithm for a given problem. For example, assume that an algorithm A with a specific parameter setting outperforms algorithm B. However, when changing one parameter of the algorithm A, it might produce worse results than B. As a consequence, it may hinder to compare the performance of

different algorithms, if the optimal parameter selection for each algorithm is unknown. To overcome this problem, *Grid Search* has been proposed to tune the hyperparameters in order to select the optimal values for a machine learning algorithm [PVG+11, BBBK11]. In a nutshell, Grid Search is an exhaustive search based on a grid of parameters (i.e., a manually predefined subset of the hyperparameters). The performance of each combination of the hyperparameters is evaluated via K-Fold Cross-Validation using a certain performance metric (cf. Evaluation metric in Section 6.2.2). As a result of applying Grid Search, the combination of hyperparameters providing the highest performance value is considered being the best parameters, and thus, can be used for the respective ML algorithm. This way, we can ensure that we compare different algorithms with their best setup, and thus, that they produce the best prediction model (as far as the data allows).

Consequently, in our training process, we apply both K-Fold Cross-Validation and Grid Search to select hyperparameters, and thus, achieve the best prediction model for each machine learning algorithm.

6.2 Evaluation

In this section, we present the evaluation of our approach, using the seven ML algorithms introduced in the previous section. First, we propose three research questions regarding what we want to answer with this evaluation. Afterwards, we present the design of our experiment with details of datasets, parameters setting, and evaluation metrics. Finally, we present our results and discuss the performance and robustness of the prediction models.

6.2.1 Research Questions

The research demonstrated in this chapter is used to explore the answers of RQ3 mentioned in Chapter 1. To evaluate our proposed approach of combining rule-based with learning-based feature term extraction, we are interested in the following *sub-research questions*:

RQ3.1*: To what extent are the six attributes we propose for learning prediction models applicable to the task of feature term extraction?*

For RQ3.1, we intend to figure out whether the six attributes we identified make sense for training a prediction model. We apply seven widely used machine learning algorithms to train models on the six attributes in order to reduce the bias caused by only using one or two machine learning algorithms.

RQ3.2*: How accurate can our trained model predict (true) feature terms in another domain (i.e., on another dataset)?*

For RQ3.2, we intend to apply the prediction models trained from one labeled dataset in a particular domain to predict the feature terms in another domain. Particularly, we aim at evaluating the accuracy of extracted feature terms via *precision* and *recall*.

RQ3.3*: What can we learn from the comparison among seven different machine learning algorithms?*

For RQ3.3, we intend to gain insights into the comparison of seven different machine learning algorithms based on the performance for both training and test process, which can indicate the direction for future research.

6.2.2 Experiment Design

In order to answer the proposed research questions, we design a dedicated experiment as follows.

Dataset

The dataset we used to train the prediction models was created from the software requirement specifications of BCS (cf. Section 6.1.1).

Moreover, we utilized two other existing datasets, E-shop[1] and Digital Home [HT07], to test whether the prediction models trained on the BCS dataset can identify feature terms in different domains. Although E-Shop and Digital Home were developed by using a traditional software development approach, it is able to use automated feature extraction techniques to extract features from their requirements. However, the automatically extracted features are not completely accurate. To avoid the impact of inaccurate features from automated extraction on this experiment, we manually analyzed the 62 functional requirements of E-shop and the 38 functional requirements of Digital Home to identify features and created the mappings between features and requirements, respectively.

Experiment Process

We conducted this experiment by using our proposed approach as introduced in Section 6.1 on the BCS, Digital Home and E-shop datasets. Next, we briefly describe each step and several parameter settings.

1. In order to achieve the prediction models, we used the predefined rules based on POS patterns to extract 132 candidate terms from BCS requirements. Subsequently, we computed the six attributes of each term automatically. Next, based on the BCS feature model, we created a labeled dataset of the 132 candidate terms, with true feature terms labeled as 1 and all other terms labeled as 0. The implementation of the POS patterns was based on spaCy [HM18], which is an open-source library for advanced natural language processing.

2. As next step, we preprocessed the three attributes, TF-IDF, TR, and CV to scale the original values into a range between 0 and 1. As a result, we obtained the preprocessed labeled dataset.

3. Subsequently, we fed the preprocessed labeled data into seven different machine learning algorithms to train the prediction models. In this setup, our implementation was based on scikit-learn [PVG+11] that is an open-source toolkit for machine learning. For K-Fold Cross-Validation, we preset k as 3

[1] https://www.utdallas.edu/~chung/RE/Presentations07S/Team_1_Doc/Documents/SRS4.0.doc

and 10, since we intend to test whether different k values (i.e., 3-fold and 10-fold) affect the results or not. Moreover, it can also reduce the bias, compared to results with one k value only. In the process of Grid Search, we adjusted several key hyperparameters for each algorithm to achieve an optimal combination of them. In detail, 1) for LR, we selected three parameters, "penalty", "C" and "solver"; 2) for SVM, we adjusted "C", "kernel", "gamma"; 3) for DT, "criterion" and "splitter" were selected; 4) for KNN, "n_neighbors", "weights" and "algorithm" were adjusted; 5) For RF, we selected "n_estimators", "criterion" and "bootstrap" to be adjusted; 6) for GBN, we did not specify any hyperparameters, such as "priors" that was adjusted automatically according to the training data (cf. Table 6.4 for details); 7) for MLP, "activation" and "hidden_layer_sizes" were adjusted. For the rest of the parameters of each algorithm, we used the default values in scikit-learn. Moreover, in order to determine the best hyperparameter of each algorithm, we used F1 score (cf. Evaluation metric) to evaluate the performance of the trained model. As a result, we considered the hyperparameters of the trained model with the highest F1 score as best parameters, and thus, selected them for our evaluation. After the training process, we obtained the seven best prediction models, one for each ML algorithm.

4. Finally, we used the prediction models to identify feature terms in the E-shop and Digital Home dataset. In order to extract the candidate terms, we used the same rules as for BCS. Eventually, we obtained 89 and 138 candidate terms with the six attributes for each term from E-shop and Digital Home dataset, respectively. Moreover, we also assigned labels for each term in order to evaluate the prediction results.

Evaluation Metric

In order to allow for the quantitative analysis of the results, we use F1 score to determine the best prediction model for each algorithm. Moreover, we compute *precision*, *recall* and *F1 score* to evaluate the results of using our prediction models to predict feature terms from another domain and also to analyze the robustness in terms of F1 score.

F1 score. The F1 score is computed based on precision and recall as follows.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6.5}$$

$$Precision = \frac{The\ number\ of\ correctly\ predicted\ feature\ terms}{The\ number\ of\ all\ predicted\ feature\ terms} \tag{6.6}$$

$$Recall = \frac{The\ number\ of\ correctly\ predicted\ feature\ terms}{The\ number\ of\ all\ true\ feature\ terms} \tag{6.7}$$

where,

- *Predicted feature terms* denote the terms with a predicted label as 1 by using the trained model;
- *Correctly predicted feature terms* means the predicted label is right.
- *All true feature terms* denotes all the terms labeled as 1 in the test set.

Robustness. If a prediction model is *robust*, this model exhibits the property that the performance on the test dataset is close to the performance on the training dataset [XM12]. Since we mainly use F1 score to evaluate the performance, we simply apply the following equation to compute the robustness:

$$R = F1_{train} - F1_{test} \tag{6.8}$$

where,

- The closer R is to zero, the more robust the prediction model is;
- $F1_{train}$ denotes the best F1 score for training prediction models on the BCS dataset (cf. Table 6.4).
- $F1_{test}$ denotes the *average F1 score* for predicting the true feature terms from E-shop and Digital Home dataset, using the prediction models (cf. Table 6.5);

6.2.3 Results

Next, we present the results of our evaluation, based on our research questions. While for the sake of brevity, we just provide the main insights, all data is available in our supplementary material[2].

RQ3.1 - Appropriateness of attributes prediction model

We use the BCS dataset to create the prediction models for each machine learning algorithm. By using Grid Search, we obtain the best prediction models via 3-fold and 10 fold Cross-Validation for each algorithm with a certain parameter setting, measured by F1 score.

As our results in Table 6.4 reveal, most ML algorithms achieve a relatively high F1 score between 0.8 and 0.9. We argue that this not only indicates an acceptable setup regarding hyperparameters, but also that the six attributes we used for learning the model are appropriate for feature term extraction. As an exception, GBN only achieved an F1 score of $\approx$ 0.65 for both 3-fold and 10-fold, and thus, must be considered of rather low appropriateness. The reason may be that it can not learn the interactions between attributes.

Based on the results, we answer RQ3.1 as follows: Different prediction models trained on terms with the six attributes can be achieved. Although not all prediction models provide high performance, the six attributes we identified are suitable for training a model to predict true feature terms.

[2]https://zenodo.org/record/3577855#.XfdiqdOqQSM

Table 6.4: Results of the best parameter and F1 score in the training process.

Algorithm	k-fold	Best Parameter	Best F1 Score
LR	3-fold	'C': 0.1, 'penalty': l2, 'solver': liblinear	0.9002
	10-fold	'C': 0.1, 'penalty': l2, 'solver': liblinear	0.8918
SVM	3-fold	'C': 2.9, 'gamma': scale, 'kernel': sigmoid	0.8788
	10-fold	'C': 2.4, 'gamma': auto, 'kernel': rbf	0.8936
DT	3-fold	'criterion': entropy, 'splitter': random	0.8214
	10-fold	'criterion': gini, 'splitter': random	0.8424
KNN	3-fold	'algorithm': auto, 'n_neighbors': 6, 'weights': distance	0.8890
	10-fold	'algorithm': auto, 'n_neighbors': 8, 'weights': distance	0.9162
RF	3-fold	'bootstrap': True, 'criterion': entropy, 'n_estimators': 30	0.8683
	10-fold	'bootstrap': True, 'criterion': entropy, 'n_estimators': 35	0.8753
GBN	3-fold	–	0.6509
	10-fold	–	0.6529
MLP	3-fold	'hidden_layer_sizes': 20, 'activation': tanh	0.8792
	10-fold	'hidden_layer_sizes': 20, 'activation': tanh	0.8974

RQ3.2 – Apply the prediction models to the other datasets

After obtaining the prediction models, we applied these models to predict the true feature terms from E-Shop and Digital Home dataset. We show the results in Table 6.5.

Our data reveal that 1. the recall is generally slightly higher than the precision, 2. the F1 score only slightly deviates from the one in Table 6.4, and 3. that all models exhibit a relative high robustness

The first observation could indicate that our attributes, while applicable in general, still may contain some domain-dependent aspect that hinders a higher precision. However, given the second observation, the differences are really small, and thus, we argue that in general, these models are applicable across datasets. This is also supported by the third observation, which confirms that all models have high robustness, and thus, perform similarly on training and test set.

Table 6.5: Results of feature term prediction in Digital Home and E-shop.

Algorithm	k-fold	Dataset	Precision	Recall	F1 Score	Average F1 Score	R
LR	3-fold	Digital Home	0.6742	**0.9375**	0.7843	0.7789	0.1213
		E-shop	0.6833	**0.8913**	0.7736		
	10-fold	Digital Home	0.6742	**0.9375**	0.7843	0.7789	0.1128
		E-shop	0.6833	**0.8913**	0.7736		
SVM	3-fold	Digital Home	0.6905	0.9063	0.7838	0.7656	0.1132
		E-shop	0.6981	0.8043	0.7475		
	10-fold	Digital Home	0.6628	0.8906	0.7600	0.7427	0.1509
		E-shop	0.6607	0.8043	0.7255		
DT	3-fold	Digital Home	0.7013	0.8438	0.7660	0.7541	0.0672
		E-shop	0.7059	0.7826	0.7423		
	10-fold	Digital Home	0.6747	0.8750	0.7619	0.7521	0.0903
		E-shop	0.7059	0.7826	0.7423		
KNN	3-fold	Digital Home	0.6744	0.9063	0.7733	0.7540	0.1350
		E-shop	0.6923	0.7826	0.7347		
	10-fold	Digital Home	0.6824	0.9063	0.7785	0.7604	0.1558
		E-shop	0.7059	0.7826	0.7423		
RF	3-fold	Digital Home	**0.7403**	0.8906	**0.8085**	0.7754	0.0929
		E-shop	0.7059	0.7826	0.7423		
	10-fold	Digital Home	0.6905	0.9063	0.7838	0.7705	0.1047
		E-shop	0.6842	0.8478	0.7573		
GBN	3-fold	Digital Home	0.6939	0.5313	0.6018	0.6244	**0.0265**
		E-shop	**1.0000**	0.4783	0.6471		
	10-fold	Digital Home	0.6939	0.5313	0.6018	0.6244	0.0285
		E-shop	**1.0000**	0.4783	0.6471		
MLP	3-fold	Digital Home	0.7108	0.9219	0.8027	**0.8014**	0.0778
		E-shop	0.7755	0.8261	**0.8000**		
	10-fold	Digital Home	0.7108	0.9219	0.8027	0.7953	0.1021
		E-shop	0.7358	0.8478	0.7879		

Note: Average F1 Score = (F1 score for Digital Home + F1 score for E-shop) / 2

Similar to RQ3.1, the GBN algorithm is an outlier. While it has a very high precision (1.0) for E-shop dataset, its recall is very low for both E-shop and Digital Home dataset, which eventually results in a low F1 score. Hence, even though it leads to very good robustness, we consider this algorithm as not applicable to our kind of problem and dataset.

Based on the results in Table 6.5, we answer RQ3.2 as follows: Although the performance for predicting true feature terms in both E-Shop and Digital Home is moderate (0.75 ± 0.05), it provides domain engineers with a very good starting point to identify potential feature terms by using a pre-trained prediction model, especially taking the results of MLP into account.

RQ3.3 – Comparison of ML algorithms

According to Table 6.5, we can see that the RF prediction model trained by using 3-fold cross-validation achieves both the highest precision and F1 score for the Digital Home dataset, while its performance for the E-shop dataset decreases. That is to say, the RF prediction models lack the ability to achieve stable results for data from different domains. Although the majority of the prediction models suffer from the same problem, the MLP prediction model (3-fold) produces results of Digital Home and E-shop with the smallest difference, which indicates that it is of better generalization ability than the other models.

Further, our data reveal that the LR prediction model (3-fold and 10-fold) has the highest recall for both Digital Home and E-shop. However, the relatively low precision impedes a higher average for the F1 Score. In contrast, the MLP prediction model (3-fold) neither achieves the highest precision nor recall, but comes with a better trade-off between both measures. Consequently, the MLP prediction model (3-fold) achieves the highest average F1 score.

Moreover, according to the R value in Table 6.5, the models trained by using 3-fold cross-validation are slightly more robust than the models trained by using 10-fold. The reason is that 10-fold means there are more data for training and fewer data for testing, which easily leads to higher training performance than 3-fold models. That is to say, although the 10-fold models fit the training data better than 3-fold models, it results in the worse ability to predict the unknown data than 3-fold models in terms of average F1 score. However, the difference of average F1 scores from prediction models trained by using 3-fold and 10-fold is very small, especially for the probabilistic approaches (i.e., LR and GBN).

Given the insights above, we answer RQ3.3 as follows: The ability to achieve a suitable balance between precision and recall as well as the robustness of the prediction models is vital for using them to identify feature terms in different domains in practice. Obviously, the MLP prediction model (3-fold) is proven to be the best one in this experiment.

In summary, the supervised machine learning approach integrated with the predefined rules is capable of identifying true feature terms from requirements. However, different ML algorithms present different performance and robustness. In order to achieve the best prediction model, we make a comparison among seven ML algorithms and the neural-network-based MLP prediction model outperforms others.

Certainly, more experiments need to be conducted to prove this conclusion, for example, using more requirements from different domains to test the prediction models and retraining the models by using a training dataset of more requirements. We will focus on improving the current research in future work.

6.3 Threats to Validity

Construct validity. Our proposed feature terms extraction method is able to predict true feature related terms from the candidate term extracted by using certain rules based on POS patterns from requirements. Although the candidate terms are extracted by rules, we approach is actually not sensitive to how these candidate terms are extracted. That is to say, the prediction models can be trained on the candidate terms extracted by any other different methods, since the algorithms are focused on learning the six attributes of terms rather than the terms themselves. However, our prediction models can only be used to identify the true feature terms from the candidate terms, which means they cannot provide a true feature term that does not belong to the candidate terms.

Conclusion validity. The results are predicted by using the prediction models trained on a relatively small dataset. This might affect the accuracy of results, since a small dataset probably can not provide enough data for learning. However, the dataset we use for training a model is generated from a case study of real-world scenario including a feature model used as ground truth, which means we can achieve a training dataset of relatively high quality and reduce the bad effect of the small size of data. Certainly, only using Digital Home and E-shop dataset may be not enough to support the conclusions regarding the robustness of prediction models and applicability of prediction models in different domains.

6.4 Related Work

In this section, we categorize the research on terms extraction in SPL in terms of the basic methods they used: unsupervised methods and supervised methods.

Unsupervised Methods

Sree-Kumar et al. proposed a framework named FeatureX to automate the process of feature model generation by analyzing the terms extracted from requirements [SKPC18]. However, the main method they used for extracting terms is based on some extraction rules from research [ASBZ17] and [ASBZ16]. The idea behind these extraction rules is actually using existing natural language processing tools, such as NLTK [LB02] and OpenNLP to obtain text chunks following some predefined rules or POS patterns. Niu et al. extracted terms from requirements as conceptual information by using a two-word unit named lexical affinity proposed in research [MBK91]. However, they also generated the domain vocabulary for one-word unit rather than only extracting the two-word units which belong to verb-DO pairs. In Sree-Kumar et al.'s research, they directly regarded the terms extracted by rules as features and created feature models based on these terms, while Niu et al. used limited rules to identify the domain-specific terms. This may cause a

problem that for a large size of requirements, there must be a huge number of terms and the features created by the group of these terms lack the key information of the mapping between features and raw requirements. The enormous terms and the lack of key information may impede the applicability in practice. Moreover, just using one word weighting method (i.e., TF-IDF) in research [SKPC18] may lead to bias. In contrast, our approach focus on predicting the true feature terms from the candidate terms extracted from the requirements belonging to features. We take different attributes of each term containing three weighting methods into account to train prediction models.

Besides extracting feature related terms from requirements, there is also some research focusing on identifying terms information from other types of textual documents. Ferrari et al. also utilized POS patterns to extract candidate domain-specific terms from online product descriptions [FSD13]. And then, they applied C-NC value to weight the terms and predefined a threshold to filter the terms. Finally, the terms are re-ranked by using contrastive analysis. In addition, in research [BKSJ16] and [BKSH17], authors focused on extracting terms from online reviews. They also used POS patterns to identify candidate terms. However, the extracted terms will be grouped using clustering algorithms to form features. Compared with requirements, this kind of informal documents can only provide limited information of features.

Supervised Methods

Bagheri et al. utilized Named Entity Recognition (NER) technique to not only extract potential features, but also identify integrity constraints [BEG12]. They applied the conditional random field-based NER method to train a labeled dataset, resulting in a learned NER model that can be used to identify the features and integrity constraints. And, they directly labeled the raw requirements rather than identifying some specific attributes, which may lead to needing numerous labeled data to train a model of high performance. The reason is that the algorithm has to learn the characteristics of the terms by itself and adjusts the parameters and weights in the learning process. If the dataset is not large enough, the algorithm can not learn enough characteristic information of terms.

We can observe that POS patterns are widely used for extracting candidate features terms from textual documents in the context of SPL and the majority of the related works are based on unsupervised methods. Although we also apply POS patterns to extract candidate terms, the key difference is that we further analyze the potential attributes of these terms, which makes our approach insensitive to how the candidate terms are extracted. Moreover, we utilize supervised machine learning techniques to avoid defining many detailed rules that might be only suitable for certain cases.

6.5 Summary

In order to figure out the intentions of features, we intend to provide feature terms of high quality that contain pivotal information of functionality and what a feature is. To this end, we firstly extract all the candidate terms in terms of the predefined rules. Secondly, we identify six attributes of each term and generate a labeled dataset to train prediction models by using seven different machine learning algorithms.

Subsequently, we apply K-fold Cross-Validation and Grid Search to select the best prediction model for each algorithm based on F1 score. Finally, we utilize the prediction models trained on one dataset to predict feature terms from the other two datasets in the different domains. We analyze the performance and robustness of the prediction models and the results present that our approach is capable of identifying feature terms from requirements.

7. Automated Extraction of Domain Knowledge in Practice

This chapter is based on and shares material with the SPLC'20 paper "Automated Extraction of Domain Knowledge in Practice: The Case of Feature Extraction from Requirements at Danfoss" [LSSF20].

In previous chapters, we have shown that 1. Software Requirements Specifications (SRS) are suitable artifacts to extract such information (cf. Chapter 3) and 2. proposed approaches that make use of advanced Natural Language Processing (NLP) techniques to reliably extract features and variability from SRS (cf. Chapter 4, Chapter 5 and Chapter 6). 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 chapter, we address this problem by analyzing software requirements specifications 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. In particular, we apply Doc2Vec to train a language model on the requirements, use a clustering algorithm to group the similar requirements in terms of the information from the pre-trained language model and take advantage of feature terms prediction techniques (cf. Chapter 6) to present the key information of a particular feature. Moreover, we present a GUI that supports domain engineers to visualize and adjust the extracted features in practice.

We make the following contributions to answer RQ4:

- We present a study that applies automated (domain) feature extraction and feature tree creation on a large amount of real-world requirements, since the

ability to process massive requirements is necessary to make feature extraction techniques practicable in practice.

- 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 the technique in Chapter 5) combined with the technique in Chapter 6 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.

7.1 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 7.1, 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 7.1.1 and Section 7.1.2). 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. Chapter 6). Finally, we propose a GUI that can present the initial feature tree and feature terms, and thus, support domain engineers in revising the generated features.

Note: since the requirements from Danfoss are confidential, in this section we use requirements from Body Comfort System [LLLS13] and Digital Home [HT07] as running examples to illustrate our approach.

7.1.1 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 that describe 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

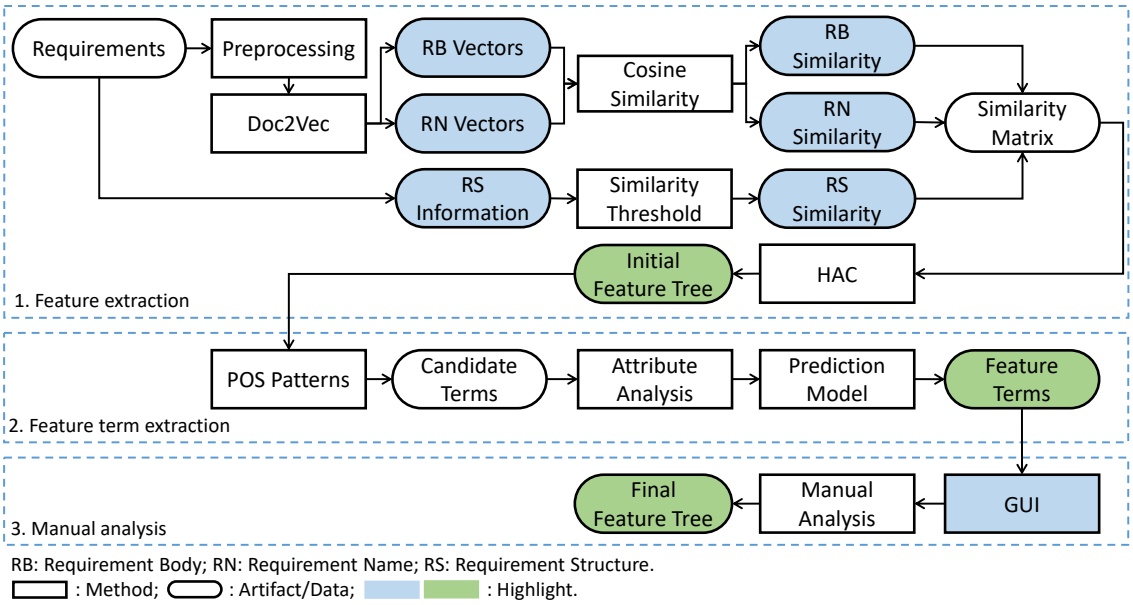


Figure 7.1: Overall workflow of our proposed approach for feature extraction.

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, tokenization, stop words removal, and lemmatization. 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.

7.1.2 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 *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. Then, we use HAC to create a feature tree based on these similarities.

Vectors of requirement bodies.

Doc2Vec is a neural-network-based, but unsupervised learning algorithm to generate a vector space of documents [LM14]. Thus, by using Doc2Vec, each document can be converted into a vector representation. One of the advantages of 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 achieve 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 in Danfoss, the name of a requirement usually comprise several words rather than only one word, and the average number of the words in the names is around three. In the process of analyzing the requirements in Danfoss, we find that although the length of a requirement name (i.e., the number of words in a requirement name) affect the similarity of each pair of the 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 Chapter 5, in which the length of a text will make a big difference to the similarity. In order to achieve the reasonable similarity of each pair of requirement names, we need to acquire the 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 [ZLT15].

$$Vector(rn) = \frac{\sum_{i=1}^{n} IDF(w_i) \times Vector(w_i)}{n} \tag{7.1}$$

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. In the same way, we can also compute the similarity between each pair of requirement names. Since Doc2Vec is a technique of distributional semantic models 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 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

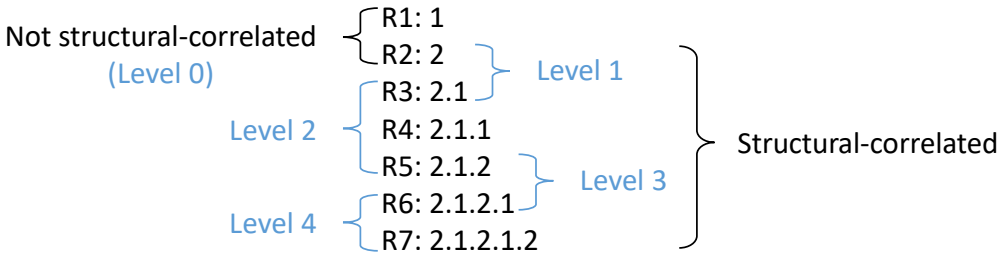


Figure 7.2: An example of structural similarity.

requirements. We use an example shown in Figure 7.2 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.

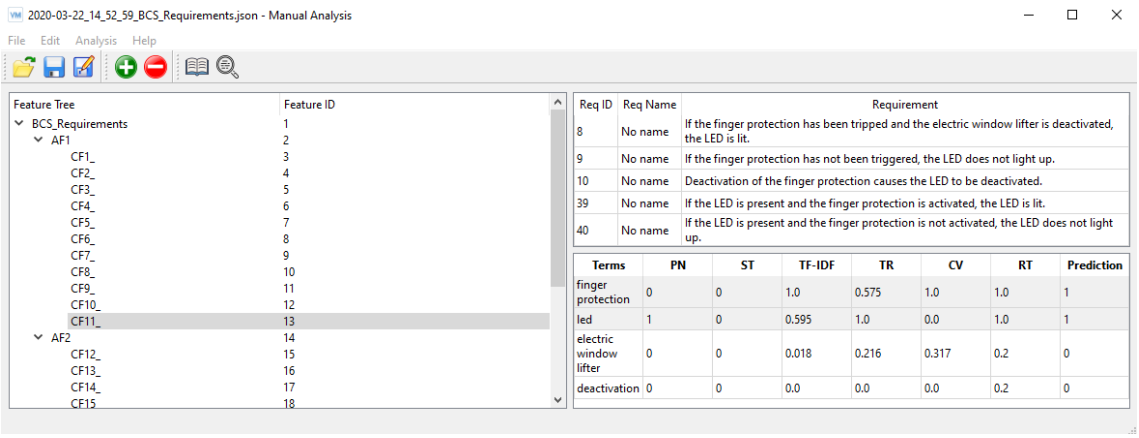
As a result, we compute the similarity of a pair of requirements using both, distributional semantic similarity and structural similarity, according to Equation 7.2. We compute this similarity for all pairs of requirements, and thus, obtain a similarity matrix of all the requirements in terms of Equation 7.3.

$$simReq(r_i, r_j) = wb \times SB(r_i, r_j) + wn \times SN(r_i, r_j) + ws \times SS(r_i, r_j) \tag{7.2}$$

$$simMatrix = \sum_{i=1}^{n} \sum_{j=1}^{n} simReq(r_i, r_j) \tag{7.3}$$

where,

- n is the number of requirements, while r denotes an individual requirement.



AF: Abstract feature; CF: Concrete feature; PN: Proper Noun; ST: Sub-Term; TF-IDF: Term Frequency – Inverse Document Frequency; TR: TextRank; CV: C-Value; RT: Ratio of Terms;

Figure 7.3: The GUI for manual analysis with a feature tree view, a list of corresponding requirements, and associated feature terms.

- $SB(r_i, r_j)$ 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 7.2.2).

- 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., $wb + wn + ws = 1$).

After achieving the similarity matrix, we rely on the same technique as we introduced in Chapter 5 to obtain the initial feature tree. Based on the result of feature extraction, we can achieve the Feature-Requirement mapping (i.e., F-R mapping) in terms of which we know which the requirements belong to a particular feature. Furthermore, we select a prediction model from Chapter 6 to identify the feature-related terms that can provide hints to understand the intentions of features. 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 7.3, 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.

7.2 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.

7.2.1 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 7.1.2.

Research Questions

For our study, in order to answer RQ4, we formulate the following sub-research questions.

RQ4.1*: What is the accuracy of the automatically extracted features?*

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

RQ4.2*: 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.

***RQ4.3**: 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.

7.2.2 Extraction Process

We performed feature extraction according to our technique, described in Section 7.1 on all of the 2 389 requirements. Next, we briefly describe each step and its output.

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, where we used the distributed memory version of paragraph vector and the vector size was 150. Based on the pre-trained 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 7.1.2, 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 7.2, wb, wn and ws are equal to 0.2, 0.2 and 0.6, respectively. According to Equation 7.3, a 2231×2231 similarity matrix was achieved.

3. As a third step, the similarity matrix was fed into HAC to extract the initial feature tree. In order to provide comparative results to show that the refined approach is capable of improving the accuracy, we also use the original VarMine introduced in Chapter 5 to analyze the requirements from Danfoss. Moreover,

the value of inconsistency threshold affects the granularity of the features. In order to achieve a moderate granularity and reasonable comparison, we set the inconsistency threshold to 1.1 for both the original VarMine and the refined approach. After HAC, we achieved 499 features resulting in the initial feature tree. Furthermore, we selected MLP prediction model (3-fold) (cf. Chapter 6) to identify feature terms.

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 7.2.3.

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

7.2.3 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 RQ4.1

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:

$$Precision = \frac{The\ number\ of\ meaningful\ features}{The\ number\ of\ extracted\ features} \tag{7.4}$$

Metric for RQ4.2

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:

$$Precision = \frac{The\ number\ of\ meaningful\ feature\ terms}{The\ number\ of\ extracted\ feature\ terms} \tag{7.5}$$

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

7.2.4 Results

In this section, we present the results and answer the research questions by means of both quantitative analysis and qualitative analysis. In particular, two domain engineers from Danfoss are involved in analyzing the results, providing particular perspectives on the Danfoss case in practice.

RQ4.1: the accuracy

In order to present a comparative result, we not only apply the refined approach (Approach I) proposed in this chapter, but also use the original VarMine (Approach II) in Chapter 5 to process the requirements to extract features, shown in Table 7.1. 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 7.1, 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 7.1) 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

Table 7.1: 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.

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 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 [HT07]:

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 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.

RQ4.1: Although the precision of the extracted features is not very high, three-quarters of the extracted features can be regarded as a reliable basis for engineers to further refine the true features. And, the additional information about the structure and name of requirements is beneficial to capturing the specialties of the requirements, thus improving the accuracy.

RQ4.2: hints from feature terms

We use the technique presented in Chapter 6 to identify feature-related terms from each extracted by using the refined approach (approach I). The number of the feature terms extracted from each feature ranges from 2 to 262. Overall, we identify 13078 feature terms from all meaningful features, and 8196 of them are meaningful, 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 of extracting feature terms into six categories according to the number of terms extracted from each feature. We use *No. of Terms* to denote the number of feature terms extracted from a feature. The six categories (C1-C6) are shown below:

C1: If $0 <$ No. of Terms $\leqslant 20$, the results belong to C1;

C2: If $20 <$ No. of Terms $\leqslant 40$, the results belong to C2;

C3: If $40 <$ No. of Terms $\leqslant 60$, 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;

Table 7.2: 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

From Table 7.2, 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 a relatively large number 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. That is to say, engineers do not need to check whether every feature term is meaningful and all they need to do is 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 helpful for analyzing the intention of features.

RQ4.2: 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.

RQ4.3: improvement in 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 [KCH+90]. 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 requirement analysis for extracting features, we intend to present the effectiveness of the combination of automated feature and key 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 to say, the weight (i.e., ws) for requirement structure is higher than another two. 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. Chapter 3) do not provide a tool for further visualizing and revising the extraction results, resulting in a lack of applicability in practice. A part of the previous researches used *FeatureIDE* [TKB+09] 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 7.3 presents the GUI with an example of the extraction results. The feature terms included in 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 find that these attributes also play a role in 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 notions 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 also 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 could not 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 in terms of functionality; 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 chapter).

RQ4.3: 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.

7.3 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 structural similarity into consideration. Although our similarity-based method might result in a bias for identifying features, it is usually a kind of parametric bias, since changing the parameters can affect the results of feature extraction. However, manual analysis is also prone to produce bias and different engineers can establish different feature models from different perspectives or personal experiences. Moreover, this manual bias can be hardly measured or parameterized, and thus, is out of control. The empirical evaluation results reveal that our proposed approach is capable of providing features from the perspective of similarity of a certain combination of parameters for domain engineers as a reference or starting point for generating the final feature model.

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 in Chapter 5 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 structural similarity to the distributional semantic similarity in the final similarity matrix. We preset the structural 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.

7.4 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.

7.4.1 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 [ASB+08]. Weston et al. proposed a tool named ArborCraft to identify features from requirements also based on applying LSA and HAC [WCR09]. 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 [KTWF12].

By contrast, we use neural-network-based technique, Doc2Vec that is an extension of word2vec [MSC+13], 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" [BDK14]. In terms of Baroni et al.'s research, prediction models have been proven to outperform common "count models" [BDK14]. 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. In addition, we not only compute the distributional semantic similarity of the requirements, but also take the structural information of the requirements into account.

7.4.2 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 [Wan15, Wan16], 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 [MSB+14] and WordNet [Mil95] to assist the process of analysis. Although the semantic information was identified, they did not propose a complete approach to use semantic frames to extract features in the SPL context. Itzik et al. analyzed the requirements based on SRL and parsed each sentence in the requirements in terms of different semantic roles they predefined [IRB14]. 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 [IRBW16], authors extended their prior research [RBIW14, IRB14] 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. In addition, requirement parsing is usually used to process the requirements with only one or two sentences. However, in the Danfoss case, one individual requirement may contain many sentences. It is unrealistic to parse the requirements with many sentences, since it is very hard to obtain uniform semantic roles.

7.4.3 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 [ACP+12, DGH+11, DDH+13, NBA+17] or online software reviews [BKSJ16, BKSH17]. They also used different techniques, such as K-means [BKSH17], Fuzzy C-Means [BKSJ16], Association Rule Mining [DDH+13], to aid feature extraction.

However, informal product descriptions and online software reviews just contain a very small part of the 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.

7.5 Summary

In this chapter, we refined and combined the approaches in Chapter 5 and Chapter 6 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 structural 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. However, The large size of the requirements and the lack of the mapping between requirements and variants lead to the fact that the method proposed in Chapter 5 cannot be used to extract the variability information in the Danfoss case. 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.

8. Conclusion and Future Work

In this chapter, we first conclude this thesis. Among them, we summarize the main approaches, results, and contributions of each chapter referring to the corresponding research questions proposed in the introduction. Then, based on the insights in our research, we discuss potential future work from different directions.

8.1 Conclusion

Natural language requirement documents contain the 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 an explicit mapping of features and variants to other artifacts. In order to realize a viable approach to automate the extraction process, this thesis is focused on the following four phases.

First, in order to systematically understand the research status in the field of feature and variability information extraction from natural language documents, we performed a systematic literature review (cf. Chapter 3), in which we investigate the technologies that have been used as well as their applicability, reliability and degree of automation. As a result, we achieve the first goal of this thesis. The obtained multi-dimensional overview and key insights of current research status not only form the contributions answering research question RQ1, but also result in an explicit guideline for our follow-up research.

Second, we present an initial self-learning structure to extract features (cf. Chapter 4). In particular, we apply Laplacian Eigenmaps, an unsupervised dimensionality reduction technique, to embed text requirements into compact binary codes. And, requirements are transformed into a matrix representation by looking up a pre-trained word embedding. Then, the matrix is fed into Convolutional Neural Network (CNN) to learn linguistic characteristics of the requirements. Furthermore, we train CNN by matching the output of CNN with the pre-trained binary codes. We conduct preliminary experiments and discuss the results. Although the self-learning structure has not yet reached a usable accuracy, it has the potential to

realize automatic information capture. Moreover, it provides valuable experience for the following research. We show a feasible approach called VarMine to extract features and variation points from software requirements specifications (cf. Chapter 5). We integrate probabilistic relevance and neural word embedding techniques to achieve the similarity of each pair of requirements. Then, the hierarchical clustering is used to group features, and we utilize heuristic and recognizing textual entailment based method to detect variation points between identified features. We perform a case study to evaluate the usability and robustness of VarMine and to compare it with the results of other related approaches. The results show that VarMine not only has the ability to accurately extract features, but also recognizes meaningful variability information. In summary, we achieve the second goal of this thesis and our contributions answer the RQ2.

Third, we present an approach to train prediction models by using machine learning techniques to identify feature terms, since feature terms as the smallest units in a feature can be regarded as vital indicators for describing a feature (cf. Chapter 6). To this end, we extract the candidate terms from requirement specifications in one domain and take six attributes of each term into account to create a labeled dataset. Subsequently, we apply seven commonly used machine algorithms to train prediction models on the labeled dataset. We then use these prediction models to predict feature terms from the requirements belonging to the other two different domains. Our results show that a) feature terms can be predicted with high accuracy within a domain; b) prediction across domains leads to a decreased but still good accuracy; and c) machine learning algorithms perform differently. This answers the RQ3 regarding figuring out the intention of an extracted feature.

Fourth, we integrate the technologies presented in Chapter 5 and Chapter 6, and apply to analyze the requirements from Danfoss (cf. Chapter 7). In order to solve the specialities of the Danfoss requirements, we not only use Doc2Vec to obtain the similarity of the requirement name and body, but also take the structure of the requirements into account. Moreover, we utilized feature terms prediction techniques to provide key information to domain engineers for further analyzing the extraction results. In particular, we developed a GUI to visualize the entire process for supporting to analyze the extracted features. We empirically demonstrate the accuracy of the results of applying the combined approach for analyzing the real-world software requirements specifications, and discusses how the proposed approach can improve the extraction process. As a consequence, we argue the integration is a reasonable structure that can be used to provide a starting point for extracting features from legacy requirements documents, thereby answering RQ4.

8.2 Future Work

We provide an overview of potential directions for future research. We identify and discuss not only feature work on automatically generating a feature model in domain engineering but also open challenges related to using extracted domain knowledge in application engineering.

Feature descriptions

Features refer to the abstraction of domain knowledge. A complete feature contains a lot of potential information. In our current research, we have applied different techniques to extract features and the corresponding requirements, and we provide feature-related terms to refer to the intention or name of each extracted feature. In future work, we plan to apply automatic text summarization technology to generate a summary of the requirements documents corresponding to the feature, which can be regarded as a description of the extracted feature. Automatic text summarization technology can shorten a long text to form an accurate summary that can contain concise and important information [GG17], which can benefit extracting features in tremendous requirements. The reason is that in the feature extraction with numerous requirements, an accurate and fluent summary of the requirements in each feature can allow the domain engineer to quickly understand the meaning of the extracted feature. Moreover, compared with the extracted feature terms, a summary of the requirements in a feature can more fluently convey and pass the intended information to domain engineers. In addition, automatic text summarization technology can also be applied in the process of the initial feature extraction. Therefore, in future work, we will pay more attention to the application of automatic text summarization technology in the field of software product lines.

Different artifacts

Currently, we are focusing on extracting features and variation points from natural language requirements. However, in the process of software development and evolution, there exist many artifacts, such as requirements, user cases, source code, class diagrams, etc. These artifacts not only describe the functionalities of a software system, but they are also related to each other. Therefore, all these artifacts contain potential information about commonality and variability, which can help improve the extraction process. For example, feature location technology supports identifying the initial location of functionality in the source code of a software system. It also analyzes source code comments that are a type of textual information to obtain a mapping between source code of the implementation of a function and textual description of the function [DRGP13]. Therefore, in future work, we plan to analyze different artifacts to extract features and variation pints to obtain more comprehensive information. Moreover, the information extracted from different artifacts can complement and be connected with each other, which can not only improve the accuracy of extracting feature and variability information but also form a feature model with a more comprehensive perspective than extracting information only from requirements.

Product configuration

As presented in previous chapters, natural language processing techniques are beneficial to extracting features and variation points from textual requirements in domain engineering of software product lines. We are considering applying natural language processing techniques in other phases of software product lines. For instance, in application engineering, engineers need to analyze the requirements from a particular customer to select existing features to derive a particular variant that

satisfies the customer's needs from an existing software product line. Although experienced engineers are involved in the activity of the analysis, this process is still time-consuming, especially confronted with numerous requirements. In future work, we plan to improve the feature extraction technology to automatically learn the domain knowledge of an existing product line. The learned domain knowledge can be used to analyze customers' needs to improve the efficiency of engineers in feature selection and product configuration.

Bibliography

[ABKS13] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, 2013. (cited on Page 1, 5, 6, 7, 8, and 9)

[ACP+12] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On extracting feature models from product descriptions. In *Proceedings of the International Workshop on Variability Modeling of Software-Intensive Systems*, pages 45–54. ACM, 2012. (cited on Page 23, 32, and 115)

[AK09] Sven Apel and Christian Kästner. An overview of feature-oriented software development. *Journal of Object Technology*, 8:49–84, 2009. (cited on Page 9)

[AM09] Ion Androutsopoulos and Prodromos Malakasiotis. A survey of paraphrasing and textual entailment methods. *Journal of Artificial Intelligence Research*, 38:135–187, 2009. (cited on Page 59)

[ANAV10] Vander Alves, Niu Niu, Carina Alves, and George Valença. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52:806–820, 2010. (cited on Page 37)

[ASB+08] Vander Alves, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummler. An exploratory study of information retrieval techniques in domain analysis. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 67–76. IEEE, 2008. (cited on Page 13, 23, 40, 48, 76, and 114)

[ASBZ16] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Extracting domain models from natural-language requirements: Approach and industrial evaluation. In *Proceedings of the Conference on Model Driven Engineering Languages and Systems*, pages 250–260. ACM, 2016. (cited on Page 95)

[ASBZ17] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10):918–945, 2017. (cited on Page 95)

[BAPM15] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 632–642. ACL, 2015. (cited on Page 16)

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the International Conference on Neural Information Processing Systems*, page 2546–2554. Curran Associates Inc., 2011. (cited on Page 88)

[BDK14] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the International Conference on Association for Computational Linguistics (ACL)*, pages 238–247. ACL, 2014. (cited on Page 40, 49, and 115)

[BEG12] Ebrahim Bagheri, Faezeh Ensan, and Dragan Gasevic. Decision support for the software product line domain engineering lifecycle. *Automated Software Engineering*, 19:335–377, 2012. (cited on Page 23, 80, and 96)

[BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, pages 135–146, 2016. (cited on Page 14)

[BKS15] Hasrina Bakar Bakar, Zarinah M. Kasirun, and Norsaremah Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, 106:132–149, 2015. (cited on Page 18 and 37)

[BKSH17] Noor Hasrina Bakar, Zarinah Mohd Kasirun, Norsaremah Salleh, and Azni Haslizan Ab Halim. Extracting software features from online reviews to demonstrate requirements reuse in software engineering. In *Proceedings of the International Conference on Computing and Informatics*, pages 184–190. Sintok: School of Computing, 2017. (cited on Page 24, 79, 80, 96, and 115)

[BKSJ16] Noor Hasrina Bakar, Zarinah M. Kasirun, Norsaremah Salleh, and Hamid A. Jalab. Extracting features from online software reviews to aid requirements reuse. *Journal of Applied Soft Computing*, 49:1297–1315, 2016. (cited on Page 24, 79, 80, 96, and 115)

[BLB+14] Luca Bigliardi, Michele Lanza, Alberto Bacchelli, Marco DAmbros, and Andrea Mocci. Quantitatively exploring non-code software artifacts. In *Proceedings of the International Conference on Quality Software*, pages 286–295. IEEE, 2014. (cited on Page 10)

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. (cited on Page 42)

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the International Workshop on Variability Modeling of Software-Intensive Systems*, pages 1–8. ACM, 2013. (cited on Page 37)

[CH74] Tadeusz Caliński and JA Harabasz. A dendrite method for cluster analysis. *Journal of Communications in Statistics*, 3(1):1–27, 1974. (cited on Page 56)

[CHN19a] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Modelling equivalence classes of feature models with concept lattices to assist their extraction from product descriptions. *Journal of Systems and Software*, 152:1–23, 2019. (cited on Page 24)

[CHN19b] Jessie Carbonnel, Marianne Huchard, and Clémentine Nebut. Towards complex product line variability modelling: Mining relationships from non-boolean descriptions. *Journal of Systems and Software*, 156:341–360, 2019. (cited on Page 24)

[CN01] Paul C. Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001. (cited on Page 1)

[CZZM05] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 31–40. IEEE, 2005. (cited on Page 22, 33, and 75)

[DDH+13] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Proceedings of the European Software Engineering Conference/Foundations of Software Engineering (ESECFSE)*, pages 290–300. ACM, 2013. (cited on Page 23 and 115)

[DGH+11] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the International Conference on Software Engineering (ICSE)*, page 181–190. ACM, 2011. (cited on Page 23, 32, and 115)

[DRB+13] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of

cloning in industrial software product lines. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 25–34. IEEE, 2013. (cited on Page 2)

[DRGP13] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: A taxonomy and survey. *Journal of of Software: Evolution and Process*, 25:53–95, 2013. (cited on Page 2, 37, and 119)

[DRSZ13] Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. *Recognizing Textual Entailment: Models and Applications*. Morgan & Claypool Publishers, 2013. (cited on Page 15)

[Dum04] Susan T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, 2004. (cited on Page 14)

[Dun74] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974. (cited on Page 56)

[FAT98] Katerina T. Frantzi, Sophia Ananiadou, and Junichi Tsujii. The c-value/nc-value method of automatic recognition for multi-word terms. In *Proceedings of the European Conference on Research and Advanced Technology for Digital Libraries*, pages 585–604. Springer, 1998. (cited on Page 84)

[FFGS18] Alessandro Fantechi, Alessio Ferrari, Stefania Gnesi, and Laura Semini. Requirement engineering of software product lines: Extracting variability using nlp. In *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, pages 418–423, 2018. (cited on Page 24)

[FGS17] Alessandro Fantechi, Stefania Gnesi, and Laura Semini. Ambiguity defects as variation points in requirements. In *Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems*, page 13–19. ACM, 2017. (cited on Page 24 and 77)

[FSD13] Alessio Ferrari, Giorgio O. Spagnolo, and Felice Dell'Orletta. Mining commonalities and variabilities from natural language documents. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 116–120. ACM, 2013. (cited on Page 23, 77, and 96)

[FSGD15] Alessio Ferrari, Giorgio O. Spagnolo, Stefania Gnesi, and Felice Dell'Orletta. Cmt and fde: Tools to bridge the gap between natural language documents and feature diagrams. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 402–410. ACM, 2015. (cited on Page 24 and 32)

[FSK+16] Thomas Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. Ten years of product line engineering at danfoss: Lessons learned and way ahead. In *Proceedings of the International*

Systems and Software Product Line Conference (SPLC), pages 252–261. ACM, 2016. (cited on Page 1)

[GDN13] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Proceedings of the International Conference on Speech Communication Association*, pages 1756–1760. ISCA, 2013. (cited on Page 45)

[GG17] Mahak Gambhir and Vishal Gupta. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47:1–66, 2017. (cited on Page 119)

[GGN+18] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. Allennlp: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6. ACL, 2018. (cited on Page 64)

[HCHM+13] Negar Hariri, Carlos Castro-Herrera, Mehdi Mirakhorli, Jane Cleland-Huang, and Bamshad Mobasher. Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 39:1736–1752, 2013. (cited on Page 23)

[HG11] Julian PT Higgins and Sally Green. *Cochrane Handbook for Systematic Reviews of Interventions*. John Wiley & Sons, 2011. (cited on Page 18)

[HJ15] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1373–1378. ACL, 2015. (cited on Page 54)

[HM18] Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2018. (cited on Page 64 and 89)

[HT07] Thomas B. Hilburn and Massood Towhidnejad. A case for software engineering. In *Proceedings of the International Conference on Software Engineering Education and Training*, pages 107–114. IEEE, 2007. (cited on Page 62, 89, 100, and 109)

[HW15] Mostafa Hamza and Robert J. Walker. Recommending features and feature relationships from requirements documents for software product lines. In *Proceedings of the International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 25–31. IEEE, 2015. (cited on Page 23, 33, and 40)

[HZS+16] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. Preprocessor-based variability in open-source and industrial software systems: An empirical study. 21(2):449–482, 2016. (cited on Page 1)

[IRB14] Nili Itzik and Iris Reinhartz-Berger. Generating feature models from requirements: Structural vs. functional perspectives. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 44–51. ACM, 2014. (cited on Page 23, 33, 40, 47, 71, 76, and 115)

[IRBW16] Nili Itzik, Iris Reinhartz-Berger, and Yair Wand. Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. *IEEE Transactions on Software Engineering*, 42:687–706, 2016. (cited on Page 24, 33, 40, 47, 76, and 115)

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988. (cited on Page 56)

[KAT16] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. Explaining anomalies in feature models. In *Proceedings of the International Conference on Generative Programming: Concepts and Experiences*, page 132–143. ACM, 2016. (cited on Page 58)

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv e-prints*, Dec 2014. (cited on Page 45)

[KBA15] Amal Khtira, Anissa Benlarabi, and Bouchra El Asri. Detecting feature duplication in natural language specifications when evolving software product lines. In *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 257–262. IEEE, 2015. (cited on Page 23 and 33)

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990. (cited on Page 1, 6, 9, and 111)

[KG09] Mahvish Khurum and Tony Gorschek. A systematic review of domain analysis solutions for product lines. *Journal of Systems and Software*, 82:1982–2003, 2009. (cited on Page 37)

[KGB14] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. 2014. (cited on Page 43)

[Kim14] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 1746–1751. ACL, 2014. (cited on Page 63)

[Kit07] Barbara A. Kitchenham. Guidelines for performing systematic literature reviews in software engineering. In *EBSE Technical Report*, 2007. (cited on Page 18)

[Kru01] Charles W. Krueger. Easing the transition to software mass customization. In *Proceedings of the International Workshop on Software Product-Family Engineering*, pages 282–293. Springer, 2001. (cited on Page 1 and 2)

[KTWF12] Kentaro Kumaki, Ryosuke Tsuchiya, Hironori Washizaki, and Yoshiaki Fukazawa. Supporting commonality and variability analysis of requirements and structural models. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 115–118. ACM, 2012. (cited on Page 13, 23, 40, 48, 77, and 114)

[LB02] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 63–70. ACL, 2002. (cited on Page 95)

[LGL+10] Liana Barachisio Lisboa, Vinicius Cardoso Garcia, Daniel Lucrédio, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira, and Renata Pontin de Mattos Fortes. A systematic review of domain analysis tools. *Information and Software Technology*, 52:1–13, 2010. (cited on Page 37)

[Li18] Yang Li. Feature and variability extraction from natural language software requirements specifications. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, page 72–78. ACM, 2018. (cited on Page 39 and 49)

[LKYW05] Luying Liu, Jianchu Kang, Jing Yu, and Zhongliang Wang. A comparative study on unsupervised feature selection methods for text clustering. In *Proceedings of the National Conference on Natural Language Processing and Knowledge Engineering*, pages 597–601. IEEE, 2005. (cited on Page 64)

[LLLS13] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. Delta-oriented software product line test models - the body comfort system case study. Technical report, TU Braunschweig, 2013. (cited on Page 45, 62, 81, and 100)

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning*, pages II–1188–II–1196. JMLR.org, 2014. (cited on Page 101)

[LSR07] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag, 2007. (cited on Page 5 and 8)

[LSS17] Yang Li, Sandro Schulze, and Gunter Saake. Reverse engineering variability from natural language documents: A systematic literature review. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 133–142. ACM, 2017. (cited on Page 17 and 22)

[LSS18a] Yang Li, Sandro Schulze, and Gunter Saake. Extracting features from requirements: Achieving accuracy and automation with neural networks. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering*, pages 477–481, March 2018. (cited on Page 39)

[LSS18b] Yang Li, Sandro Schulze, and Gunter Saake. Reverse engineering variability from requirement documents based on probabilistic relevance and word embedding. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 121–131. ACM, 2018. (cited on Page 49)

[LSSF20] Yang Li, Sandro Schulze, Helene Hvidegaard Scherrebeck, and Thomas Sorensen Fogdal. Automated extraction of domain knowledge in practice: The case of feature extraction from requirements at danfoss. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 1–11. ACM, 2020. (cited on Page 99)

[LSX20] Yang Li, Sandro Schulze, and Jiahua Xu. Feature terms prediction: A feasible way to indicate the notion of features in software product line. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 90–99. ACM, 2020. (cited on Page 79)

[LZ11] Yuanhua Lv and Chengxiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 7–16. ACM, 2011. (cited on Page 52, 54, and 64)

[MBBA16] Mariem Mefteh, Nadia Bouassida, and Hanêne Ben-Abdallah. Mining feature models from functional requirements. *The Computer Journal*, 59:1784–1804, 2016. (cited on Page 24)

[MBK91] Yoelle Maarek, Daniel Berry, and Gail Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17:800–813, 1991. (cited on Page 95)

[Mil95] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. (cited on Page 40 and 115)

[MM08] Marie-Catherine De Marneffe and Christopher D. Manning. Stanford typed dependencies manual, 2008. (cited on Page 12)

[MSB+14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the International Conference on Association for Computational Linguistics (ACL)*, pages 55–60, 2014. (cited on Page 40 and 115)

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the National Conference on Neural Information Processing Systems (NIPS)*, pages 3111–3119. Curran Associates, Inc., 2013. (cited on Page 14, 51, 52, 63, and 114)

[MT04] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 404–411. ACL, 2004. (cited on Page 84)

[NBA+17] Sana Ben Nasr, Guillaume Bécan, Mathieu Acher, João Bosco Ferreira Filho, Nicolas Sannier, Benoit Baudry, and Jean-Marc Davril. Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software*, 124:82–103, 2017. (cited on Page 24, 32, 77, and 115)

[NSN+14] Nan Niu, Juha Savolainen, Zhendong Niu, Mingzhou Jin, and Jing-Ru C. Cheng. A systems approach to product line requirements reuse. *IEEE Systems Journal*, 8:827–836, 2014. (cited on Page 23)

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005. (cited on Page 1, 5, and 6)

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. ACL, 2014. (cited on Page 14)

[PTDU16] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2249–2255. ACL, 2016. (cited on Page 59)

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (cited on Page 64, 88, and 89)

[RBIW14] Iris Reinhartz-Berger, Nili Itzik, and Yair Wand. Analyzing variability of software product lines using semantic and ontological considerations. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 150–164. Springer, 2014. (cited on Page 23, 39, 47, 75, 76, and 115)

[RBSA19] Iris Reinhartz-Berger, Ilan Shimshoni, and Aviva Abdal. Behavior-derived variability analysis: Mining views for comparison and evaluation. In *CAiSE*, pages 675–690. Springer International Publishing, 2019. (cited on Page 24)

[RBWH15] Iris Reinhartz-Berger and Ora Wulf-Hadash. Improving the management of product lines by performing domain knowledge extraction and cross product line analysis. *Information and Software Technology*, 59:191–204, 2015. (cited on Page 23)

[RM05] Lior Rokach and Oded Maimon. Clustering methods. In *Data Mining and Knowledge Discovery Handbook*, pages 321–352. Springer US, 2005. (cited on Page 55)

[RR03] C. Riva and C. Del Rosso. Experiences with software product family evolution. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 161–169. IEEE, 2003. (cited on Page 2)

[ŘS10] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the International LREC Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA, 2010. (cited on Page 45 and 64)

[RTL09] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009. (cited on Page 87)

[RZ09] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. (cited on Page 52 and 54)

[SG03] Alexander Strehl and Joydeep Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2003. (cited on Page 64 and 65)

[SKPC18] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. Extracting software product line feature models from natural language specifications. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, pages 43–53. ACM, 2018. (cited on Page 24, 79, 95, and 96)

[SWY75] Gerard M Salton, Andrew Wong, and Chungshu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. (cited on Page 14)

[TCO00] Peter Toft, Derek Coleman, and Joni Ohta. A cooperative model for cross/divisional product development for a software product line. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 111–132. Springer, 2000. (cited on Page 1)

[TKB+09] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: A tool framework for feature-oriented software development. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 611–614. IEEE, 2009. (cited on Page 64 and 112)

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. (cited on Page 64)

[Wan15] Yinglin Wang. Semantic information extraction for software requirements using semantic role labeling. In *Proceedings of the Conference on Progress in Informatics and Computing (PIC)*, pages 332–337. IEEE, 2015. (cited on Page 24, 40, 47, 76, and 115)

[Wan16] Yinglin Wang. Automatic semantic analysis of software requirements through machine learning and ontology approach. *Journal of Shanghai Jiaotong University*, 21:692–701, 2016. (cited on Page 24, 40, 47, 76, and 115)

[WCR09] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. A framework for constructing semantically composable feature models from natural language requirements. *Proceedings of the International Software Product Line Conference (SPLC)*, pages 211–220, 2009. (cited on Page 13, 23, 32, 40, 48, 71, 77, and 114)

[WRH+12] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer, 2012. (cited on Page 20, 22, and 25)

[XM12] Huan Xu and Shie Mannor. Robustness and generalization. *Machine Learning*, 86:391–423, 2012. (cited on Page 91)

[YWYL13] Yue Yu, Huaimin Wang, Gang Yin, and Bo Liu. Mining and recommending software features across multiple web repositories. In *Proceedings of the Asia-Pacific Symposium on Internetware*, pages 1–9. ACM, 2013. (cited on Page 23)

[ZLT15] Jiang Zhao, Man Lan, and Junfeng Tian. Ecnu: Using traditional similarity measurements and word embedding for semantic textual similarity estimation. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*, pages 117–122. ACL, 2015. (cited on Page 102)

[ZWCL10] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *Proceedings of the International Conference on Research and Development in Information Retrieval*, pages 18–25. ACM, 2010. (cited on Page 43)

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 30.10.2020

Yang Li