Mälardalen University Licentiate Thesis 306

Requirements-Level Reuse Recommendation and Prioritization of Product Line Assets

Muhammad Abbas

MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Licentiate Theses
No. 306

REQUIREMENTS-LEVEL REUSE RECOMMENDATION AND PRIORITIZATION OF PRODUCT LINE ASSETS

Muhammad Abbas

2021

School of Innovation, Design and Engineering

ISBN 978-91-7485-504-3
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

Software systems often target a variety of different market segments. Targeting varying customer requirements requires a product-focused development process. Software Product Line (SPL) engineering is one possible approach based on reuse rationale to aid quick delivery of quality product variants at scale. SPLs reuse common features across derived products while still providing varying configuration options. The common features, in most cases, are realized by reusable assets. In practice, the assets are reused in a clone-and-own manner to reduce the upfront cost of systematic reuse. Besides, the assets are implemented in increments, and requirements prioritization also has to be done. In this context, the manual reuse analysis and prioritization process become impractical when the number of derived products grows. Besides, the manual reuse analysis process is time-consuming and heavily dependent on the experience of engineers.

In this licentiate thesis, we study requirements-level reuse recommendation and prioritization for SPL assets in industrial settings. We first identify challenges and opportunities in SPLs where reuse is done in a *clone-and-own* manner. We then focus on one of the identified challenges: requirements-based SPL assets reuse and provide automated support for identifying reuse opportunities for SPL assets based on requirements. Finally, we provide automated support for requirements prioritization in the presence of dependencies resulting from reuse.

Sammanfattning

Programvarusystem riktar sig ofta till en mängd olika marknadssegment. Uppfyllandet av olika kundkrav kräver ofta en produktfokuserad utvecklingsprocess. Software Product Line (SPL)-tekniker är en möjlig lösning baserad på återanvändning för att underlätta snabb leverans av produktvarianter i stor skala med hög kvalitet. SPLer återanvänder funktionalitet från tidigare produkter och möjliggör samtidigt varierande konfigurationer. De vanligaste funktionerna realiseras i de flesta fall av återanvändbara tillgångar. I praktiken återanvänds tillgångarna på ett "clone-and-own"-manér för att minska de initiala kostnaderna för systematisk återanvändning. Dessutom implementeras tillgångarna i steg, och kravprioritering måste också göras. I detta sammanhang blir manuell analys och prioritering av återanvändning opraktisk när antalet härledda produkter växer. Dessutom är den manuella analysen av återanvändning tidskrävande och starkt beroende av ingenjörernas erfarenhet.

I den här licentiatavhandlingen studerar vi rekommendation för återanvändning och prioritering av SPL-tillgångar i industriella miljöer. Vi identifierar först utmaningar och möjligheter i SPL där återanvändning sker på ett *clone-and-own*-sätt. Vi fokuserar sedan på en av de identifierade utmaningarna: kravbaserad återanvändning och tillhandahåller automatiskt stöd för att identifiera återanvändningsmöjligheter för SPL-tillgångar baserat på krav. Slutligen fokuserar vi på kravprioritering i närvaro av beroende beroende på återanvändning.

To my parents

Acknowledgments

I would like to thank my main advisor, Prof. Daniel Sundmark, for the kind support and constructive feedback throughout the thesis. Thanks to my co-advisor, Dr. Eduard Paul Enoiu, for all the help in the focus groups and study designs. Many thanks to my co-advisor, Dr. Mehrdad Saadatmand, for those short but effective coffee breaks which shaped most of the research articles. Also, many thanks for teaching me some good words in Persian. I would also like to thank all my co-authors and collaborators for their contributions and support.

I have been fortunate to have worked on real industrial problems at Alstom (formerly Bombardier). This was made possible by the support of Claes Lindskog, and Daran Smalley. I would also like to thank Dr. Raluca Marinescu, Max Johansson, Jörgen Ekefjial, and all the Power Propulsion Control Software team at Alstom for their participation in the focus group sessions.

RISE Research Institutes of Sweden has always been at the center of all these collaborations with Alstom. I would like to thank my managers at RISE (at different times), Prof. Markus Bohlin, Larisa Rizvanovic, Stig Larsson, and Karolina Winbo, for all the support. Also, I would like to thank Tomas Olsson, Mats Tallfors, and the AI team at RISE for the feedback on the experiment designs. Many thanks to my academic sister Mahshid Helali Moghadam for the fun talks during the coffee breaks. Also, special thanks to all my fellow Ph.D. students at MDH.

The work presented in this thesis is funded by the Swedish Knowledge Foundation through the ARRAY industrial school and Vinnova through the eXcellence In Variant Testing (XIVT) project.

Muhammad Abbas, Västerås, March 2021

List of Publications

Papers included in this thesis[1]

Paper A: Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, Daniel Sundmark. "*Product Line Adoption in Industry: An Experience Report from the Railway Domain.*" In the 24th International Systems and Software Product Line Conference (SPLC 2020).

Paper B: Muhammad Abbas, Mehrdad Saadatmand, Eduard Paul Enoiu, Daniel Sundmark, Claes Lindskog. "*Automated Reuse Recommendation of Product Line Assets based on Natural Language Requirements.*" In the 19th International Conference on Software and Systems Reuse (ICSR 2020).

Paper C: Muhammad Abbas, Alessio Ferrari, Anas Shatnawi, Eduard Paul Enoiu, Mehrdad Saadatmand. "*Is Requirements Similarity a Good Proxy for Software Similarity? An Empirical Investigation in Industry*" In the 27th International Working Conference on Requirement Engineering: Foundation for Software Quality (REFSQ 2021).

Paper D: Muhammad Abbas, Irum Inayat, Naila Jan, Mehrdad Saadatmand, Eduard Paul Enoiu, Daniel Sundmark. "*MBRP: Model-based Requirements Prioritization Using PageRank Algorithm*" In the 26th Asia-Pacific Software Engineering Conference (APSEC 2019).

[1] The included papers have been reformatted to comply with the thesis layout.

Related publications, not included in this thesis

Paper W: Muhammad Abbas, Irum Inayat, Mehrdad Saadatmand, Naila Jan. "*Requirements dependencies-based test case prioritization for extra-functional properties*" In the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2019).

Paper X: Saad Shafiq, Irum Inayat, Muhammad Abbas. "*Communication Patterns of Kanban Teams and their Impact on Iteration Performance and Quality*" In the Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2019).

Paper Y: Muhammad Abbas, Abdul Rauf, Mehrdad Saadatmand, Eduard Paul Enoiu, Daniel Sundmark. "*Keywords-based test categorization for Extra-Functional Properties*" In the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2020).

Paper Z: Muhammad Abbas. "*Variability Aware Requirements Reuse Analysis*" In the Doctoral Symposium of ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings (ICSE 2020).

Contents

I

Thesis

Chapter 1

Introduction

Software-intensive products are often seen in variants. The variants are developed to target different market segments within the same industry. For example, the Tesla Model S comes in two variants, targeting the performance and long-range electric vehicle market segments. In many cases, the product variants should also comply with regional standards and regulations. In addition, products working with hardware should be able to tackle a variety of hardware configurations. Besides, these products are expected to be delivered quickly with high quality. Meeting these quick delivery and customization requirements necessitates an effective engineering process. Software Product Line (SPL/PL) are said to help achieve the quick delivery of quality products by providing an effective way to derive/develop an individual product by reusing common features across the products [1]. SPL Engineering (SPLE) typically consists of two main phases, known as Domain Engineering and Application Engineering. In domain engineering, a set of common features are realized via domain assets, satisfy a set of common requirements in a particular domain. Variations in the assets are introduced to handle varying customization requirements of the same product. In application engineering, the focus is mainly on deriving a product out of the SPL to satisfy particular customer requirements. Among other reported benefits of SPLE, the most commonly perceived benefits include reduced time to market, confidence boost, and increased product quality, achieved via a high degree of asset reuse [2].

However, SPLE adoption is inherently a complex and expensive task. Lit-

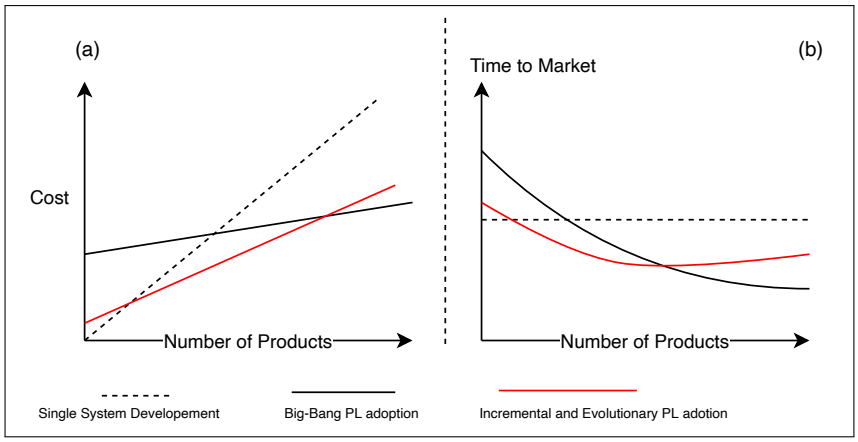

Figure 1.1: Cost comparison of PL adoption strategies [3]

erature suggests that the SPLE adoption can be done in two main ways, mostly dependent on the initial investment the company is willing to pay [3]. One way of adoption is the heavyweight approach (also known as the big-bang approach), where the process and practices are changed drastically. Such an approach to the SPLE adoption has a high upfront investment but reduces the time-to-market significantly [3] (shown as the black line in Figure 1.1). Companies are often not willing to pay substantial upfront investments if they see a low short-term return on investment [4]. In contrast, a prevalent industrial practice to the SPLE adoption is through incremental development of domain assets (e.g., railway [5] and automotive [6] industries). This adoption strategy allows an incremental change in the practices and might require less upfront investment (shown as a red line in Figure 1.1). Besides, there is evidence that most companies do not invest in a systematic reuse process but instead, go for a *clone-and-own* manner of reuse [5, 7].

In clone-and-own based evolutionary SPL, functionality is added to the SPL assets when needed. This approach results in many functional variants of the product line assets. Thus this way of SPLE adoption comes with some maintenance and co-evolution challenges as a by-product [5]. Figure 1.1 compares the two SPLE adoption strategies with single system development in terms of cost and time-to-market. Note that Figure 1.1 is a modified form

of the figure presented by Tüzün *et al.* [3].

Also, requirements prioritization becomes a significant activity in the context of SPL realized by clone-and-own reuse. In the case of SPLs, the product requirements are usually inter-dependent with varying development costs. In some cases, a significant amount of software can be reused from the SPL and thus reducing the cost of development. Very few requirement prioritization approaches consider inter-dependent product requirements with varying associated risk and cost.

Problem. This thesis is motivated by practical problems companies face in situations where reusable assets[1] realize the SPL, reused in a clone-and-own manner. In such a context, when a new product has to be derived from the product line, a reuse analysis of the SPL assets has to be conducted to ensure a high degree of asset reuse. In product derivation, the development team only has access to the agreed-upon requirements. Some key engineers read the requirements and recall if they have done something similar in other products [7]. If so, the engineers recommend SPL assets or their functional variants (usually from existing projects) for reuse. The recommended assets usually need modifications and thus are prioritized for implementation. The reuse analysis also helps in avoiding redundant development efforts in the early stages of product derivation. However, this process depends on the experience of the engineers and is time-consuming. Manual reuse analysis also becomes impractical when the number of existing derived products grows.

Summary of the Contributions. This thesis is a collection of four papers, realizing three contributions (i.e., C_1, C_2, and C_3). In the first contribution, we report the state-of-practice, challenges, and research opportunities in the SPLE process with *clone-and-own* reuse. In the second contribution, we focus on one of the identified challenges in the first contribution. In particular, we focus on aiding automated reuse analysis of SPL assets using requirements similarity as a proxy for software similarity. In addition, in the third contribution, we provide means for requirements prioritization in the presence of dependencies arising from reuse.

Thesis Outline. This thesis is divided into two parts. Part I gives an overview of the thesis and is organized as follows. Chapter 2 gives an overview of the research process followed, and Chapter 3 discusses the background and related work to this thesis. In Chapter 4, we provide an overview of the in-

[1] In our case, an asset is a Simulink model implementing a functionality.

cluded papers and the contributions. In Chapter 5, we conclude the thesis with a discussion on the planned work for the doctoral dissertation. Part II includes the collection of included papers, reformatted to comply with the thesis layout.

Chapter 2

Research Overview

In this chapter, we present the overall goals of the thesis, the research process, and the research methods used to realize the research goals.

2.1 Context & Research Goals

SPLs are based on the reuse rationale. The idea is to reuse implemented common features across variants to aid the quick delivery of complex products at scale. Typically, the product line is documented in feature models, and a product configurator is used to derive a product from the product line. However, in the safety-critical domain, compliance with safety standards requires companies to demonstrate the traceability between natural language requirements and their implementation. Thus a common practice in the safety-critical product lines is to describe the product line using natural language requirements. In our case, we consider the safety-critical product lines realized by assets (developed evolutionary) where the clone-and-own practices of reuse are followed. The assets typically realizes one or more customer requirements within the domain.

In the studied setting, the product derivation and configuration could modify product line assets. As shown in Figure 2.1, the product line (shown in purple) assets realizes a set of common requirements. When new products are derived and new functionality has to be added, the assets are evolved within

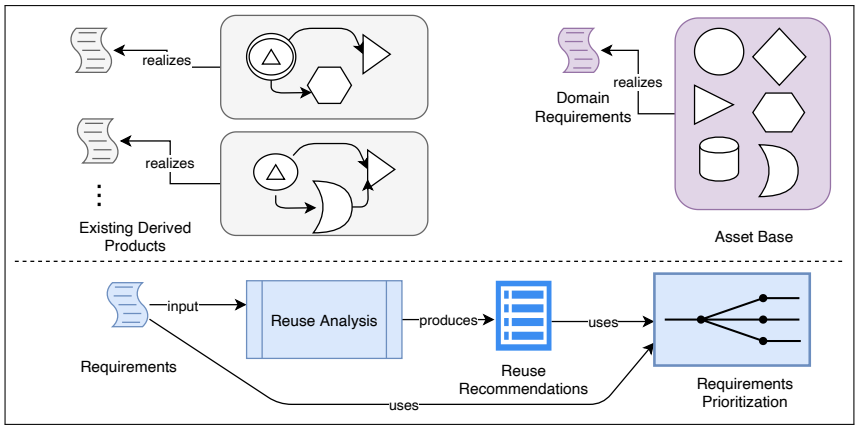

Figure 2.1: Reuse Analysis in clone-and-own reuse-based evolutionary product lines

the derived products. Figure 2.1 shows the evolution of the product line assets in the derived products (in gray color). As seen, the asset in circular shape is modified in both of the derived products to satisfy the product requirements.

In the considered SPLE context, companies are facing several challenges in their SPLE process. With many derived products and functional variants of product line assets, a company has to know if a new requirement(s) could already be satisfied by an existing asset(s). Same challenges are also reported in the literature [7]. To avoid redundant development efforts and ensure a high degree of reuse, a reuse analysis and prioritization process for product development is often introduced. Figure 2.1 outlines the reuse analysis activities, which uses existing derived products, product line, and new requirements as input. The reuse analysis phase's output is a list of recommended assets that could be reused to realize the new requirements. The requirements are then also prioritized for implementation. This process of reuse analysis in the SPLE context is heavily dependent on the experience of some key engineers and is time-consuming. Therefore, we propose to enhance the existing process of SPLE and formulated our first research goal as follows:

RG$_1$*: To identify challenges and opportunities in the current state-of-practice of a safety-critical SPLE process where reuse is done in a clone-*

and-own manner.

RG_1 also aims at collecting data about the current SPLE practices. In addition, it focuses on identifying concrete challenges and opportunities for research in the area.

In the journey towards achieving RG_1, we first selected a representative industrial case, following similar SPLE practices with requirements at the center of the development process and with a clone-and-own reuse process. In particular, we selected the Power Propulsion Control (PPC) division of the Bombardier Transportation AB (BT) as a representative of the considered context. BT is one of the leading railway vehicle manufacturing companies in the world. The PPC team is responsible for developing the PPC software for BT's railway vehicles for different customers. The team follows similar SPLE practices, as discussed above. We started to study the RG1 through document analysis [8], participant observation [9], and focus groups [10]. As a result, we identified several challenges in the current practices, requiring further investigation. In particular, we found that identifying reuse opportunities for SPL assets (reuse analysis) at the requirements level is a laborious activity, it depends on some key engineers' experience and is prone to human error (also realized in the literature [7]). As discussed, the reuse analysis typically uses new product requirements as input and looks for reuse opportunities for already implemented software assets. The idea is to find reusable assets that could be reused as-is or with fewer modifications to realize the new product's requirements. This process could also result in the addition of a new reusable asset(s) to the product line. While in some case, candidate assets could directly realize new customer requirements. Therefore, requirements prioritization also becomes a key activity in this context. This leads to the definition of our second research goal as follows:

***RG_2**: To support and automate the resource-intensive reuse analysis process in industrial SPLE settings.*

We mainly focus on the product lines described using natural language requirements. In addition, we assume that the requirements could be traced to the assets implementing them.

Literature suggests that the reuse analysis process follows a series of steps as follows [11]).

1. Identify high-level features that can help implement the requirements
2. Search existing assets in the asset base and in existing projects, to locate the different implementations of the feature
3. Analyze and select from the shortlisted assets
4. Plan and adapt the assets to the new product requirements

The RG_2 focuses on automating reuse analysis and requirements prioritization in the presence of dependencies (shown in blue color in Figure 2.1). Specifically, RG_2 is focused on supporting the first three steps in reuse analysis, and it partially supports the fourth activity in planning (i.e., requirements prioritization).

2.2 Research Process

Software engineering research often lacks practical relevance in the industry [12, 13]. If the research is conducted in an industrial setup, it mostly lacks engineers' views on the results. As a solution, the research community suggests the co-production process, where industry-academia collaboration is a key to achieve practical relevance [13]. Our research was performed in close collaboration with an industrial partner under the eXcellence In Variant Testing (XIVT) project [14]. In our case, the focus was to address a research problem of practical relevance. With this in mind, we conducted most of our studies in an industrial context. In addition, we focus on both qualitative and quantitative aspects. With qualitative assessment, we aim to realize our RG_1 and provide an overview of what engineers think of the results (obtained in RG_2) and how they can be improved. To gather qualitative data, we use empirical methods which requires less time from the participants, such as document analysis and focus group research. Specifically, we use a mixed-method research approach by using a combination of empirical studies (focus groups and case studies) with constructive research [15] to realize our research goals. We mainly followed a modified version of the standard collaborative research model proposed by Gorschek *et al.* [16]. The modifications were made to highlight our included papers. The model is shown in Figure 2.2. We summarized each step of our collaborative research process below.

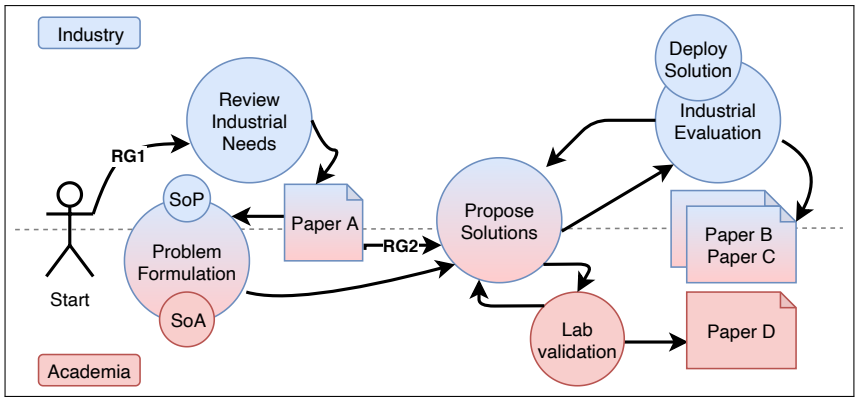

Figure 2.2: Research process followed in this thesis for technology transfer to industry

Review of Industrial Needs. We started with RG_1 in order to review the current practice of SPLE and identify challenges and opportunities in it. In Paper A, we started with the state-of-the-art of SPLE adoption. We supplemented document analysis with around twelve months of participant observation to report the state-of-practice in the team developing safety-critical software systems using SPLE. In addition, we conducted a focus group session with key engineers to identify challenges and opportunities in the studied context. We recorded the focus group and performed a thematic analysis to obtain results relevant to RG_1.

Problem formulation. The results of Paper A inspired the problem formulation. One of the identified challenges in Paper A was considered for further investigation. The problem was formulated under the supervision of the researchers and the involved industrial partner and was refined over several iterations. An early version of the problem specific to this thesis also resulted in a doctoral symposium publication [17]. This motivated the formulation of RG_2.

Propose Solutions & Evaluation. Solutions were proposed, evaluated, and prototypes were developed. In Paper B, we hypothesized that semantic similarity among requirements could be used to identify reuse opportunities for product line assets. We proposed a solution based on requirements similarity

and clustering to aid the reuse analysis process in the studied context. The proposed solution is evaluated in an industrial SPLE context.

In Paper C, we gathered empirical evidence for the hypothesis of Paper B ("semantic similarity among requirement can be used to identify reuse opportunities"). We explored the relationship between requirements similarity and software similarity in our particular case, studying the extent to which the similarity of the requirements can be used as a proxy for software similarity.

In Paper D, we proposed an approach to prioritize requirements based on dependencies. We proposed a domain-specific modeling language (DSML) for modeling requirements and their dependencies. We provided a tooled-solution to generate instance models of the proposed DSML from spreadsheets containing requirements and their dependencies. The proposal then uses the PageRank algorithm to rank the requirements based on dependencies, associated risk, and development cost. The approach is evaluated using the experiment research method in an academic setup.

Deploy Solution(s). The thesis resulted in two solutions called VARA (Variability-Aware requirements Reuse Analysis) and MBRP (Model-Based Requirements Prioritization). VARA is deployed at the PPC division of BT in Sweden. According to the company's internal evaluation, VARA can already reduce the time to market of the PPC software system by at least 20 days [1]. MBRP is available as an open-source tool [2].

[1] "VARA in News", available online, https://itea3.org/news/promising-results-using-nlp-and-machine-learning-to-automate-variability-and-reuse-analysis-at-bombardier-transportation.html

[2] "MBRP", available online, https://github.com/a66as/mbrp

Chapter 3

Background & Related Work

This chapter discusses the background and related work on the included papers in this thesis and is structured as follows. This chapter first presents background on the product line engineering adoption and the clone-and-own reuse practices. This thesis uses natural language processing approaches to aid the reuse analysis at the requirements-level. Therefore, the chapter also presents requirements similarity computation approaches and their role in reuse recommendation and other related software engineering tasks. Finally, the chapter ends with a short discussion on the related work.

3.1 Software Product Line Engineering and its Adoption

Software Product Line Engineering (SPLE) refers to the engineering of similar software products from a common assets base using a common means of production. The SPLs are based on the rationale of predictive reuse, unlike opportunistic reuse. The common assets in an SPL are developed if they will be reused in one or more software products. The assets satisfy a set of common requirements within a particular market segment. One or more assets could be combined to realize a high-level system feature.

Features are user-visible, high-level abstractions of the software capabilities. The Feature-Oriented Domain Analysis (FODA) method introduced the

concept of feature modeling [18]. Ideally, feature modeling is used to model the common set of features, variations, and dependencies within an SPL. In a feature model-based SPL, new products are derived from the SPL by selecting a set of features from the feature model in a specific configuration. The selection of features and feature configurations must satisfy a set of pre-defined constraints and dependencies requirements. The feature models could aid the automation of various resource-intensive activities such as product configuration, verification, and combinatorial interaction testing.

However, most companies are not often willing to maintain yet another model. Therefore, low-cost approaches are used to product line abstractions. In some cases, the SPLs are documented in natural language in the form of domain requirements or feature descriptions. Furthermore, companies try to reduce the SPLE cost by using an evolutionary development approach aided by clone-and-own reuse. This way of SPLE allows engineers with *Free Selection* [19]. Free selection allows engineers to freely browse and clone artifacts from asset-base to satisfy new product requirements. This can lead to the violation of dependency requirements, such as mutual exclusion.

Generally, clone-and-own reuse and free selection are not recommended in SPLE. However, it requires significantly less coordination, it is a low-cost approach, and is quite quick [7]. Furthermore, it allows companies to reduce the cost of SPLE adoption. In addition, requirements similarity analysis could be used to aid engineers in the free selection in this context. Note that in this thesis, we refer to the free selection process as reuse analysis.

3.2 Requirements Similarity

Requirements are typically written as natural language text. Therefore, most of the Natural Language Processing (NLP) and Information Retrieval (IR) algorithms that work with textual data are used for different requirements engineering tasks. Estimating the degree of similarity between requirements can be done on different levels as follows.

Lexical Similarity is a word-level textual similarity measure of the closeness of surface between requirements. Therefore, requirements with more overlapping terms would result in a high lexical similarity value.

Semantic Similarity is a phrase-level textual similarity measure of the closeness of meaning between the requirements. The semantic similarity approaches

Table 3.1: Example Requirements for demonstration

ID	Text
R1	When a new waypoint is created, it shall be added to the end of the flight route.
R2	The user shall reorder waypoints using mouse drag actions within a window listing waypoints for the route.

focus on the chain of words, unlike the lexical similarity to capture the semantics. Requirements might share high lexical similarity with very different meaning and therefore, semantic similarity approaches focuses on the meaning rather than the surface. For demonstration of different approaches, we will use example requirements taken from the design specification of the Dronology project [1]. The example requirements are shown in Table 3.1.

Requirements similarity computation approaches exist both on the string-level and on the vector-level. For example, the Jaccard similarity index is computed by dividing the intersection of words in the requirement pair on their union. For the example requirements, the Jaccard index would be 0.15, and the intersecting set is a set with $\{$*a, route, shall, the*$\}$ as members. Besides, Levenshtein distance is another string-level metric computed based on the edit distance between two strings of the requirements. On the other hand, vector-based requirements similarity approaches mostly use the cosine angle between the extracted vectors from the requirements as a measure of the degree of similarity. These approaches are focused on learning the representation of the documents in many dimensions represented in the form of vectors. The vectors can be derived from the requirements using word embedding and term-document-based information retrieval (IR) approaches. However, most of the document representation approaches require clean and pre-processed input text.

3.2.1 Pre-Processing for representation and similarity

Pre-processing textual data can vary between different NLP tasks. However, in this section, we outline and discuss the commonly used pre-processing pipeline for requirements engineering tasks. As shown in Figure 3.1, the example requirement *R1* is given as an input to the pipeline, and the requirement text

[1] Available online, https://dronology.info/datasets/

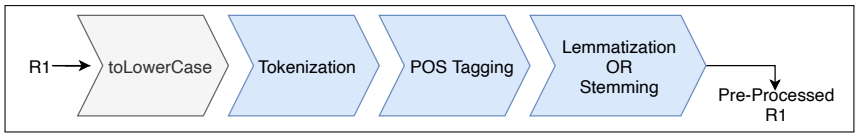

Figure 3.1: Pre-Processing pipeline

is tokenized, tagged, and finally, lemmatization is applied. In some cases, language-specific stop words (such as shall, the, and a) might be removed from the text of the requirements. We will explain each step in details using *R1* as an example input from the Table 3.1.

Lower Case. The requirements text might be converted in to lower case. However, some word embedding algorithms make use of the case information within the requirements' text. Therefore, this step might be skipped.

Tokenization is the process of demarcating the tokens of the text in the requirements. Simply put, the requirement text is split into sentences, and then sentences are split into words. The tokenization helps in guiding the part-of-speech (POS) tagging. Also, tokenization guides n-grams, where n number of tokens are considered, which carry more semantic and contextual meaning. The *R1* only has one sentence and 17 tokens in total.

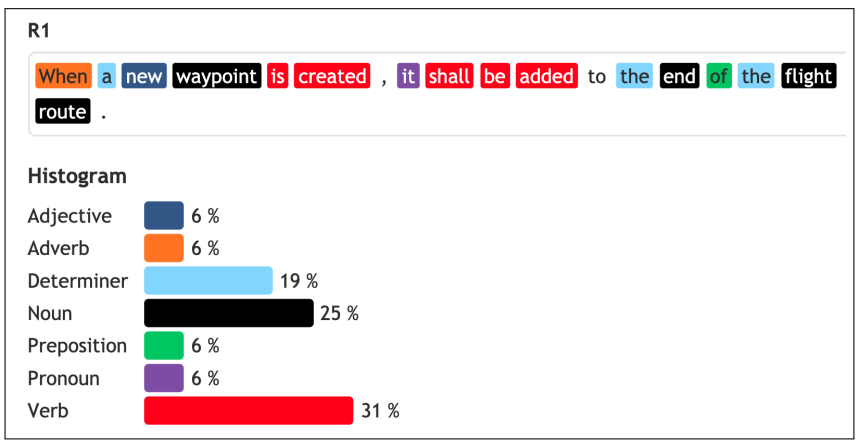

Figure 3.2: Tagged text of R1 with Part-Of-Speech

POS Tagging tags the tokens of text with their part-of-speech tags based on their context. The tagged token of *R1* is shown in Figure 3.2. The POS-Tagging guides the lemmatization and further classification of the tokens. For example, the nouns can be further classified into stakeholders, sub-systems, or parameter names. In addition, Named Entity Recognition (NER) is also applied in some cases to locate and classify named entities (such as names, codes, and expressions).

Stemming and Lemmatization. The future NLP tasks can interpret the same word in various forms differently. Therefore, the tokens of the requirements are rooted to their lemmas. Lemmas are computed using either stemming or lemmatization algorithms. The Porter stemming algorithm [20] is widely used in NLP. However, lemmatization is preferred since it produces lemmas that syntactically belongs to the same language with better precision [21]. For example, the WordNet lemmatizer uses a database search to lemmatize the tokens to their lemmas that exist within the language. The stemmed and lemmatized version of *R1* is as follows:

Stemmed: When a new waypoint is creat , it shall be ad to the end of the flight rout .
Lemmatized: Same as the text of R1

After the above pre-processing steps, other text-related tasks, such as word embedding, classification, and clustering are performed.

3.2.2 Word Embeddings

Word embedding algorithms are used to derive meaningful feature vectors out of textual data. Generally, a word embedding model maps the words of text into many different dimensions. This helps in providing more information (such as context) to the machine than the word itself. The idea is that semantically similar words in text should end up close to each other on the euclidean space. Requirement vectors can be extracted by taking an average of all the word vectors in the requirement. Upon query, similar requirements can be retrieved by inferring vectors for the query requirement. Note that the use of word embedding is not just limited to requirements engineering. In 2019, Google introduced the Bidirectional Encoder Representations from Transformers (BERT) embedding model to their searches [22]. BERT improved the web search and results

ranking significantly. For example, for the search query "math practice books for adults" Google search before BERT included a book in the "Young Adult" category. After BERT, Google can better understand the context and retrieve more helpful entries [2].

However, the requirements engineering tasks are not that complex compared to general language understanding and processing tasks. Therefore, simple term frequency-based approaches for vector generation may yield accurate results for retrieval. This is because the choice of terms when writing requirements is limited [23]. Therefore, in this subsection, we first present term frequency-based document representation approaches followed by more advanced neural network-based approaches for embeddings.

Term Frequency-Based Approaches. Term frequencies can be used to represent the presence/absence and the frequencies of terms in the documents. The terms are treated as the features of the vectors, and their frequencies are treated as values of the features. The bag-of-words approach, for example, is one popular frequency-based approach to term-document matrix generation and vectorization [24]. A term-document matrix is basically the representation of all the vectors of frequencies for all the documents and their terms. For the running example, the term-document matrix, generated using the Count Vectorizer, is shown in Figure 3.3. The rows of the matrix represent documents/requirements and can be treated as a vector.

Term-frequency Inverse Document Frequency (TFIDF) approach adds more information to the term-document matrix. As the name suggests, the term frequencies are divided by the document's total number of terms. This helps in considering the importance of the terms in the requirements. Note that the vectors generated by the frequency-based approaches might be very sparse (containing a lot of zeros). Therefore, the vectors are scaled to get more dense vectors.

Req.	actions	added	created	drag	end	flight	listing	mouse	new	reorder	route	user	using	waypoint	waypoints	window
R1	0	1	1	0	1	1	0	0	1	0	1	0	0	1	0	0
R2	1	0	0	1	0	0	1	1	0	1	1	1	1	0	2	1

Figure 3.3: Term-Document Matrix for the Example Requirements

[2] Available online, https://blog.google/products/search/search-language-understanding-bert/

Furthermore, the frequency vectors can be enriched with information by considering the co-occurrences of terms (n-grams). In n-grams, n number of terms are combined and treated as features. For example, "new" and "waypoint" from *R1* can be combined to add more semantically rich features to the term-document matrix. However, this could result in a very large term-matrix with a lot of dimensions. Therefore, dimensionality reduction approaches are used to reduce the vectors to a predefined number of dimensions without losing meaningful information.

The term-matrix based approaches can be combined with other statistical methods for IR. For example, Latent Semantic Indexing (LSI) is used to identify patterns in the relations of the terms and high-level concepts. LSI is being utilized in many software engineering tasks such as feature location and traceablity [25, 26]. Frequency vectors are also used by other topic modeling approaches, such as Latent Dirichlet Allocation (LDA) for grouping similar documents. LDA is also heavily utilized in requirements traceability, test selection, and even malicious mobile application detection [27, 28].

Neural Network-Based Approaches. The vectors can be more enriched by considering the contextual meaning of the words inside documents. This can be done using many available neural network architectures for representation learning. In this part of the subsection, we will only focus on the most commonly used neural network-based word embedding models, i.e., approaches based on skip-gram architecture and BERT.

The neural network-based word embedding approaches are based on the hypothesis of "words in similar context have similar meaning". In other words, the surrounding words before and after a word are focused. One popular example of these models is the architectures based on the word2vec approach [29]. A word2vec model tries to learn the context pairing, given the word as input. Word2vec uses a pre-defined window on the corpus in which it tries to learn the context of the seminal words. For example, ("Waypoint", "create") can be used as a context pair for learning as a positive example. In addition, negative pairs for training are also generated by randomly pairing unrelated words, such as ("Waypoint", "end"). The values on the model's output layer can be used as vectors representing a word in many dimensions.

Doc2Vec uses word2vec to learn vector representation of documents [30]. The Doc2Vec model also considers the uniqueness of each document by incorporating identifiers as features. As shown in Figure 3.4, a Doc2Vec model uses

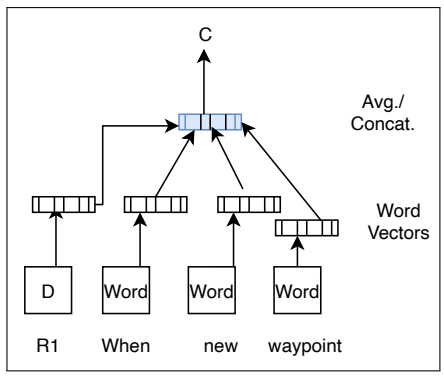

Figure 3.4: The architecture of Doc2Vec

averaging/concatenation of word vectors to generate a representation for the entire document. FastText is another similar widely used sentence representation learning approach developed by Facebook [31]. The FastText approach enriches the learning of representation with character-level (sub-word) information.

BERT is a recent breakthrough by Google in language understanding and representation learning. BERT uses a transformer-based deep neural network architecture to learn the contextual relations of the words and sub-words. The architecture uses the encoder-decoder mechanism where the encoder reads the input text, and the decoder produces the predictions. However, for language modeling tasks, only encoders are utilized. The BERT model takes the input sequence and breaks it down to token embeddings, sentence embeddings, and positional embeddings. The sentence embeddings could be used to derive meaning-rich representation vectors for requirements.

3.3 Related Similarity-Driven Tasks

Textual similarity metrics are exploited both on the vector representation-level and on the text-level in different software engineering tasks. Typically, the vectors together with similarity metrics are used in clustering and classification to aid different software engineering tasks. In this section, we present the related work to this thesis where NLP and language models are exploited.

3.3.1 Relevant Recommenders at the Requirements-Level

Recommender systems are basically information filtering systems based on relevance to the query or user actions. Typically, the content-based recommenders exploit similarity metrics and clustering to recommend existing artifacts from the repository. For example, most requirements recommenders exploit requirements similarity to recommend stakeholders, dependencies, traceability links, or reuse [32]. The underlying hypothesis for reuse is that associated artifacts (such as implementation) to similar requirements are similar and can be reused. This subsection will focus only on the related applications of these approaches and recommender systems in requirements engineering. First, we will present work related to requirements reuse and reuse recommendations, and then we will present the recommendation and prediction of requirements dependencies. We refer interested readers to a more comprehensive overview of the recommender systems in requirements engineering [32].

Identifying similar requirements for reuse can substantially reduce the development time and efforts. In literature, most approaches in this area mainly aid the reuse of requirements by deriving or structuring domain requirements. Models have been used to structure requirements for reuse and for configuration at the requirements level [33]. For example, Zen-ReqConf is a tooled approach to extract requirements hierarchies based on two different similarity measures. Moon *et al.* [34] propose a systematic method (called DREAM) for deriving generic domain requirements as core reusable assets. The method is based on analyzing the existing legacy requirements for commonalities and variabilities in base (primitive) requirements concerning system built. Literature also reveals the effective combination of IR techniques, NLP techniques, and Fillmore's theory for case analysis to extract requirements profile and Orthogonal Variability Model (OVM) [35]. The approach focuses on extracting the functional requirements profile using lexical affinity. A framework [36] for managing requirements was proposed. The framework uses a defined taxonomy to aid the reuse of requirements in an SPL. In addition, the literature reveals the use of a multi-ontology based approach for requirements reuse [37]. The approach is based on four ontologies, and the domain expert is tasked to construct domain and task-specific ontologies. Computer-aided tools and questionnaires are used to guide the analysts and end-users in deriving/eliciting application-specific requirements. Furthermore, Catalog-based approaches

for functional requirements reuse [38, 11, 39] can also aid reuse of functional requirements in new similar projects. Such approaches can be useful for small companies with no SPL in place.

Dependencies extraction. Product requirements can be inter-dependent, therefore extracting the relationships between different requirements becomes crucial. Example dependencies for requirements could be mutual exclusion, a requirement requiring another requirement, and refinements [40]. The requirement dependency information can be used for requirements prioritization, test case prioritization, and change impact analysis. In this part of the subsection, we summarize some efforts towards automated dependencies extraction from textual requirements. Atas *et al.* [41] propose an approach based on classification for extracting *requires* type of dependencies between requirements. The work compares different classifiers in the context of dependencies extraction from a labeled data-set. Their results show that the Random Forest classifier outperformed other classifiers in terms of F_1 score for dependencies extraction. Samer *et al.* [42] propose two variants of their approach for dependencies extraction of type *requires*. The first variant of the approach uses the TFIDF feature with probabilistic features for learning and applies classifiers for dependencies extraction. The second variant uses LSA for dependencies extraction. Their results also show that the Random Forest-based classifier performs better than all other classifiers and the LSA-based recommender. Deshpande *et al.* [43] combine ontologies with active learning for dependencies extraction. The approaches classifies the dependencies into *requires*, *refines* and *others*.

3.3.2 Traceability

One popular application of similarity in software engineering is the trace link recovery between requirements and other artifacts. These approaches typically use term tracing and similarity to establish traceability links between requirements and other artifacts. In this part of the subsection, we summarize some of the seminal approaches for traceability using IR and NLP approaches. Lucassen *et al.* propose an approach to extract traceability matrix to link user stories with source code given the textual test specifications [44]. IR-based trace link recovery approaches can be found in the literature [45, 46, 47, 48]. These approaches mostly make use of the Vector Space Model (VSM) and TFIDF for the similarity-based tracing of requirements. Probabilistic inference models

are also used for trace recovery [49]. The tool computes a probability value between the source query and the target artifact for linking. The calculations are based on term frequencies of textual content of the source and target artifacts. Latent Semantic Indexing (LSI [50]) for traceability link recovery is also one of the popular choices of algorithms [51]. These approaches extract terms from the source and target artifacts and then use LSI to suggest traceability links.

Traceability link recovery problem is also often formulated as a search problem [52, 53]. Ghannem *et al.* use TFIDF-based semantic similarity, recency, and frequency of change to recover traceability links using a non-dominant genetic algorithm (NSGA-II). Hariri *et al.* propose a conceptual framework for traceability link recovery for self-adaptive systems at runtime [53]. The conceptual framework is based on the use of NLP and IR techniques, together with evolutionary algorithms.

Another conceptual model [54] is also proposed to handle the traceability of variability. The conceptual model aims to provide a one-to-one mapping of variability from problem to solution space. The model is demonstrated using a small case study. An effort has been made for formalizing the traceability relations between PL artifacts [55]. The study presents a formalization of different types of relationships (such as satisfiability, similarity, variability, etc.). The study also presents tool support for traceability link recovery in the context of a PL. The tool relies on XQuery rules for trace recovery. Other rule-based trace link recovery approaches also exist in the literature [56].

3.3.3 Feature Model Extraction

Some feature model extraction approaches also use NLP and IR-based approaches, together with clustering and similarity metrics. These approaches typically *reverse* engineer variability from existing product variants. However, some feature model extraction approaches do support the domain engineering phase in *forward* engineering manner. For example, public documents such as brochures and online reviews can be used for extracting domain terms and their variabilities [57, 58]. Bottom-up technologies for reuse (But4Reuse) uses existing variants to identify the feature and extract variability [26]. But4Reuse also allows the construction of reusable assets for systematic reuse. Feature and Feature Relationship Extraction (FFRE) [59] is a tooled approach for recommending features and their relationships. Like other feature model extraction

approaches, the FFRE focuses on aggregating the natural language requirements to extract a high-level system feature model. ArborCraft [60, 61] uses LSA to calculate the similarity between requirements pairs and clusters the requirements based on similarity (shared concepts in the requirements) to extract feature trees. SOVA [62] uses semantic role labeling to calculate similarity-based semantic roles. The roles extracted in SOVA reflect behavioral information and base the similarity metrics on behavior. SOVA applies a hierarchical clustering algorithm to cluster the requirements based on behavioral similarity.

3.3.4 Feature Location

Feature location approaches locate SPL features in the source code. Typically, these approaches assume that the feature description and source code of the variants share common terms. Identifying the features' initial location might help in maintenance activities, such as the feature to code traceability, change impact analysis, and re-engineering of legacy variants. As shown in Figure 3.5, applying feature location approaches identifies initial location of pre-defined features in the variants of the product.

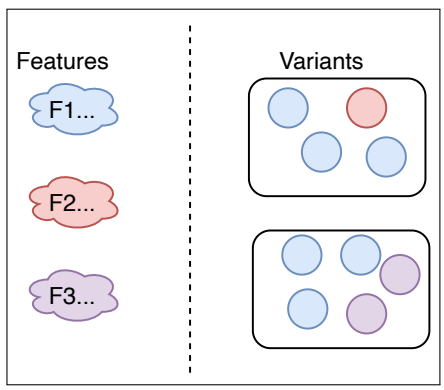

Figure 3.5: The result of Feature Location approaches

Over the years, many approaches for feature location have been proposed [63]. The feature location approaches can be classified into three main classes as follows.

Run-Time feature location approaches use run-time information for feature location. These approaches are also called dynamic feature location approaches.
Static feature location approaches use static analysis on the product variants to locate feature in product variants.
Text-based feature location approaches use NLP and textual analysis to locate features in product variants.

Note that many feature location approaches combine different classes of methods (such as static and text-based) for feature location. Here, we provide an overview of some of the approaches for feature location.

But4Reuse use LSI to locate features in different product variants. In addition, But4Reuse provides several real-world benchmarks for evaluating new feature location approaches [64]. Zhao *et al.* use branch reverse call graphs together with IR to locate features in the source code [65]. This is done by establishing an initial trace between the functions and feature description. The initial traces are filtered by pruning and analyzing the call graph. Poshyvanyk *et al.* combine execution scenario-based probabilistic ranking with LSI to locate features in the source code [66]. The approach uses ranked dynamic events from the execution traces with lexical similarity of the text of the source to locate the feature of interest. The approach is evaluated using two case studied, Eclipse and Mozilla. Andam *et al.* propose the tool called FLOrIDA for feature location [67]. FLOrIDA uses a TFIDF-based approach, Lucene, to compute the similarity between feature description and source files. The most similar source files are then ranked using PageRank algorithm. Moslehi *et al.* use topic modeling on textual data in the graphical interfaces and speech, related to products, to locate the features in source code [68]. Recent empirical evidence shows that the feature location approaches are sensitive to the system under analysis [69]. Results further indicate that the Lucene-based approach for feature location performed significantly better than other approaches.

3.4 Requirements Prioritization

Requirements prioritization is the process of ordering requirements based on their relative importance for implementation [70]. Typically, the prioritization is done based on multi-criteria, such as development cost, risk, and business value. The results of requirements prioritization approaches is a ranked list of requirements with the most valuable requirements on the top. Therefore, the

use of multi-criteria decision making and ranking approaches for requirements prioritization is a common practice. In this section, we first present some of the most seminal requirements prioritization approaches, followed by a discussion on the use of models for requirements prioritization.

Analytical Hierarchy Process (AHP) is one of the most widely known requirements prioritization approaches [71]. AHP models the process in multi-level hierarchies considering the goal, criteria, and options/alternatives. The approach then computes the relative importance of the considered criteria based on their contribution towards the goal. Then the options are evaluated based on each criterion. Finally, judgments are made based on pair-wise comparisons. In addition, a generalized form of the AHP method is also used for requirements prioritization called Analytical Network Process (ANP). ANP uses networks instead of hierarchies and thus helps in considering bidirectional relationships between the criteria. Furthermore, the pair-wise comparison for judgments becomes double in the case of ANP.

Cumulative Voting (CV) assigns fix number of prioritization scores to stakeholders that are participating in the prioritization process. All the stakeholders then assign some points from their prioritization score to different requirements. At the end of the process, a ranked list of requirements is obtained based on the score assigned to them by different stakeholders. The 100-points method is also a variant of the CV method, where stakeholders are given 100 points to be assigned to their favorite requirements. Furthermore, other similar numerical methods, such as Planning Game (PG) and Numerical Assignment Technique (NAT), use a fixed scale, between which stakeholder is supposed to assign a priority to requirements [72].

Binary Search Trees (BST) has also been used for prioritization of requirements [73]. In the BST method, a root requirement is selected, followed by a systematic comparison of each requirement with the root node. This results in a prioritized list of requirements. Furthermore, the requirements prioritization is also formulated as a search problem. A genetic algorithm (GA) is used together with manual input to minimize the disagreement between the total order of requirements [74]. The approach relies on user input in cases where requirements with high fitness cannot be distinguished.

Domain-Specific Modeling and Requirements Prioritization. Domain-Specific Modeling Languages (DSMLs) provides domain-specific constructs for modeling systems for analysis and development tasks. Typically, the lan-

guages are based on a meta-model describing the semantics of the language. The meta-model is instantiated by the end-users with concrete syntax to model the system in the DSML using its capabilities. The intended use of DSML can vary. For example, SysML is a domain-specific modeling language for modeling system's architectures [3]. On the other hand, the MARTE profile provides more concrete concepts relevant to time and is used to model real-time systems [4].

Many DSMLs have been proposed for modeling and visualizing the requirements, their inter-dependencies, and stakeholders. For example, the SystemWeaver Requirement Management tool [5] uses a DSML for requirements management. The tool allows modeling high-level features and requirements with a focus on traceability. Furthermore, goal modeling frameworks, such as i*, are used for understanding the problem domain of the systems [75]. The i* framework allows modeling of actors, goals, tasks, and resources and thus could be used for goal-driven requirements engineering. Therefore, the instance models of the requirements' DSMLs can also be used for requirements prioritization. For example, the DRank approach extract dependencies from i* models and apply PageRank algorithm to rank the requirements based on several criteria [76].

Reflection on the Related Work. Many related approaches for requirements-level reuse exists in the literature. The problem of reuse recommendation can also be formulated as a traceability or feature location problem. In addition, requirements prioritization is heavily studied in the literature. However, the context of the studies is very different from the studies included in this thesis. In this thesis, we first report the challenges and opportunities in our studied settings. We then target requirements-level reuse recommendation and prioritization for safety-critical SPLs (where requirements can be traced) with clone-and-own reuse.

[3] Available online, https://sysml.org/
[4] Available online, https://www.omg.org/omgmarte/
[5] Available online, https://www.systemweaver.se/

Chapter 4

Research Results

In this chapter, we discuss our results and present a summary of the contributions. We highlight the specific contributions of the included papers with a discussion on the validity of the results.

4.1 Thesis Contributions

This thesis focus on two main goals, realized by three contributions. Here we re-present the goals as follows:

> ***RG***$_1$*: To identify challenges and opportunities in the current state-of-practice of a safety-critical SPLE process where reuse is done in a clone-and-own manner.*
>
> ***RG***$_2$*: To support and automate the resource-intensive reuse analysis process in industrial SPLE settings.*

Our research goals are realized by three main contributions as follows.

- **C**$_1$: Identified challenges and opportunities in SPLE adoption (SPLE Challenges)
- **C**$_2$: An approach for automated reuse recommendation of product line assets based on natural language requirements similarity (VARA)

- **C_3** An approach for requirements prioritization based on requirements dependencies (MBRP)

Table 4.1 present a mapping of the contributions to our research goals.

Table 4.1: Mapping of contributions to the research goals

	RG_1	RG_2
C_1	X	
C_2		X
C_3		X

4.1.1 C1: SPLE Challenges

C_1 is realized by Paper A [5], which outlines the current practices of SPLE at a company, benefits of SPLE, perceived challenges, areas of improvements, and future vision of the company regarding their SPLE process. Using the clone-and-own and evolutionary SPLE process, the company was able to achieve significant improvements. In particular, the development and testing time was reduced significantly. In addition, a confidence boost was experienced in derived products.

Several challenges and improvement opportunities were identified. The challenges were divided into three classes as follows: Product Derivation, Automation, and SPLE Awareness. In product derivation, among several concrete challenges, the identification of reuse opportunities was one of them. We specifically focused on the identification of reuse opportunities and requirements prioritization in the process of reuse analysis. In the automation, it was observed that the configuration of general supporting tools for SPLE is a challenge. Besides, it was also observed that the lack of SPLE awareness leads to architectural decisions that negatively impact reuse.

The future vision of the company includes Test automation, continuous integration (CI), and the creation of a regression test suite. Specifically, the focus would be on variability-aware automated test generation for product variants in a CI environment.

Validity: The paper realizes this contribution using the focus group research combined with thematic analysis on the transcript. Several measures were taken to tackle the potential validity threats, such as following well-established research methods, selecting a diverse set of participants within the company, and including multiple researchers and practitioners in study design. Also, the focus group results were supplemented with document analysis and participant observation.

4.1.2 C2: VARA

Paper B [77] and Paper C [78] forms this thesis contribution. In the identified challenges in C_1, this contribution is focused on supporting the reuse analysis activities at the requirements-level. Therefore, we explored the use of requirements similarity for reuse recommendation. In Paper B, we proposed an approach for SPL asset reuse recommendation based on customer requirements. The approach uses word embedding algorithms and clustering to recommend the reuse of SPL assets for new customer requirements. Results show that the approach can recommend reuse with around 74% average accuracy. Besides, engineers think that the results produced by the approach are useful and insightful for reuse analysis.

The approach assumes that similar software assets realize similar requirements. However, this assumption is not validated in the literature. Therefore, in Paper C, we explore the relationship between requirements similarity and software similarity. Specifically, we computed the correlation between requirements similarity and software similarity. We found a moderate positive correlation between the two variables and concluded that there is a need for new methods to compute requirements similarity that reflects the software similarity better.

Validity: We evaluated the approach in industrial settings and presented engineer's perception on the obtained results. The typical assumption that "requirement similarity can be used to recommend reuse" was validated. To tackle potential validity threats, we used data obtained from two different projects at the company.

4.1.3 C3: MBRP

Paper D [79] realizes this thesis contribution and is motivated by C_2. When reuse is done in an evolutionary context, the risk and cost associated with the requirements may vary. This is because some requirements could be realized by making no/some adaptations to existing assets. While for some requirements, a new asset should be developed to implement them. For release planning in such scenarios, requirements prioritization has to be done.

In this contribution, we propose and evaluate an approach for dependencies based requirements prioritization. The approach uses a domain-specific modeling language for modeling the requirements and their inter-dependencies. We then use a modified version of the PageRank algorithm to transform the instance model into a prioritize requirements list. The prioritization is done based on associated risk, cost, business value, and dependencies of the requirements. Results show that our approach was able to produce closer results to human subjects than the state-of-the-art.

Validity: We evaluated the approach in academic settings and compared the results to five state-of-the-art requirements prioritization approaches. We conducted an experiment with 30 graduate students to obtain a baseline for comparison, which could be a potential threat to the validity of the results. However, scientific evidence [80] supports the use of students in software engineering experiments.

4.2 Paper Contributions

In this section, we present the abstracts and a short description of the contributions for the included papers. Each thesis contribution is mapped to at least one included, as shown in Table 4.2.

Table 4.2: Mapping of contributions to the included papers

	Paper A	Paper B	Paper C	Paper D
C_1	X			
C_2		X	X	
C_3				X

4.2.1 Individual Contributions

I have been the primary driving researcher and the author for all the included papers. However, the co-authors helped in writing few sections in some of the included papers. The supervision team participated in the brainstorming and planning sessions for the research and provided useful feedback on the drafts of the included papers.

4.2.2 Included Papers

Paper A: Product Line Adoption in Industry: An Experience Report from the Railway Domain
Authors: *Muhammad Abbas*, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, Daniel Sundmark
Abstract: The software system controlling a train is typically deployed on various hardware architectures and is required to process various signals across those deployments. Increases of such customization scenarios, as well as the needed adherence of the software to various safety standards in different application domains, has led to the adoption of product line engineering within the railway domain. This paper explores the current state-of-practice of software product line development within a team developing industrial embedded software for a train propulsion control system. Evidence is collected by means of a focus group session with several engineers and through inspection of archival data. We report several benefits and challenges experienced during product line adoption and deployment. Furthermore, we identify and discuss research opportunities, focusing in particular on the areas of product line evolution and test automation.
Paper Contributions: The results of Paper A confirms the benefits of SPLE in the literature. Besides, it adds to the body of knowledge by identifying concrete challenges that need investigation. Furthermore, it outlines the future vision and area of improvements regarding SPLE from the engineer's perspective.

Paper B: Automated Reuse Recommendation of Product Line Assets based on Natural Language Requirements
Authors: *Muhammad Abbas*, Mehrdad Saadatmand, Eduard Paul Enoiu, Daniel Sundmark, Claes Lindskog
Abstract: Software product lines (SPLs) are based on reuse rationale to aid

quick and quality delivery of complex products at scale. Deriving a new product from a product line requires reuse analysis to avoid redundancy and support a high degree of asset reuse. In this paper, we propose and evaluate automated support for recommending SPL assets that can be reused to realize new customer requirements. Using the existing customer requirements as input, the approach applies natural language processing and clustering to generate reuse recommendations for unseen customer requirements in new projects. The approach is evaluated both quantitatively and qualitatively in the railway industry. Results show that our approach can recommend reuse with 74% accuracy and 57.4% exact match. The evaluation further indicates that the recommendations are relevant to engineers and can support the product derivation and feasibility analysis phase of the projects. The results encourage further study on automated reuse analysis on other levels of abstractions.
Paper Contributions: Many existing approaches are mainly focused on requirements reuse. This paper proposes a similar reuse recommender but specifically focused on the reuse of SPL assets based on customer requirements. The approach supports product derivation in industrial settings and is evaluated both qualitatively and quantitatively.

Paper C: Is Requirements Similarity a Good Proxy for Software Similarity? An Empirical Investigation in Industry
Authors: *Muhammad Abbas*, Alessio Ferrari, Anas Shatnawi, Eduard Paul Enoiu, Mehrdad Saadatmand
Abstract: **[Context and Motivation]** Content-based recommender systems for requirements are typically built on the assumption that similar requirements can be used as proxies to retrieve similar software. When a new requirement is proposed by a stakeholder, natural language processing (NLP)-based similarity metrics can be exploited to retrieve existing requirements, and in turn identify previously developed code. **[Question/problem]** Several NLP approaches for similarity computation are available, and there is little empirical evidence on the adoption of an effective technique in recommender systems specifically oriented to requirements-based code reuse. **[Principal ideas/results]** This study compares different state-of-the-art NLP approaches and correlates the similarity among requirements with the similarity of their source code. The evaluation is conducted on real-world requirements from two industrial projects in the railway domain. Results show that requirements similarity computed with the

traditional *tf-idf* approach has the highest correlation with the actual software similarity in the considered context. Furthermore, results indicate a *moderate positive* correlation with Spearman's rank correlation coefficient of more than 0.5. **[Contribution]** Our work is among the first ones to explore the relationship between requirements similarity and software similarity. In addition, we also identify a suitable approach for computing requirements similarity that reflects software similarity well in an industrial context. This can be useful not only in recommender systems but also in other requirements engineering tasks in which similarity computation is relevant, such as tracing and categorization.
Paper Contributions: Content-Based requirements reuse recommenders typically assume that similar requirements are realized by similar software. However, the relationship between the requirements similarity and software similarity remains un-explored. The paper contributes to the body of knowledge by providing industrial empirical evidence on the relationship between requirements similarity and software similarity.

Paper D: MBRP: Model-based Requirements Prioritization Using PageRank Algorithm
Authors: *Muhammad Abbas*, Irum Inayat , Naila Jan , Mehrdad Saadatmand, Eduard Paul Enoiu, Daniel Sundmark
Abstract: Requirements prioritization plays an important role in driving project success during software development. Literature reveals that existing requirements prioritization approaches ignore vital factors such as interdependency between requirements. Existing requirements prioritization approaches are also generally time-consuming and involve substantial manual effort. Besides, these approaches show substantial limitations in terms of the number of requirements under consideration. There is some evidence suggesting that models could have a useful role in the analysis of requirements interdependency and their visualization, contributing towards the improvement of the overall requirements prioritization process. However, to date, just a handful of studies are focused on model-based strategies for requirements prioritization, considering only conflict-free functional requirements. This paper uses a meta-model-based approach to help the requirements analyst to model the requirements, stakeholders, and inter-dependencies between requirements. The model instance is then processed by our modified PageRank algorithm to prioritize the given requirements. An experiment was conducted, comparing

our modified PageRank algorithm's efficiency and accuracy with five existing requirements prioritization methods. Besides, we also compared our results with a baseline prioritized list of 104 requirements prepared by 28 graduate students. Our results show that our modified PageRank algorithm was able to prioritize the requirements more effectively and efficiently than the other prioritization methods.

Paper Contributions: Very few existing requirement prioritization approaches considers requirements dependencies. This paper contributes with an approach for requirements prioritization based on requirement dependencies, associated risk, development cost, and business value. The paper also evaluates the proposed approach in comparison with five state-of-the-art techniques for requirement prioritization.

Chapter 5

Conclusion, Discussion, & Future Work

5.1 Conclusion & Summary

Evolutionary adoption of SPLE with a clone-and-own reuse helps companies in saving time and resources. Companies may see a quick return on the investment in the SPLE. However, down the road, companies have to deal with many challenges in co-evolution and assets management. In safety-critical product lines, the natural language requirements are at the center of the development process. By supporting the product derivation at the requirements-level in such cases, we aim to allow companies to avoid redundant development efforts and ensure a high degree of asset reuse.

This thesis focuses on two research goals: identifying challenges in the SPLE process and supporting the reuse analysis, and prioritization process for the SPL assets at the requirements-level. The goals are realized by three contributions, packaged into four included papers.

In Paper A, we used empirical methods to report the current practices of SPLE at a company. We collected data about the benefits of SPLE adoption, perceived challenges and improvement opportunities, and future vision of the SLPE process at the company. Results show that with unsystematic reuse in the evolutionary SPLE process, the company could reduce time-to-market. Fur-

thermore, the adoption allowed incremental safety assessment and resulted in a confidence boost in the derived products. However, challenges in product derivation and maintenance were observed. Mainly, maintaining a high degree of asset reuse, the evolution of assets, and change impact analysis were seen as challenges.

In Paper B, and Paper C, we support the reuse analysis activities in product derivation. We hypothesized that similarity among requirements could be used to recommend the reuse of product line assets. Results show that we were able to recommend the reuse of product line assets with around 74% of average accuracy. Results further indicate that the reuse recommendations generated by our approach are useful to engineers and can support reuse analysis activities in the studied settings, Besides, we tested the typical assumption that similar requirements are realized by similar software. We applied correlation analysis to study the relationship between requirements similarity and software similarity. We found a moderate correlation between the two variables.

Evolutionary development of product line assets requires requirements prioritization for release planning. When parts of the requirements are already implemented, such cases reduces the risk and development cost associated with the requirements. Furthermore, the reuse creates a dependency between the new requirements and the reused ones. In such cases, requirements prioritization based on dependencies, development cost, and associate risk becomes an ideal solution.

Paper D provides support in modeling the requirement and their interdependencies. Furthermore, it uses a modified version of the PageRank algorithm to prioritize the requirement for implementation. Results from the evaluation show that dependencies-based prioritization produces closer results to human subjects compared to the state-of-the-art.

5.2 Discussion and Future Work

In the future, we plan to address some of the encountered challenges in the included papers. This section provides a brief overview of the possible future directions and extensions for the work presented in this thesis.

Extensions to the Included Papers: In Paper B, we presented an approach for reuse recommendation of SPL assets. We noticed that the results are sensitive to the existing requirements used for training by the approach. In some cases, existing requirements could result in high similarity with multiple new customer requirements. We found that some requirements could be broken down into multiple requirements. A future extension of the VARA approach could be to warn about requirements that could be broken down into multiple requirements. Furthermore, metrics on the quality of the requirements could be calculated. Requirements that are hard to read, ambiguous, and are not complete could be marked for review. This same analysis could also be applied at different levels of abstraction to support project bids and testing.

Writing new test cases is a resource-intensive task. Test cases written for existing products are often available in a shared repository. One of our future areas of research is to support testers in manual test design. We aim to use static analysis to identify similar software slices to the software under the test from the repository and recommend the reuse of test cases. Specifically, we aim to support the test data reuse and test oracle reuse.

The work presented in Paper C correlates the similarity of the requirements with software similarity in a particular case. A future extension could be extending existing requirement similarity computation methods to reflect the software similarity better. A large-scale empirical evaluation of multiple case studies could also yield interesting results. Furthermore, companies are often not willing to pay substantial investment in SPLE unless they see a good return on investment. The approaches used in Paper C could be packaged into a tool that derives a business case for product line adoption and systematic reuse. Metrics on incoming customer requirements and the implementing software can indicate that a huge portion of the products is similar and can be migrated to an SPL to save time and avoid redundant development efforts. Finally, automatically migrating clone variants is also an interesting area of research and is one of our planned future directions.

Paper D focuses on requirements prioritization in the presence of dependencies. A relevant parallel activity is the test case prioritization and selection in cases where the execution of the entire test suite is not possible. An extension to work presented in Paper D could be test case prioritization based on dependencies. PageRank algorithm could be used to rank the test cases based on their dependencies with other test cases and software components. Further-

more, the requirements similarity information, bug detection capability, test case diversity, and coverage could also be used for test selection.

Delta-oriented Test repair and Amplification (DoTA): In the studied settings, the SPL assets are modified to realize new customer requirements. The test cases for the original version of the assets are usually available. The modification to the assets results in test breakages. Future works include a *what-if* analysis to support the developer in the change-impact analysis. We aim to classify the test cases into two classes (repairable and not repairable) based on heuristics. This way, we would be able to quickly compute the impact of a given change on the linked test cases and let the developer know about the potential test breakages. In addition, an approach that would automatically repair the test cases for the modified assets is also planned for future research. Furthermore, we aim to use search-based algorithms to amplify the existing test suite for coverage of the deltas in the modified assets.

Towards a more systematic SPLE process: Due to the safety-critical nature of the studied case, requirements are at the center of the process. The product line is described using natural language in the form of domain requirements. Automated product derivation is not of high interest. This is due to the fact that the derived product might end up with a dead code. However, small increments towards a more systematic SPLE process are of very high interest to the company. Given the existing assets (Simulink models in the studies setting) and their natural language requirements, feature model synthesis approaches could be used to extract the feature model. In addition, decision models can also be extracted from the variants. This will open a lot of possible research directions for the company, such as combinatorial interaction testing and requirements to feature traceability.

Co-evolution of the product line and the derived products is seen as a challenge. In many cases, smaller patches are committed to the product line assets, and the decision of whether or not to propagate those changes to the derived products becomes challenging. Migrating the clone-and-own SPL to more systematic reuse would also be of high interest. Given the existing variants, an approach could be designed to automatically identify the identical clones in the variants and migrate them to a central library for future reuse. This way, the future modifications could be done in a central assets base, and the changes

could be easily propagated to the derived products. However, the propagation of changes to the derived products will lead to another challenge of re-testing the derived products. Variability-aware change impact analysis methods are needed to avoid re-testing of the entire product.

Metrics, Smells, and Process Consistency Checking: During product derivation cycles, many modifications are done to the standard product line assets. The quality characteristic may vary in each derived variant. Metrics relevant to the quality trends could be computed across product variants by looking into history. Integration points, clones, and bug rates could be computed across product variants and commits to visualize quality trends in the derived products. Furthermore, configuration smells relevant to the evolutionary development of SPLs could be detected in the variants. For example, features/assets that are modified in most variants could be flagged, and assets that do many things and are used by many other assets (god asset) could be detected.

One of our future directions also includes domain-specific modeling of the SPLE process. We aim to provide support to the architect to model the SPLE process and define feature-level rules that must be followed. We then aim to propose an approach to check the defined rules in the variants. For example, two mutually exclusive features cannot be part of the same product, and this is a rule violation and should be flagged.

Bibliography

[1] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[2] Frank Dordowsky and Walter Hipp. Adopting software product line principles to manage software variants in a complex avionics system. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, page 265–274, USA, 2009. Carnegie Mellon University.

[3] Eray Tüzün and Bedir Tekinerdogan. Analyzing impact of experience curve on roi in the software product line adoption process. *Inf. Softw. Technol.*, 59:136–148, 2015.

[4] Jan Bosch. On the development of software product-family components. In Robert L. Nord, editor, *Software Product Lines*, pages 146–164, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, and Daniel Sundmark. Product line adoption in industry: An experience report from the railway domain. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, SPLC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[6] Damir Bilic, Daniel Sundmark, Wasif Afzal, Peter Wallin, Adnan Causevic, Christoffer Amlinger, and Dani Barkah. Towards a model-driven product line engineering process – an industrial case study. In *13th Innovations in Software Engineering Conference*, March 2020.

[7] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of cloning in industrial software product lines. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 25–34. IEEE, 2013.

[8] Glenn A Bowen et al. Document analysis as a qualitative research method. *Qualitative research journal*, 9(2):27, 2009.

[9] Barbara B Kawulich. Participant observation as a data collection method. In *Forum qualitative sozialforschung/forum: Qualitative social research*, volume 6, 2005.

[10] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. The focus group method as an empirical tool in software engineering. In *Guide to advanced empirical software engineering*, pages 93–116. Springer, 2008.

[11] C. Pacheco, I. Garcia, J. A. Calvo-Manzano, and M. Arcilla. Reusing functional software requirements in small-sized software enterprises: a model oriented to the catalog of requirements. *Requirements Engineering*, 22(2):275–287, 2017.

[12] Vahid Garousi, Markus Borg, and Markku Oivo. Practical relevance of software engineering research: synthesizing the community's voice. *Empirical Software Engineering*, pages 1–68, 2020.

[13] V. Basili, L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh. Software engineering research and industry: A symbiotic relationship to foster impact. *IEEE Software*, 35(05):44–49, sep 2018.

[14] Holger Schlingloff, Peter M. Kruse, and Mehrdad Saadatmand. Excellence in variant testing. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, VAMOS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Gordana Dodig Crnkovic. *Constructive Research and Info-computational Knowledge Generation*, pages 359–380. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[16] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. A model for technology transfer in practice. *IEEE software*, 23(6):88–95, 2006.

[17] Muhammad Abbas. Variability aware requirements reuse analysis. In *The 42nd International Conference on Software Engineering Companion*. ACM, May 2020.

[18] 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, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[19] Mike Mannion and Javier Camara. Theorem proving for product line model verification. In *International Workshop on Software Product-Family Engineering*, pages 211–224. Springer, 2003.

[20] Martin F. Porter. An algorithm for suffix stripping. *Program*, 40:211–218, 1980.

[21] Vimala Balakrishnan and Ethel Lloyd-Yemoh. Stemming and lemmatization: a comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2014.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[23] A. Ferrari, G. O. Spagnolo, and S. Gnesi. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505, 2017.

[24] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[25] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[26] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 101–110, 2015.

[27] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *ICSE'14: Proceedings of the 36th International Conference on Software Engineering*, 2014.

[28] Stephen W Thomas, Hadi Hemmati, Ahmed E Hassan, and Dorothea Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212, 2014.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[30] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[31] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2017.

[32] Alexander Felfernig, Gerald Ninaus, Harald Grabner, Florian Reinfrank, Leopold Weninger, Dennis Pagano, and Walid Maalej. An overview of recommender systems in requirements engineering. In *Managing requirements knowledge*, pages 315–332. Springer, 2013.

[33] Yan Li, Tao Yue, Shaukat Ali, and Li Zhang. Enabling automated requirements reuse and configuration. *Software and Systems Modeling*, 18(3):2177–2211, 2019.

[34] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering*, 31(7):551–569, 2005.

[35] 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.

[36] Maximiliano Arias, Agustina Buccella, and Alejandra Cechich. A Framework for Managing Requirements of Software Product Lines. *Electronic Notes in Theoretical Computer Science*, 339:5–20, 2018.

[37] Zong Yong Li, Zhi Xue Wang, Ying Ying Yang, Yue Wu, and Ying Liu. Towards a multiple ontology framework for requirements elicitation and reuse. In *Proceedings - International Computer Software and Applications Conference*, volume 1, pages 189–195, 2007.

[38] C. L. Pacheco, I. A. Garcia, J. A. Calvo-Manzano, and M. Arcilla. A proposed model for reuse of software requirements in requirements catalog. *Journal of Software: Evolution and Process*, 27(1):1–21, 2015.

[39] Fabiane Barreto Vavassori Benitti and Rodrigo Cezario da Silva. Evaluation of a systematic approach to requirements reuse. *Journal of Universal Computer Science*, 19(2):254–280, 2013.

[40] He Zhang, Juan Li, Liming Zhu, Ross Jeffery, Yan Liu, Qing Wang, and Mingshu Li. Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, 56(1):40–53, 2014.

[41] Muesluem Atas, Ralph Samer, and Alexander Felfernig. Automated identification of type-specific dependencies between requirements. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 688–695. IEEE, 2018.

[42] Ralph Samer, Martin Stettinger, Müslüm Atas, Alexander Felfernig, Guenther Ruhe, and Gouri Deshpande. New approaches to the identification of dependencies between requirements. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1265–1270. IEEE, 2019.

[43] Gouri Deshpande, Quim Motger, Cristina Palomares, Ikagarjot Kamra, Katarzyna Biesialska, Xavier Franch, Guenther Ruhe, and Jason Ho. Requirements dependency extraction by integrating active learning with ontology-based retrieval. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 78–89. IEEE, 2020.

[44] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E.M. Van Der Werf, Sjaak Brinkkemper, and DIdar Zowghi. Behavior-driven requirements traceability via automated acceptance tests. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, pages 431–434. IEEE, 2017.

[45] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, E. Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. REquirements TRacing On target (RETRO): Improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[46] Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Bjorn Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Softw.*, 22(1):32–39, January 2005.

[47] Wentao Wang, Nan Niu, Hui Liu, and Zhendong Niu. Enhancing automated requirements traceability by resolving polysemy. In *Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018*, pages 40–51. IEEE, 2018.

[48] Wentao Wang, Arushi Gupta, Nan Niu, Li Da Xu, Jing Ru C. Cheng, and Zhendong Niu. Automatically Tracing Dependability Requirements via Term-Based Relevance Feedback. *IEEE Transactions on Industrial Informatics*, 14(1):342–349, 2018.

[49] Jun Lin, Chan Chou Lin, Jane Cleland-Huang, Raffaella Settimi, Joseph Amaya, Grace Bedford, Brian Berenbach, Oussama Ben Khadra, Chuan Duan, and Xuchang Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, page 356–357, USA, 2006. IEEE Computer Society.

[50] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[51] Andrea De Lucia, Rocco Oliveto, and Paola Sgueglia. Incremental approach and user feedbacks: A silver bullet for traceability recovery. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, ICSM '06, page 299–309, USA, 2006. IEEE Computer Society.

[52] Adnane Ghannem, Mohamed Salah Hamdi, Marouane Kessentini, and Hany H. Ammar. Search-based requirements traceability recovery: A multi-objective approach. In *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, pages 1183–1190. IEEE, 2017.

[53] Reihaneh H Hariri and Erik M Fredericks. Towards Traceability Link Recovery for Self-Adaptive Systems. pages 762–766, 2016.

[54] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: from problem to solution space. In *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 182–191, 2005.

[55] Luis C. Lamb, Waraporn Jirapanthong, and Andrea Zisman. Formalizing traceability relations for product lines. In *Proceedings - International Conference on Software Engineering*, pages 42–45, 2011.

[56] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana, and Paul Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[57] Alessio Ferrari, Giorgio O Spagnolo, and Felice Dell Orletta. Mining Commonalities and Variabilities from Natural Language Documents. In *International Software Product Line Conference*, pages 116–120, Tokyo, Japan, 2013. ACM.

[58] Noor Hasrina Bakar, Zarinah M. Kasirun, Norsaremah Salleh, and Hamid A. Jalab. Extracting features from online software reviews to aid requirements reuse. *Applied Soft Computing Journal*, 49:1297–1315, 2016.

[59] Mostafa Hamza and Robert J Walker. Recommending Features and Feature Relationships from Requirements Documents for Software Product

Lines. In *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 25–31. IEEE, 2015.

[60] J A R Noppen, P M van den Broek, N Weston, and A Rashid. Modelling Imperfect Product Line Requirements with Fuzzy Feature Diagrams. In *Third International Workshop on Variability Modelling of Software-intensive Systems, Sevilla, Spain*, number 29, pages 93–102, 2009.

[61] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. A framework for constructing semantically composable feature models from natural language requirements. In *Proceedings of the 13th International Software Product Line Conference*, pages 211–220, 2009.

[62] Nili Itzik and Iris Reinhartz-Berger. Sova - a tool for semantic and ontological variability analysis. In *CAiSE*, 2014.

[63] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of software: Evolution and Process*, 25(1):53–95, 2013.

[64] Jabier Martinez, Tewfik Ziadi, Mike Papadakis, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Feature location benchmark for software families using eclipse community releases. In *International Conference on Software Reuse*, pages 267–283. Springer, 2016.

[65] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. Sniafl: Towards a static noninteractive approach to feature location. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2):195–226, 2006.

[66] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–432, 2007.

[67] Berima Andam, Andreas Burger, Thorsten Berger, and Michel RV Chaudron. Florida: Feature location dashboard for extracting and visualizing

feature traces. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*, pages 100–107, 2017.

[68] Parisa Moslehi, Bram Adams, and Juergen Rilling. A feature location approach for mapping application features extracted from crowd-based screencasts to source code. *Empirical Software Engineering*, 25(6):4873–4926, 2020.

[69] Abdul Razzaq, Andrew Le Gear, Chris Exton, and Jim Buckley. An empirical assessment of baseline feature location techniques. *Empirical Software Engineering*, 25(1):266–321, 2020.

[70] Patrik Berander and Anneliese Andrews. *Requirements Prioritization*, pages 69–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[71] Thomas L Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[72] Muhammad Aasem, Muhammad Ramzan, and Arfan Jaffar. Analysis and optimization of software requirements prioritization techniques. In *2010 International Conference on Information and Emerging Technologies*, pages 1–6. IEEE, 2010.

[73] Thomas Bebensee, Inge van de Weerd, and Sjaak Brinkkemper. Binary priority list for prioritizing software requirements. In *International working conference on requirements engineering: foundation for software quality*, pages 67–78. Springer, 2010.

[74] Paolo Tonella, Angelo Susi, and Francis Palma. Interactive requirements prioritization using a genetic algorithm. *Information and software technology*, 55(1):173–187, 2013.

[75] Xavier Franch, Lidia López, Carlos Cares, and Daniel Colomer. The i* framework for goal-oriented modeling. In *Domain-specific conceptual modeling*, pages 485–506. Springer, 2016.

[76] Fei Shao, Rong Peng, Han Lai, and Bangchao Wang. Drank: A semi-automated requirements prioritization method based on preferences and dependencies. *Journal of Systems and Software*, 126:141–156, 2017.

[77] Muhammad Abbas, Mehrdad Saadatmand, Eduard Enoiu, Daniel Sundamark, and Claes Lindskog. Automated reuse recommendation of product line assets based on natural language requirements. In Sihem Ben Sassi, Stéphane Ducasse, and Hafedh Mili, editors, *Reuse in Emerging Software Engineering Practices*, pages 173–189, Cham, 2020. Springer International Publishing.

[78] Muhammad Abbas, Alessio Ferrari, Anas Shatnawi, Eduard Paul Enoiu, and Mehrdad Saadatmand. Is requirements similarity a good proxy for software similarity? an empirical investigation in industry. In Fabiano Dalpiaz and Paola Spoletini, editors, *Requirements Engineering: Foundation for Software Quality*, pages 3–18, Cham, 2021. Springer International Publishing.

[79] Muhammad Abbas, Irum Inayat, Naila Jan, Mehrdad Saadatmand, Eduard Paul Enoiu, and Daniel Sundmark. Mbrp: Model-based requirements prioritization using pagerank algorithm. In *The 26th Asia-Pacific Software Engineering Conference*, December 2019.

[80] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. Are students representatives of professionals in software engineering experiments? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 666–676. IEEE, 2015.

Address: P.O. Box 883, SE-721 23 Västerås. Sweden
Address: P.O. Box 325, SE-631 05 Eskilstuna. Sweden
E-mail: info@mdh.se Web: www.mdh.se

ISBN 978-91-7485-504-3
ISSN 1651-9256