Noname manuscript No.
(will be inserted by the editor)

On the Relationship between Similar Requirements and Similar Software

A Case Study in the Railway Domain

Muhammad Abbas · Alessio Ferrari · Anas Shatnawi · Eduard Enoiu · Mehrdad Saadatmand · Daniel Sundmark

Received: date / Accepted: date

Abstract [**Context**] Recommender systems for requirements are typically built on the assumption that similar requirements can be used as proxies to retrieve similar software. When a stakeholder proposes a new requirement, 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 between requirements are available. However, there is little empirical evidence on their effectiveness for code retrieval. [**Method**] This study compares different NLP approaches, from lexical ones to semantic, deep-learning techniques, and correlates the similarity among requirements with the similarity of their associated software. The evaluation is conducted on real-world requirements from two industrial projects from a railway company. In addition, we perform a focus group with members of the company to collect qualitative data. [**Results**] Results show that a moderately positive correlation exists between requirements similarity and software similarity, with the *tf-idf* language model outperforming advanced deep learning ones. Practitioners confirm that requirements similarity is normally regarded as a proxy for software similarity. However, they also highlight that additional aspects come into play when deciding code reuse, e.g., domain/project knowledge, information coming from test cases, and trace links. [**Contribution**] Our work is among the first ones to explore the relationship between requirements and software similarity from a quantitative and qualitative standpoint. 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 change impact analysis.

Keywords Requirements Similarity · Software Similarity · Correlation · Perception of similarity · Language Models

This work has been supported by and received funding from the ITEA3 XIVT, and KK Foundation's ARRAY project.

M. Abbas (✉)
RISE Research Institutes of Sweden, Västerås, Sweden
E-mail: muhammad.abbas@ri.se

A. Ferrari (✉)
CNR-ISTI, Pisa, Italy
E-mail: alessio.ferrari@isti.cnr.it

A. Shatnawi
Berger-Levrault, Montpellier, France
E-mail: anas.shatnawi@berger-levrault.com

E. Enoiu
Mälardalen University, Västerås, Sweden
E-mail: eduard.paul.enoiu@mdh.se

M. Saadatmand
RISE Research Institutes of Sweden, Västerås, Sweden
E-mail: mehrdad.saadatmand@ri.se

D. Sundmark
Mälardalen University, Västerås, Sweden
E-mail: daniel.sundmark@mdh.se

1 Introduction

Recommender systems have been widely studied in requirements engineering (RE) [32, 60, 40], and several diverse applications of this paradigm have been proposed in the literature. These include stakeholder recommendation for requirements discussions [19], refactoring recommendation based on feature requests [58] and bid management [32]. One typical application scenario of recommender systems in RE is related to *requirements retrieval* [43, 23]. Specifically, when a new requirement is proposed, the requirements analyst looks for reuse

opportunities and compares the new proposal with existing requirements in order to adapt their previously developed models and implementations [48, 74]. This can be supported by content-based recommender systems [52], which, given a new requirement, return the most similar ones in a historical database of product releases, together with the associated artifacts. The rationale of the approach is that similar requirements can be used as proxies to retrieve similar software, i.e., code that can be adapted with little effort to address the new needs.

Different NLP techniques exist to compute requirements similarity, and the recent emerging of novel NLP language models provides promising options [85]. However, it is unclear to which extent requirements similarity implies software similarity and what are the most effective techniques to support requirements similarity computation in a way that is optimized for code retrieval. Furthermore, little is known about the viewpoint of practitioners on this matter.

This paper aims to empirically study the problem in the context of the requirements of Alstom Transport AB (Alstom), a world-leading railway company, which aims to improve its code reuse process by means of requirement-based software retrieval. To study the relationship between requirements similarity and software similarity in this setting, we consider 254 real-world requirements related to two Power Propulsion Control (PPC) projects. We consider different state-of-the-art language models to semantically represent the requirements and support similarity computation, namely the traditional *tf-idf*, the Jaccard similarity metric [53], and the more advanced Doc2Vec [50], FastText [14], Bidirectional Encoder Representations from Transformers (BERT) [28], and the Universal Sentence Encoder (USE) [20]. We complement this quantitative analysis with a focus group involving participants from the company. Specifically, the data from the quantitative analysis are used to trigger discussion around the topic of requirements-based code reuse.

Our results show that, in our context, the traditional *tf-idf* model is the one that leads to the highest correlation with the software similarity, computed with JPLag [66]. Furthermore, we show that, with the exception of the Doc2Vec case, the correlation between requirements similarity and code similarity is moderate. This provides some evidence that similar implementations realize similar requirements in the context of the considered case study. On the other hand, it also suggests that there is further space for research about novel methods to retrieve similar software that goes beyond requirements similarity. The evidence is confirmed by the viewpoint of practitioners, who clearly state that similar requirements *must* be related to similar software; otherwise, something might have gone wrong in the development process. On the other hand, they also notice that requirements are only the starting point for code reuse. Domain/project knowledge, conversations involving different profiles, analysis of trace links, inspection of test cases, and other aspects play a crucial role in deciding reuse opportunities.

The work presented in this paper is an extension of an earlier conference contribution [1]. With respect to the previous work, we made the following substantial extensions: (i) we added two more NLP metrics to measure requirements similarity, a lexical one (Jaccard) and a deep learning-based one (USE); (ii) we performed a focus group to collect rich qualitative data on the topic of the research, and we thus provide evidence from the voice of practitioners about requirements-based code reuse; (iii) we extended the analysis of the related works to position our contribution in RE considering other tasks in which similarity computation is essential, including traceability, and change impact analysis.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the background of the requirements similarity approaches used in this paper. Section 4 discusses the research design, with context, research questions, and procedures. In Section 5, we present the results, and in Section 6 we discuss the main takeaway messages. Threats to validity are presented in Section 7. We conclude the paper and draw future directions in Section 8.

2 Related Work

In software engineering, several approaches rely on similarity measurements to analyze relationships between different software artifacts. Typical goals include feature identification [86], feature location [30], architecture recovery [74], reusable service identification [73] and clone detection [83].

In the RE field, similarity computation normally involves the usage of NLP techniques to represent the requirements [85], as these are typically written in Natural Language (NL) [33, 34, 47]. Similarity computation is key for many typical requirements management tasks, including traceability [22, 15, 39, 37], identification of equivalent requirements [31], change impact analysis [16, 9], glossary terms extraction and grouping [7], and artifact retrieval through automatic recommender systems [3, 23, 32, 60, 19, 58, 29, 40, 69]. In the following, we compare our research with representative works in RE, focusing in particular on the topics of traceability, change impact analysis, and recommender systems, which are closely related to our work, as they deal with both

requirements and software similarity. Also, we give attention to the code clone detection topic due to its relation with the source code similarly measurement used in our study.

Traceability. Requirements tracing consists in linking related artifacts of the software process, such as requirements, models, code, tests, etc., to facilitate reuse, external assessment and other management activities. Keeping trace links aligned during software evolution is particularly challenging, and information retrieval (IR) approaches have been experimented with to support this task [22, 80]. In this regard, Borg *et al.* [16] performed a systematic mapping of IR approaches to software traceability. The study considers 79 publications. The majority of them are concerned with tracing requirements to requirements (37, 47%) and requirements to code (32, 41%). The study shows that works typically use algebraic models—i.e., the vector space model and latent semantic indexing (LSI)—to support artifact representation and similarity computation between artifacts. The *tf-idf* index is by far the most common weighting scheme. Less common is experimentation with probabilistic and language-based models. The study also observes the need for more industrial case studies on the topic.

Recently, other traceability studies focus on more advanced strategies for similarity computation. Among others, Guo *et al.* [39] experiment with deep learning techniques through the usage of word embedding and recurrent neural network (RNN). Their results show that these semantic-laden techniques outperform classical vector space and LSI models. Another research in this direction is performed by Wang *et al.* [82], who use artificial neural networks (multi-layer perceptron, MLP) to overcome the problem of polysemy that affects classical lexical techniques for similarity computation [25]. Further enhancement in the accuracy of similarity computation for tracing is shown by Lin *et al.* [51], who use Bidirectional Encoder Representations from Transformers (BERT) language models to trace between GIT *issues*—which can be regarded as forms of requirements—and commits in open source projects. Their work shows that BERT appears to rule-out traditional techniques in terms of performance (over 60% with respect to the vector space model. In addition, it also addresses the problem of limited annotated data that affect the performance of RNNs used by Guo *et al.* [39], thanks to the transfer learning paradigm [61].

Change Impact Analysis. Change impact analysis (CIA-) consists in estimating the consequences of a certain change in one or more artifacts produced during the software process, including requirements, in terms of refactoring effort for other artifacts. This can be based on novel requirements, identified bugs, or other sources of change [13]. Representative works in this field are those by Arora *et al.* [9] and Borg *et al.* [16]. Arora *et al.* [9] use the SEMILAR (SEMantic simILARity) toolkit [71] to experiment with different similarity metrics and select the best combinations to support inter-requirements CIA. Their work suggests that the best metrics are the *Levhenstein* distance [53], a syntactic metric, combined with *Path* [71], a semantic one. Borg *et al.* [16] reuse previous CIA information coming from an issue tracking system. This is used to build a graph that links artifacts—e.g., requirements, test cases—based on their previous changes identified by the issue tracking system. Given a novel issue, similar issues are detected in the knowledge base, and artifacts potentially impacted by the change are retrieved. Issue similarity is evaluated by means of the Lucene library [41] with the traditional LSI, which was used also by previous studies in the field (e.g., *ImpactMiner* from Gethers *et al.* [38]).

CIA is a task that heavily relies on trace links, as one change in a software artifact needs to be propagated on the related ones, and trace links can be a relevant source to channel the ripple-effect [13]. Aung *et al.* [10], presents a recent literature review on automatic trace links recovery for the purpose of CIA. In line with works focused on traceability previously surveyed by Borg *et al.* [15], the authors confirm that the most common language models to support similarity computation are vector space with *tf-idf* and LSI, Jensen & Shannon models (JSMs) [5], and Latent Dirichlet Allocation (LDA) [45, 62]. The study also shows that CIA mostly focuses on relations between textual artifacts (e.g., requirements, issues, features) and source code, followed by inter-requirements relations.

Recommender Systems. One of the seminal contributions on recommender systems in RE is the work by Natt och Dag *et al.* [23], where the *tf-idf* language model and cosine similarity are used to support retrieval of previous requirements on a large industrial dataset. The authors developed a tool called ReqSimile, which reaches a recall of around 50% for the top-10 requirements. This is estimated to save considerable time in the given industrial context when compared to keyword-based search. Dumitru *et al.* [29] describe an approach for feature recommendation based on online product descriptions. With the support of association rule mining and kNN (nearest neighbor) clustering, they use vector-based representation with *tf-idf*. Given a novel product description, the approach

mines Softpedia.com, and proposes possible features based on similar products in the market. A combination of *k*NN clustering and *tf-idf* is also used by Castro-Herrera *et al.* [19] to recommend relevant stakeholders to requirements discussion forums based on their expertise. Similarity measures are computed based on expressed stakeholder needs and different stakeholder preferences.

The OpenReq EU project [60,32] aims to take a more holistic perspective, with recommendations in elicitation, specification, and analysis, and also includes a proposal for bid management. The researchers plan to use content-based recommender systems for requirements and adopt vector-space language models to support similarity computation. The project has released a specific service for similarity computation among requirements, which is based on the *tf-idf* metrics. The service is made available on GitHub[1].

On a different note, Nyamawe *et. al.* [58,59] recommend refactorings based on new feature requests. The recommended refactorings are based on the history of the previously requested features, applied refactorings, and code smells information. The approach is applied to issue tracking systems, and, as in previous works, the *tf-idf* vector-space model is used to compute the similarity between feature requests.

With a focus on requirements dependencies, Samer *et al.* [72] compare different approaches to detect whether two requirements have a dependency relationship or not. The authors use *tf-idf* and Pointwise Mutual Information (PMI) to support similarity computation, aided with machine learning algorithms. The best performance in this context is achieved with PMI and Random Forest classification. Still on recommendation systems for requirements dependencies, Ninaus *et al.* [57] developed *Intellireq*, an interactive platform that uses OpenThesaurus to improve the measurement of semantic similarity.

Finally, in a recent contribution [3], we used requirements descriptions to recommend the reuse of their implementation for new requirements. Compared to our previous work [3], which was dedicated to the whole task of software reuse, the current investigation is explicitly focusing on exploring the relationship between requirements similarity and the actual software similarity.

Clone Detection. Clone Detection Techniques (CDTs) aim to identify code clones, i.e., identical or similar code pieces that are reused within the same application or across different ones. To measure the similarity between code pieces, CDTs rely on several similarity metrics that can be calculated based on their textual, syntactical, or semantic information [79]. Ragkhitwetsagul *et al.* [67] perform a comparison between 30 code similarity approaches used to identify code clones. The textual similarity is measured by comparing code pieces in terms of text and string. Normally, they are identified as code clones if they have identical textual content [77], i.e., Type 1 of code cloning. Ito *et al.* [44] develop a web-based application of their approach that measures the code similarity based on the hash signatures identified from the code using the b-bit min-wise hashing algorithm. The syntactical similarity is calculated based on sub-tree comparison techniques between the Abstract Syntax Trees (ASTs) extracted from code pieces. Narasimhan *et al.* [56] use the CDT Eclipse plugin to extract ASTs from the C/C++ source code. Then, they rely on the Robust Tree Edit Distance Algorithm (RTED) proposed by Pawlik *et al.* [63] to identify syntactical similar code clones that can be merged into a more reusable piece of code using the abstraction pattern. To understand variability in android families, Shatnawi *et al.* [75] rely on ASTs to identify code clones allowing to analyze the commonality and variability between android applications of the same family. Based on their approach, a parameterized tool is provided to identify code clones at different levels of abstraction in which practitioners can configure different software similarity metrics. The semantic similarity is used to identify code clones that have different textual and syntactical representations but have similar behavior when the corresponding programs are executed, i.e., Type 4 code cloning. Statically, this type of similarity can be calculated using the control and data-flow analysis of program dependency graph reverse-engineered from code pieces [81]. In our empirical study, to measure the software similarity between the code pieces implementing similar requirements, we rely on JPLag [66] because it is based on a syntactical-based similarity metric that is compatible with the type of similarity realized in our case studies, it is freely and publicly available and can be executed on local machines which allow us to comply with our confidentiality agreement with the company.

Contribution. To the best of our knowledge, the work presented in this paper is the first one that compares the most recent state-of-the-art NLP techniques for requirements similarity computation in terms of their correlation with software similarity. The usage of advanced techniques based on deep learning and transfer learning follows the developments from Guo *et al.* [39] and Lin *et al.* [51]. In addition, we address the need for case studies observed, among others, by the survey of Borg *et al.* [15] in the field of traceability. Our work can be use-

[1] https://github.com/OpenReqEU/similarity-detection

ful also for the CIA area since Aung *et al.* [10] observed that most works rely on relationships between textual artifacts and source code, as in our case. Furthermore, works on the analysis of requirements dependencies can also benefit from our work, as the similarity is one of the most common dependencies identified in the empirical study by Deshpande *et al.* [26]. These contributions are particularly relevant also for the whole NLP for the RE area, as the recent survey of Zhao *et al.* [85] clearly highlights the limited experimentation with advanced NLP techniques in RE research, as well as the limited set of industrial studies.

3 Background: Measuring Requirements Similarity

Several metrics exist for measuring similarity between natural language artifacts in general and requirements in the specific [53]. Some metrics are purely lexical because they measure the term-based surface similarity between requirements. Others are more semantic-laden and aim to measure the similarity of the meaning of the requirements. In general, before measuring requirements similarity, one needs to define or learn a *language model*, which can be regarded as a statistical representation of the frequency and relationship between words in a language [12,64]. Given the text of a requirement, a language model can be used to map it into a numerical vector. Similarity among requirements boils down to measuring distance among vectors, and this is typically performed using the cosine similarity [53], which measures the cosine of the angle between the vectors. The effectiveness of the similarity computed with cosine is heavily dependent on the choice of the language model used for computing feature vectors.

In the following, we briefly describe the six language models used in our paper. These are Jaccard Similarity Index (JSI), Term Frequency Inverse Document Frequency (*tf-idf*), Doc2Vec, FastText, Bidirectional Encoder Representations from Transformers (BERT), and Universal Sentence Encoder (USE). Note that JSI is not a language model but rather a string-level similarity metric. However, we refer to it as a language model in the remainder of this paper for the simplicity of reporting.

Jaccard Similarity Index (JSI) is a numerical measurement of the intersection of common terms between two requirements divided by the union of the terms. More formally, given two requirements r and q, their JSI is computed as:

$$JSI(r,q) = \frac{|(T(r) \cap T(q))|}{|(T(r) \cup T(q)|)}$$

where $T(r)$ and $T(q)$ represent the set of terms in r and q respectively.

Term Frequency Inverse Document Frequency is based on the *tf-idf* score from IR. TF extracts term-matrix from the input requirements where the terms are treated as features, and their frequencies represent the wights of these features. Minimum and maximum term frequencies can be defined to drop irrelevant features such as potential stop-words. The term-matrix also considers the co-occurring terms (n-grams) as features. The term matrix is usually of very high dimensions, and thus dimensionality reduction techniques are used to select the top features from the matrix. Such an approach is useful when requirements share common terms.

Doc2Vec is based on the word2vec approach, where every word in a document is mapped to a vector of real numbers using a neural network. The vectors are concatenated to get vectors for the entire document, preserving the contextual and semantic information. For example, words like "simple" and "easy" would result in similar vectors. This helps in inferring feature vectors of fixed length for a variable length of requirements.

FastText is another model based on Word2Vec, where instead of learning word vectors directly, it utilizes the character level n-grams. For example, the word "run" would be divided into n-grams such as "ru," "run," "un". Such a model is useful for cases where shorter words are used. In addition, FastText also understands suffixes (such as verb ending) and prefixes (such as unhappy, where *un* is the prefix) better because it utilizes character-level information.

Bidirectional Encoder Representations from Transformers (BERT) is a recent breakthrough in language understanding researches. It is a bi-directional model based on the Transformer encoder architecture that also considers positional and contextual information of words. BERT is known for the so-called contextual embedding and is trained on BooksCorpus and the English Wikipedia with 2,500M words. Such a model could be handy for capturing the semantic of the requirements.

Universal Sentence Encoder (USE) uses the Deep Averaging Network (DAN)-based encoder for learning the representation of text. USE is optimized for learning phrase and sentence-level representation for the tasks of text classification and semantic similarity. Therefore, USE is an ideal option for semantic similarity computation in short paragraphs of text, such as requirements.

4 Study Design

This section outlines the research method used to obtain the results. This work can be regarded as an *exploratory* case study [70], oriented to understand the

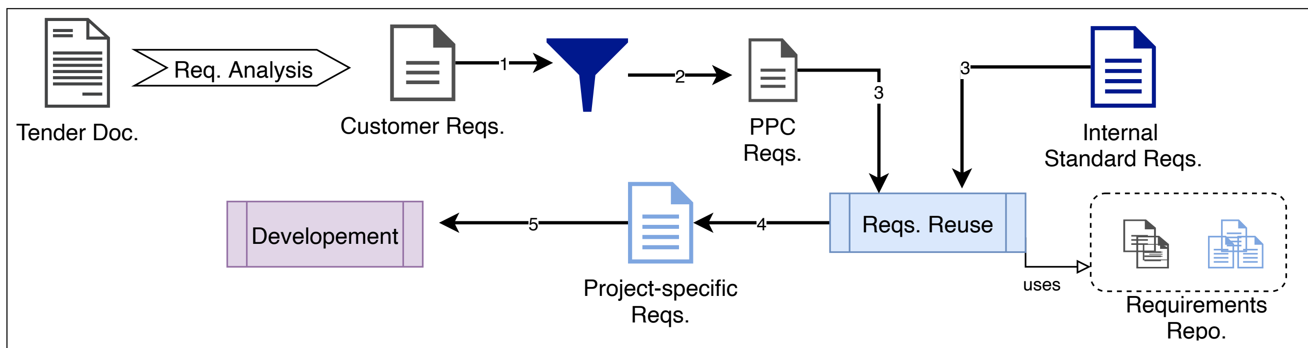

Fig. 1: The Overall Process of Receiving Requirements from a Customer

relationship between requirements and their associated code in the specific context of a railway company.

We designed this study following the guidelines of Runeson *et al.* [70] for conducting and reporting case studies. The study is designed to collect both quantitative and qualitative data to answer the research questions. Quantitative data is collected from two safety-critical projects at Alstom. To collect qualitative insights, we designed a focus group session involving Alstom engineers.

4.1 Objectives and Research Questions

Our main goal is to study the relationship between requirements similarity and software similarity in the context of requirements-based code reuse. To this end, we want to use quantitative and qualitative lenses to understand if an association can be identified between requirements similarity and software similarity so that similar requirements can be assumed to be realized by similar software. To achieve this objective, we define the following research questions (RQs).

RQ1: To what extent are similar requirements correlated with similar software in the context of requirements-based code reuse?
RQ2: How do practitioners perceive the association between similar requirements and similar software in the context of requirements-based code reuse?

RQ1 addresses the problem from a quantitative standpoint. To answer RQ1, we first measure requirements similarity between two projects with six different language models. We consider Jaccard Similarity Index (JSI), *tf-idf* (TF), Doc2Vec (DW), FastText (FT), BERT, and USE for computing requirements similarity. Then, we consider the software that implements similar requirements by means of explicit trace links, and we use JPLag to measure the similarity between the software. Finally, we compute the correlation between the different requirements similarity measures and the software similarity.

RQ2 addresses the problem from a qualitative standpoint and also aims to collect data about current practices and challenges in requirements-based software reuse. To answer RQ2, we organize a focus group with five practitioners from the company. The discussion is triggered by nine examples of pairs of similar requirements—identified by the language models— and their corresponding software. The focus group results are analyzed by means of thematic analysis, and main themes are identified concerning the current vision, common practices, and challenges in requirements-based code reuse.

4.2 Study Context

The case study is carried out within Alstom, a railway manufacturer. More specifically, the Power Propulsion Control (PPC) software development team of the company is considered for quantitative data collection. We consider both the PPC team and the Train Control and Management System (TCMS) software team for qualitative data collection. In the following, we describe the main characteristics of the two teams.

In the **PPC team**, the software is typically developed by reusing and adapting existing components from an assets base [2]. The development of a new product starts after receiving customer requirements either from different teams at the company or from customers directly. Since the system is a safety-critical software-intensive system, the requirements for all existing products can be traced to the source code. The team consists of more than 140 employees, developing safety-critical products, and thus the requirements have to be dealt with in detail. Therefore, all the team members participate in the requirements engineering activities. As shown in Figure 1, requirement analysis and elicitation activities are performed on tender documents to extract

the customer requirements. This team receives the customer requirements relevant to the propulsion system. The input requirements (PPC reqs.) are internalized by reusing standard internal domain requirements and existing requirements from other projects. This results in project-specific internal requirements to be implemented.

To support reuse, the engineers also conduct reuse analysis to identify existing similar customer requirements. The team exploits traceability information relevant to similar customer requirements to identify existing software components that could be reused to realize the new requirements. However, this process is heavily dependent on the experience of engineers and is time-consuming. Currently, the process is being automated with a recommender system called VARA [3]. Like most RE recommender systems, VARA is also based on the assumption that similar requirements can be used as proxies to retrieve similar software.

The **TCMS team** is responsible for developing the execution platform for the train applications. In the TCMS team, the requirements for the system come from different teams at the company. Unlike the PPC team, the TCMS team *does not* typically reuse software but instead focuses on evolving the existing system developed by them. Indeed, this represents a lower-level platform to support different applications, and can be regarded as a cyber-physical operating system (an enhanced firmware) specific for trains. As such, it is reused as a whole across different projects and needs to support different application-specific requirements without the need to be changed. In this team, the requirements are typically well-defined, following a structure of *Given* (a statement that specifies the current system state), *When* (a statement indicating the occurrence of a certain trigger), and *Then* (an action that is expected to be performed by the system based on the trigger).

By involving the two teams in the focus group session, we aim to collect diverse views from subjects with varying requirements engineering and software reuse practices.

4.3 Data Collection Procedure (RQ1)

Figure 2 shows a high-level view of the procedure followed for data collection. One project manager from the company was involved in validating our procedure. Two requirements documents belonging to two projects (project A and B, in the following) of the Power Propulsion Control (PPC) team were considered for this study. The project were selected based on convenience of the project managers to represent a potential scenario of requirements-based code reuse. The documents were

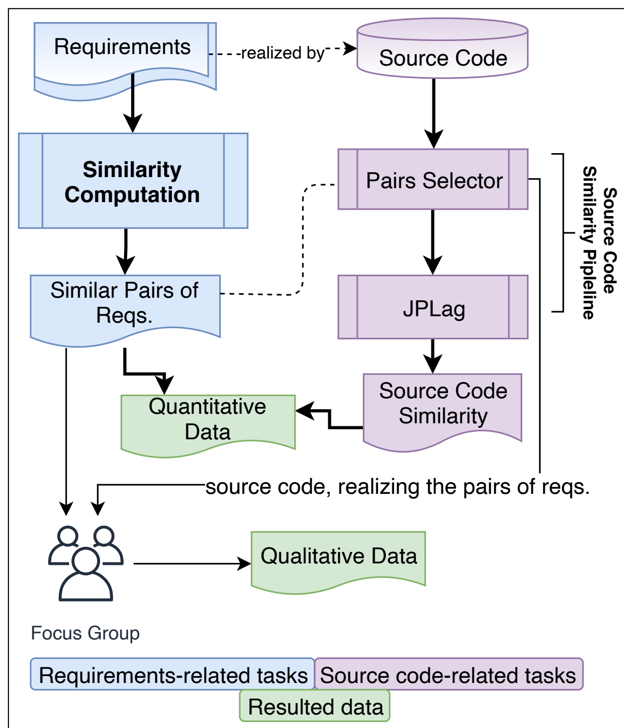

Fig. 2: Data collection procedure

subjected to cleaning to remove all non-requirements, such as headings and definitions. As a result, we consider a final set of 254 requirements—112 from project A and 142 from project B—, selected out of 265 entries. Table 1 outlines the data about the two projects with information on requirements and lines of code. The requirements were used as an input to the language models for similarity computation with and without pre-processing.

Pre-Processing. The pre-process pipeline takes the requirements text and removes English stop-words from it. After the removal of the stop-words, each token of the requirement text is tagged with Part-of-speech (POS) tags to guide the lemmatization. The pre-trained `spaCy` model[2] is used to lemmatize the text of the requirement. The output of this pipeline is the pre-processed text of the requirement. The dataset before and after pre-processing is shown in Table 1. In the remainder of this paper, the names of language models starting with "p" are the model variants where pre-processing is applied.

Language Models. In the following, we report the settings for the language models applied in our study and the specific implementations adopted in the case of pre-trained models.

[2] Available Online at `https://spacy.io/`

Table 1: Summary of the selected requirements with and without stop-words

Project	Reqs.	With		Without		SLOC
-	-	Words	AVG. Words	Words	AVG. Words	-
A	112	5823	51.9	3308	29.5	53.7K
B	142	10736	75.6	6478	45.6	61K
Total	254	16559	63.7	9786	37.5	114.7K

- **JSI:** the computation of the index has been implemented by the authors.
- **TF:** the model is configured to build the term-document matrix on project B and then uses Principal Component Analysis (PCA) [46] to select the top features based on the explained variance of 95% from the matrix. The minimum and maximum document frequencies are set to 6 and 0.5, respectively. We consider n-grams ranging from 1 to 8.
- **DW:** the pre-trained Doc2Vec model available in Gensim data[3] is used. The model has a vector size of 300, with a minimum frequency set to 2. The model is trained on the English Wikipedia documents resulting in a vocabulary size of 35,556,952.
- **FT:** we use the pre-trained FT model available in Gensim data. The model has a vector size of 100 with a minimum frequency set to 1. The model is trained on the English Wikipedia documents on the sub-word level. This results in a vocabulary size of 2,519,370. Both FT and DW are based on the skip-gram neural network architecture [55], known for contextual word prediction.
- **BERT:** we use the uncased pre-trained BERT model by Google Research [28]. The model has 12 layers and a vector size of 768. We use the BERT implementation available in BERT-as-a-service[4].
- **USE:** we use the English pre-trained model available in the TensorFlow hub [5]. The model is trained on a variety of data sources and produces a feature vector of 512 dimensions. The model does not require the input text to be pre-processed. However, to make a fair comparison, we consider the results with and without pre-processing.

Requirements Similarity Computation. We compute the similarity and identify similar requirements based on the different language models with and without pre-processing. Note that for DW, FT, USE, and BERT, the hyper-parameters are not in our control and come from the original pre-trained models. The input to each language model is a requirement, and the output is a vector. The similarity between each pair of requirements' vectors is calculated using the cosine similarity metric with the scipy implementation available in `scipy` [68]. This procedure applies to all language models, except JSI. For JSI, a pair is created by computing the index directly on the text of the requirements.

A pair of requirements is created by retrieving the most similar requirement from project B for each requirement in project A. We have chosen this approach as project B was developed *before* project A. Thus, this setting mimics a code retrieval scenario in which requirements from project A are regarded as queries, and those from project B are database items to be retrieved. Given the total number of requirements, we select the top-50 similar pairs using cosine similarity. We choose the top-50 pairs because it is a suitable number for a sample size on which to apply statistical tests, and, at the same time, does not cover all possible, likely unrelated, requirements pairs. Selecting a larger number would lead to inclusion of non-similar requirement pairs, which would not be relevant in a context of requirements retrieval such as the one under study (see RQs). The selection of top-n pairs is also common in requirements retrieval studies [23].

Code Generation. In the studied projects, the requirements are realized in Simulink models, and code is generated from the models for deployment. Therefore, in the studied setting, we used Simulink Embedded Coder[6] with the MinGW64 gmake tool-chain to generate code from the models for software similarity computation. The related code realizing each requirement was traced and moved to directories tagged with the requirement's identifiers.

Software Similarity Computation. Our software similarity pipeline takes pairs of requirement identifiers as input, and copies each pair's code to separate folders[7]. The pipeline then uses JPLag to compute the similarity between the pair of source codes. To compute the similarity between the source code of the two requirements, we use the JPLag's Java ARchive (JAR)

[3] Available at `https://github.com/RaRe-Technologies/gensim-data`

[4] Available Online at `https://github.com/hanxiao/bert-as-service`

[5] Available Online at `https://tfhub.dev/google/universal-sentence-encoder/4`

[6] The option "optimize for traceability" was selected in Embedded Coder.

[7] In our case, each folder for a pair contains two sub-folders with code of each requirement.

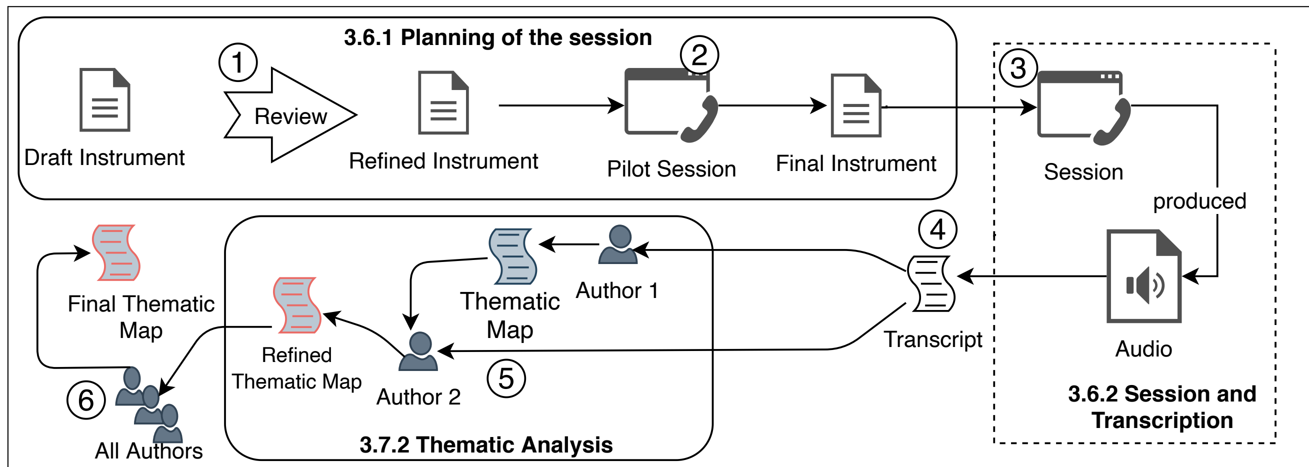

Fig. 3: An overview of the focus group planning and execution

with C/C++ as a language parameter [66]. JPLag was originally designed to detect plagiarism in students' assignments and thus is able to detect semantically similar code. Note that JPLag ignores code comments and white spaces and scans and parses the input programs to convert the programs into string tokens. JPLag then uses a greedy version of the string tiling algorithm to compute the similarity between the tokens of the source code. The similarity number is basically the percentage of similar tokens in the pairs of source codes. The output of this pipeline is the software similarity values between 0 and 100, later converted to a range between 0 and 1 for the input pairs.

The requirements similarity between pairs of requirements and the software similarity for the corresponding software modules act as the quantitative data collected to answer RQ1.

4.4 Data Analysis Procedure (RQ1)

First, we visualize the data in bar and scatter plots to provide descriptive statistics on the software similarity percentages among the identified pairs. Then, we apply correlation analysis to quantify the relationship between the two variables using R Studio[8]. As our data are not normally distributed and we do not assume any linear correlation between the variables, we use Spearman's rank correlation coefficient test.

4.5 Data Collection Procedure (RQ2)

A focus group session was planned to gather practitioners' perceptions on the association between similar requirements and similar software. The session was conducted with five practitioners (engineers and architects), selected based on convenience. Two days before the actual session, the practitioners were asked to have a look at a document containing nine pairs of similar requirements and their software, identified by three different language models. To diversify the selected pairs across the language modes, we selected three pairs from three language models with a high, mild, and low correlation of its similarity values with software similarity. The goal of the selection was not to validate the measures but rather to have a sufficient variety of cases to trigger discussion on the topic of the association between similar requirements and similar software.

The focus group session was conducted following the guidelines proposed by Breen [18]. In the remainder of this sub-section, we report our focus group protocol and the thematic analysis process, also presented in Figure 3.

4.5.1 Planning of the session

Focus Group Instrument. To ensure smooth execution of the session, a study plan was developed by all the authors. The plan contained an initial instrument with series of questions to be asked to the participants to trigger discussions. In an online session, all the authors reviewed and logically ordered the questions (see step (1) of Figure 3). Based on the refined instrument, the authors executed a pilot focus group session, as shown in step (2) of Figure 3. After the pilot session, some questions were re-ordered, and some questions were marked optional due to time limitations. The final instrument contained six questions targeted towards RQ2, presented as follows.

[8] RStudio, Available online, https://rstudio.com/

Viewpoint on Similarity

– In a software reuse context, when do you consider two requirements to be similar, and how do you evaluate them?
– In a software reuse context, when do you consider two software to be similar, and how do you evaluate that?

After the above questions, we show the selected pairs and ask the following questions.

– Do you consider these software modules to be similar? Also, how much refactoring is needed for this software to satisfy the new requirement?
– Are similar requirements representative of similar software?

Reuse Practices, Challenges, and Opportunities

– How do you identify reuse opportunities based on requirements?
– What challenges do you face in manual/automated requirements-based software reuse?

Confidentiality. Three authors of this paper (first, fourth and fifth) have a non-disclosure confidentiality agreement with the company. Two of these authors (first and fourth) recorded the session. At the start of the session, consent was obtained from the participants for audio recording. The audio recordings were transcribed, anonymized, and subsequently deleted after the analysis. The participants were made anonymous, quotes were made untraceable to the participants, but names of tools and RE practices were kept un-anonymized due to their relevance to the findings. The final report was shared with a manager of the company to ensure that the findings did not reveal any confidential information.

4.5.2 Session and Transcription.

Selected Participants. Five experts from the PPC and TCMS teams were selected to participate in the session. There was a diversity in the participants' background, gender, and role. The participants' roles vary between requirements engineering, development and testing, and project managers. All the participants have at least ten years of working experience in different software engineering and development domains.

Session. The session was conducted as an online meeting, which refers to step (3) of Figure 3. The fourth author moderated the session, while the first and fifth authors were tasked to ask follow-up questions. The session started with an introduction to the objectives of the study and the context of requirements-driven software reuse. The session was planned to be one hour and 15 minutes, and a total of one hour of audio recording was obtained after the presentation.

Transcription. The audio recording was transcribed by the first author in a document containing more than five thousand words on nine pages. This activity is represented by step (4) of Figure 3. During the transcription process, we also anonymized personal and confidential information. The transcript was also reviewed by the moderator of the session.

4.6 Data Analysis Procedure (RQ2)

Qualitative data can be analyzed following a variety of approaches. One commonly applied approach for qualitative data analysis is thematic analysis. In our case, we applied thematic analysis on the transcript of our focus group, following standard guidelines by Braun and Clarke [17]. Here, we present the commonly used terminology, followed by an overview of the thematic analysis process used in this paper:

– *Theme* is an abstraction of a commonly occurring pattern within the qualitative data.
– *Sub-Theme* is an abstraction of a sub-pattern within a theme.

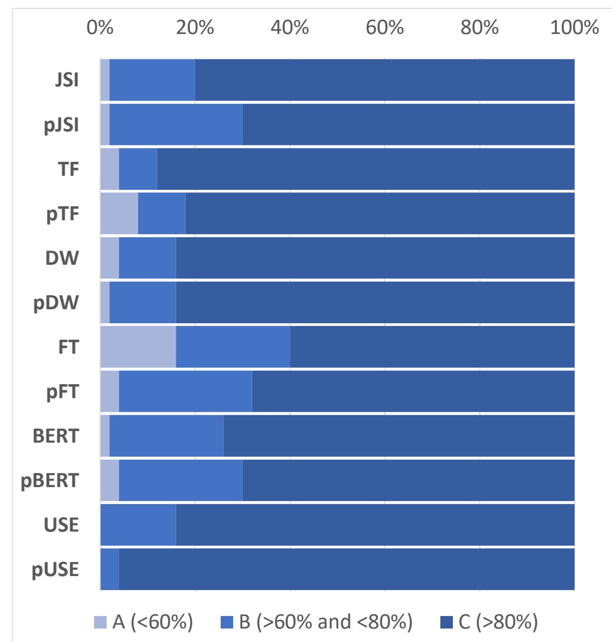

Fig. 4: Software similarity distribution in the top-50 similar requirement pairs

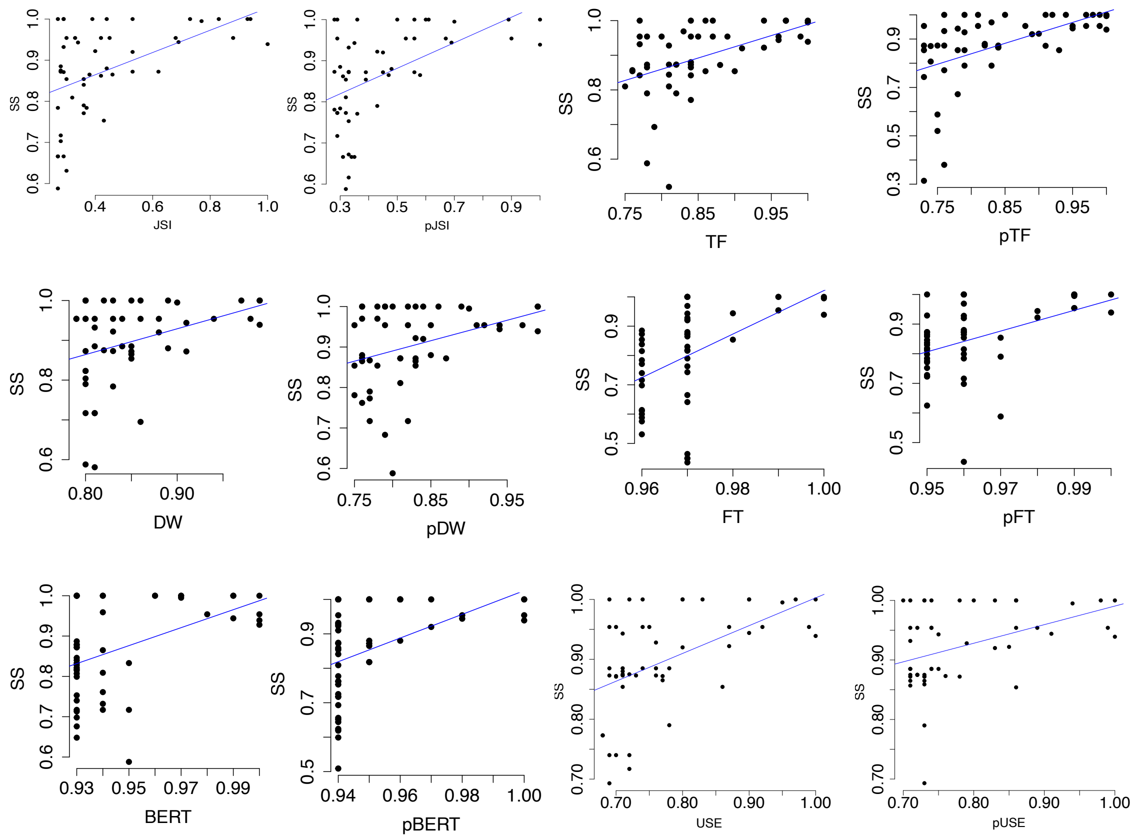

Fig. 5: Scatter plots of the requirements and software similarity

- *Codes* acts as labels assigned to chunks of qualitative data (such as sentences) for indexing.
- *Thematic Map* is a visual or tabular representation of the extracted themes, sub-theme, and codes.

The thematic analysis was performed by the first author. Following the guidelines, the author first read the transcript to get more familiar with the data. Then coding on transcript was performed, and codes were also linked to the themes and sub-themes identified during the process. As shown in step (5) of Figure 3, the fourth author reviewed and refined the final thematic map. The refined thematic map was subjected to a final review from all authors (step (6)). Finally, the results were compiled in a report (see Section 5).

5 Results

5.1 Quantitative Results (RQ1)

In this section, we present the quantitative data to answer RQ1. First, we present the descriptive statistics, and then we present the correlation analysis.

Descriptive Statistics. Figure 4 shows the distribution of software similarity among the top-50 similar pairs of requirements, based on each language model. To understand the results, for each language model, we divided the selected pairs of requirements into three classes based on the actual software similarity. The first class represents the cases in which the retrieved software shares less similarity (< 60% software similarity, A). The second class represents cases in which the retrieved software share moderate similarity (between 60 and 80% between the software of the pairs, B), finally, the third class represents cases in which the retrieved software shares high similarity (> 80% similarity between the software of the pairs, C). The above classes are defined to show the extent to which requirements similarity can be used to recommend requirements-based code reuse.

As shown in Figure 4, in all cases, in at least 60 percent of the pairs, the software similarity stays above 80 percent (i.e., class C). The USE language model retrieved no pairs with software similarity of less than 60%. In addition, the pUSE language model retrieved the highest number of pairs in class C, with more than 80% of software similarity. On the other hand, the stri-

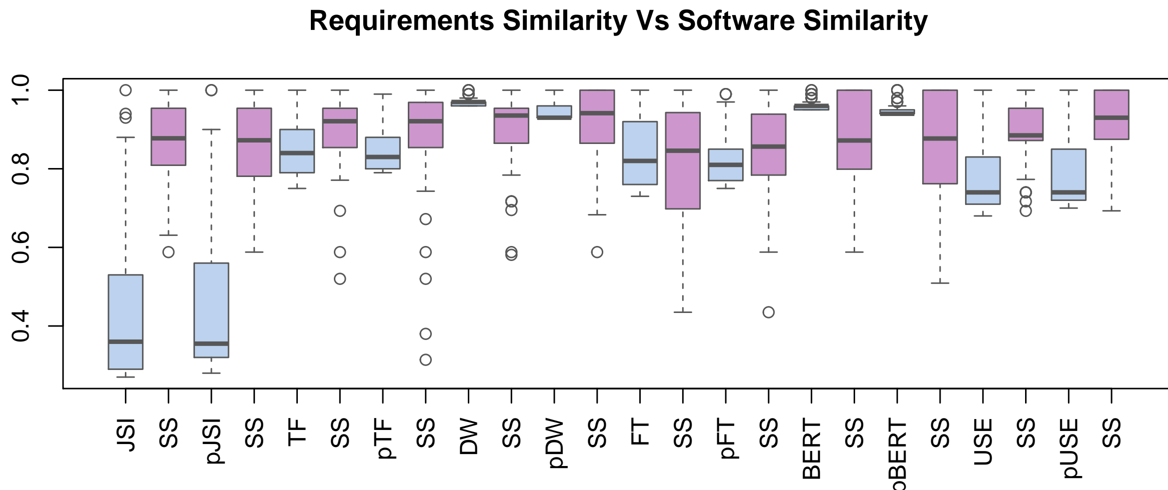

Fig. 6: Requirements Similarity (blue) and their corresponding Software Similarity (SS, purple) for all pipelines

Table 2: Spearman's rank Correlation Results with Moderate correlation in bold text. The best pipeline (pTF) is also reported in italic.

	JSI	**TF**	**DW**	**FT**	**BERT**	**USE**
rho	0.4969	**0.5089**	0.2642	**0.5718**	0.3865	**0.5235**
p-value	0.00024	0.0001	0.0636	1.439e-05	0.0055	9.554e-05
	pJSI	***pTF***	**pDW**	**pFT**	**pBERT**	**pUSE**
rho	**0.5208**	***0.5927***	0.3104	0.4676	**0.5575**	0.4151
p-value	0.0001	5.753e-06	0.0282	0.0006	2.594e-05	0.0027

ng-level lexical similarity approach, JSI, retrieved only one pair with software similarity of less than 60%.

Figure 5 presents a view of the association between the requirements similarity and software similarity for each language model. The requirements similarity is reported on the X-Axis, while the software similarity is plotted on Y-Axis and is calculated using JPLag. The blue line is the trendline between the two variables, giving insights into the relationship between them. In all cases, as can be seen from the trendlines, there could be a moderately positive association between the two variables. However, for some language models such as BERT, and FT, the variation between its similarity and software similarity is very high. For example, many pairs with varying software similarities can be seen with a requirements similarity of 0.97 in FT. The trend line for USE and pJSI seems to show more association compared to others.

In Figure 6, we also visualize the interquartile range (IQR), mean, and outliers in our variables. The boxplot shows that the software similarity for most requirement pairs stays above 70%. The boxplot also gives an overview of the requirements similarity ranges (embedding space) for the different language models. For instance, BERT tends to produce a cosine similarity value between 90% and 100%, while for FastText (FT), the embedding space ranges between 75% to 100%. This indicates that for some metrics, and especially DW and BERT, one may need to apply some scaling on the similarity values, if they want to use them to clearly distinguish between similar and non-similar requirements. On the contrary, for JSI and pJSI, requirements similarity is particularly low and shows a wider range (between 0.3 and 0.6). This measure captures exclusively the lexical similarity in terms of word overlaps, but this is nevertheless sufficient to identify similar software, as software similarity is still in the high ranges.

Correlation Analysis. We applied correlation analysis to quantify the relationship between requirements and software similarity. We measure the correlation between the similarity of the top-50 most similar pairs of the requirements and their source code similarity.

Table 2 show the results of Spearman's rank correlation. The `p-value` indicates the significance of the obtained results. The `rho` column is the correlation coefficient, which ranges from -1 to 1. As it can be observed, there is a moderately positive association (i.e., above 0.5) between the requirements similarity and software

Table 3: Themes and sub-themes relevant to the viewpoint on similarity of requirements, software and their relationship.

Theme	Sub-Theme
1. Similar requirements identification and its challenges	1.1 Requirements are perceived to be similar if they have similar logical behavior and structure 1.2 Use of synonyms affects requirements quality 1.3 Requirements' context and dependencies affects the identification of similar requirements
2. Perception of similar software	2.1 Similar software are perceived to have similar interfaces and processing on the inputs
3. Association between similar requirements and similar software	3.1 Similar requirements should ideally be realized by similar software 3.2 Requirements' structure, quality, and abstraction levels affect its association with similar software

similarity for all the language models, and the results are significant for $\alpha = 0.05$, except for DW. The highest correlation ($\rho = 0.59$) is achieved with pTF, the traditional *tf-idf* model with pre-processing, followed by FT ($\rho = 0.57$), pBERT (0.56), USE (0.52) and pJSI (0.52). As anticipated by the anslysis of the descriptive statistics, DW shows the lowest correlation ($\rho = 0.27$). Preprocessing appears to have different impact, depending on the language model used, and we cannot identify a general rule, as, e.g. TF, JSI and BERT are enhanced by preprocessing, while FT and USE are not.

5.2 Qualitative Results (RQ2)

This section presents the findings from the focus group, aiming to provide an answer to RQ2. Three main discussion topics, and associated themes, were identified based on the transcript, namely:

- **Viewpoint on Similarity:** the topic collects themes around the perception of similar requirements, similar software, and their association. The themes are reported in Table 3;
- **Reuse practices and challenges:** the topic collects themes about common practices adopted by the practitioners and challenges in requirements-driven software retrieval for reuse (Table 4).

In the following, we discuss the results according to the two main topics above. Quotes from the transcript are presented in boxes to provide evidence of the link between themes and data.

5.2.1 Viewpoint on Similarity

Theme 1. Similar requirements identification and its challenges. During the focus group, participants agreed that two requirements are considered to be similar if they are using the same input for processing and produce more or less similar outputs, and have the same logical structure. However, other contextual information, such as the system's features and interfaces, are also important to identify similar requirements. Thus, similarity evaluation between requirements, also for engineers, goes beyond the mere comparison of the surface meaning of the text.

> *"Requirements are similar if they have the same logical structure, and matching conditions, while using the same inputs and provide the same outputs." (...) "We have to look at the similarities, but it is also on the conceptual level. For example, which features that they want, do they want to bypass some features in this project? (...) these kinds of things we also consider. "*

Content-based recommender systems ease the task of identification of similar requirements using automatically computed similarity. Therefore, there are also some challenges in the automated identification of similar requirements (sub-themes 1.2 and 1.3).

Specifically, the participants observed that the quality of requirements directly affects similarity evaluation. Requirements that are non-atomic—i.e., requirements that discuss more than one topic—may be identified as similar to multiple requirements. In addition, too many details in the requirements text might also affect the identification of similar requirements. Identifying similar requirements when the input requirements are concise and describe only one topic is considered less challenging.

> *"A requirement may contain much information. While some might really describe a whole function. (...) The challenge is less in the latter case, perhaps."*

Approaches that automatically detect the non-atomic requirements are of high interest in the context of requirements-driven software reuse.

Synonyms are considered to affect the quality of requirements and, therefore, also affect the identification

of similar requirements. Requirements might be coming from different teams, written by engineers with different backgrounds and choices of terms. For example, an engineer may write "Overvoltage" or "Excessive voltage", when referring to the same concept.

> *"We have seven or eight different sources that write requirements, all from different backgrounds and at different levels of language quality. Therefore, there is a huge variability of terms in the requirements, while they are referring to the same thing."*

The use of domain-specific synonyms is considered to be common in the requirement documents of the company, especially when requirements originate from different teams. Participants also suggest a way to address this, by including the synonyms in a mapping table for resolution.

> *"Imagine you have one component that is used across 50 projects. (...)" In all 50 projects, different "terminologies have been used for that component. For example, in one document, it is called excessive voltage protection, and in another project, overvoltage protection, so you will start to see the accumulation of terminologies that are used for this component. However, we are not manually creating this terminology list."*

In some cases, the requirement text alone might not represent the whole context according to the participants. A requirement could be a derived one and may have dependencies with many other requirements, for example, when it helps to realize a larger system feature. Such a requirement might end up being similar to a new requirement but would not be a good candidate for reuse in the context of the new project, as reusing it would imply reusing several other project-dependent requirements that may not apply to the new project.

> *"Some requirements which might be very similar but in the context they are used, might be a small part of a function with many other different requirements affecting the implementation."*

Theme 2. Perception of similar software. All participants agree that software modules are perceived to be similar if they have identical inputs/interfaces, do analogous processing on the inputs, and provide similar outputs. As for analogous processing, it was clarified that they intend that the Simulink models appear to perform the same manipulation on the data, regardless of the type of representation (e.g., state machines vs. Simulink block diagrams).

> *"We look for the identical input signals, the same description of processing and output, if so, then probably the software are very similar."*

Theme 3. Association between similar requirements and similar software. Experts perceive the association between similar requirements and similar software as a ground truth. As pointed out by one of the focus group participants, if there is a low or no association between similar requirements and similar software, it might be a problem of missing trace links or poor traceability in general.

> *"Same requirements, same software. Slightly different requirements, probably slightly different software. (...) When you have small atomic requirements, which really define the input signal with no other complexity, (...) then you can actually see the similarity in software. (...) There is a correlation between requirement similarity and its implementation. If not, we have got a bit of a problem with that traceability."*

In addition, the participants observed that the more structured and formal the similar requirements are, the more likely it is to have a high association between their similarity and the corresponding software similarity. Requirements written in a structured natural language are considered to better resemble the software behavior (subtheme 3.2), thus tightening the relationship between requirements and code, and in turn, facilitating reuse.

> *"How the requirements are expressed is also a bit of interest here; if you can express it in a very clear manner, with the prerequisites and the triggering conditions that might ease reuse."*

Nevertheless, we found conflicting views on how requirements should be written. On the one hand, structuring the requirements into inputs, perquisites, and conditions (Given, When, Then structure) is perceived to be strengthening the association between similar requirements and similar software. On the other hand, some experts believe that requirements should be written in the form of free text. Because, in their view, anything in between NL requirements and software is pseudo-code.

> *"The Given, When, Then structure is an indication of pseudo code (...) It is not a true representation of natural language requirements (...) The software itself can give you all these Given, When, Then. They are interchangeable."*

Table 4: Themes and Sub-themes relevant to reuse practices and challenges.

Theme	Sub Themes
4. Relevance and benefits of reuse	4.1 Reduces development, re-certification, and testing time 4.2 Can aid the bidding process & project risk assessment 4.3 Reuse is more relevant in products with multiple deliveries or variants
5. Practices of requirements-driven software reuse	5.1 Reuse identification is based on domain knowledge and the experience of engineers 5.2 Test specifications may aid the identification of software for reuse
6. Challenges of requirements-driven software reuse	6.1 Variability management and standardization of interfaces are essential for reuse 6.2 Dependencies, context, and requirements relationships may hinder reuse 6.3 Traceability aids reuse but the granularity of links should be at the right level

5.2.2 Reuse challenges and practices

Theme 4. Relevance and benefits of reuse. Software reuse across different projects is a common practice for the PPC team and for many other similar teams in the company. During the focus group, experts discussed various benefits of reuse. Reuse at the requirements level can avoid redundant development and testing efforts. In addition, reuse can help in avoiding safety re-certification and therefore, would save time and resources. This is particularly important for safety-critical products as the railway ones, as these need to go through a structured process of verification and validation that has to be entirely repeated when the software is substantially changed.

> *"If you have validated something safety-related, reusing that without having to re-do all verification, validation, and certification (...) would save a huge amount of time."*

Identifying reuse opportunities based on requirements not only saves time but can also help the company in the bidding process of acquiring new projects. The risk of a new project can also be estimated based on the similar requirements already implemented by the companies in similar projects. In the studied setting, a manual reuse analysis is normally conducted to aid the bidding process.

> *"Today, reuse analysis is also done during the bid phase already on the higher level of requirements. We get complex hundreds of pages of technical documents, and any type of support there would be useful."*

Participants also observed that reuse is not always relevant. Specifically, this practice is not typical in those cases where one single product is developed in evolution, without multiple deliveries or versions to address diverse customer needs. For example, some participants in the focus group from the TCMS team do not reuse software in their daily work, as the team is responsible for developing a platform for the execution of other train applications. Somehow, the platform should be reused by construction as a whole, and the problem of reusing pieces of software is shifted at the application level.

> *"Our team provides a compute platform for the applications. So, we normally have a typical scenario of defining new functionality or providing support for new hardware (...) I think reuse is more relevant at the application level."*

Theme 5. Practices of requirements-driven software reuse. Identifying reuse based on input requirements can be a challenging task. When done manually, the engineer is required to have knowledge of the past projects that the company has done. This makes the process of reuse analysis dependent on the experience of the engineer. In the studied setting, three different approaches are combined for identifying reuse opportunities, as follows. The experience of the engineer is used to recall existing similar requirements to the input requirements. In addition, the engineer arranges an informal meeting over coffee with other engineers and discusses the input requirements. Finally, the engineer might also have meetings with the domain expert in the company. The domain team maintains a list of common requirements that may help realize some of the new requirements in new projects.

> *"We are usually at the coffee table with a bunch of architects looking at the requirements. (...) So, we have this coffee discussion which is one method. The second one is Dj vu; the engineer might have worked with similar requirements before, so that is the knowledge of the engineer herself. The third way is the involvement of the domain architect for the standard product (...)"*

Experts also suggest looking into the natural language test specifications linked to the requirements. A tester mainly writes test specifications, and functionally similar requirements might have similar test specifications.

> *"Does requirements similarity result in test cases being similar? Because a requirement has to be verified, and the verification methodology is similar. (...) The variable part does not matter so much anymore because you are going to verify it in the same way."*

However, for new input requirements, the test specification might not be available to be used for identifying similar requirements. Nevertheless, such an approach could be very useful in a test-driven development setting.

Theme 6. Challenges of requirements-driven software reuse. Reuse of existing software may reduce time-to-market and safety certification efforts, but there are some challenges and prerequisites to reuse. Requirements along their software might be slightly adapted to new project's needs, and therefore, variant management is essential for future reuse. In addition, the requirements adaptations in some cases results in changing just the value of some parameters. In such cases, requirements parameterization has to be done.

> *"The variability in the projects should be somehow taken into account(...) The standardization of the interfaces is the key to reuse."*

An input requirement might be implemented before, but reuse might not be possible. For example, some requirements might be part of a bigger requirement and might be implemented deep inside a bigger system function in the software. Reusing such a requirement in isolation might not be feasible without taking the dependencies into account.

> *"You might have a requirement that is very similar to the new one, but it might be a small part of a function with many different requirements affecting the software."*

It is important to note that the requirements-driven software reuse activities make use of traceability links. In many cases, companies might not maintain traceability links, and if traceability is maintained, it is mainly manual. This is also what happens at Alstom.

> *"We manually generate the traceability matrix. So, we get requirements from the requirements management tool, we then import it into Simulink and create a link with different model elements."*

Maintaining traceability between software and requirements at the right level of granularity is essential and a prerequisite to reuse. Requirements can be linked at varying granularity to the software, such as to classes, functions, and conditions. In the studied setting, the traceability links are maintained manually, and experts recommend the link to be created with testable pieces of software.

> *"You got to have that linking structure (...) We had a look at linking requirements within the functions, but if you start linking within functions, you are breaking down the testable sections of it. You cannot easily go into that function and test that little area. So, we link requirements to the functions because that is the lowest level of testing."*

6 Discussion

RQ1. To what extent are similar requirements correlated with similar software in the context of requirements-based code reuse? The correlation analysis (presented in Table 2) shows that for all language models, we were able to find a positive correlation between requirements similarity and software similarity. In particular, there is a moderately positive correlation between the similarity of the requirements computed with pJSI, TF, pTF, FT, USE, and pBERT (shown in bold text in Table 2). Results also show that pre-processing improves the correlation for all language models except FastText and USE.

Surprisingly, the decades-old term-frequency inverse document frequency (*tf-idf*) performs better than the new state-of-the-art language models. Indeed, our results indicate that the *tf-idf* language model with preprocessing shows a moderately positive correlation (with ρ of 5.92) to software similarity. In addition, the simple string-level pJSI lexical similarity approach also show a moderately positive correlation with software similarity. This can be explained by the limited vocabulary and limited terms used in the requirements of the two projects, as typical in the RE domain [36]. Also, the projects come from the same team, and the issues of synonyms previously observed may have less prominence in the considered documents. In tasks where requirements might be sharing fewer terms—e.g., in case of comparison between high-level customer requirements and low-level specifications—, the benefit of language models capturing semantics, such as BERT and USE could be more evident. The worst performance is obtained with Doc2Vec (DW and pDW). This language model works well with long documents and might not be a good candidate for RE tasks, as single requirements are typically short, but maybe beneficial in contexts where the comparison is performed between entire requirements documents. Overall, the effectiveness of simple metrics, as *tf-idf*, confirms the choice of many works in recommender systems for RE, which, as ob-

served in Section 2, use these metrics in the vast majority of the cases.

RQ1. There is a moderately positive correlation between automatically computed requirements similarity and software similarity measured with JPLag. The *tf-idf* language model with preprocessing is the one that best represents software similarity in the considered context.

Other observations in relation to RQ1 can be derived from the analysis of the descriptive statistics. The trendlines in Figure 5 visually confirm that the results from all the language models could have a positive association with software similarity. Furthermore, from the results shown in Figure 4, it can be seen that even in worst cases, requirements-based code retrieval would result in retrieving code with a high software similarity (that is more than 80%), which can be therefore a good candidate for reuse.

Observation. In the studied setting, requirements similarity can be used as a proxy for retrieving relevant software (sharing at least 80% software similarity) for reuse in at least 60% of the cases.

Different behaviors can be observed across language models. Figure 6 shows that similarity ranges largely vary between language models (e.g., BERT and DW have a very limited range with respect to the others). This suggests that having a code-retrieval system that is based on thresholds over the similarity values (e.g, consider software with requirements similarity higher than 75%) may not be the most appropriate solution. Still looking at Figure 6, we observe that the variation in the software similarity across the pairs is high in the case of FT and BERT (i.e., larger SS box plots). This suggests that these models tend to capture more nuanced semantic similarities in requirements, which may point to more fine-grained variations of the software. On the other hand, for these language models, the minimum software similarity can also be quite low, therefore indicating that the nuanced similarities in requirements can also lead to software that cannot be easily reused. These more semantically-laden representations may be more appropriate for tasks other than code retrieval, such as, e.g., requirements-to-requirements tracing, where dependencies tend to go far beyond lexical aspects. The USE language model also learns semantically-laden representation from the text but performed significantly better in terms of retrieving software based on requirements similarity. In particular, the USE language model did not retrieve any software where its similarity with existing software is less than 60%.

Observation. Different language models tend to exhibit different behaviors when measuring requirements similarity and also when used to retrieve software. FT and BERT may identify nuanced semantic requirements similarities but retrieve software that has poor similarity. USE is a semantic-laden model that tends to retrieve highly similar software.

RQ2. How do practitioners perceive the association between similar requirements and similar software in the context of requirements-based code reuse?

From the results of the focus group session, a series of main lessons have been highlighted in relation to their vision around requirements and software similarity, and their practices of code reuse. In the following, we discuss the main take-away messages, and illustrate possible developments also in relation with related works.

Similarity in Requirements-based Software Reuse. Practitioners highlight that requirements are perceived to be similar if they use comparable inputs for processing and describe the production of similar outputs. Like similar requirements, similar software is perceived to have comparable inputs/interfaces with similar processing actions on the inputs to produce similar outputs. An association between similar requirements and similar software is perceived as a *ground truth* by the experts.

Many factors, however, are observed to affect the association between similar requirements and similar software. These factors are related to the quality of the requirements, to the way requirements are expressed, and to the relationship between requirements and conceptual feature. Requirements that are expressed in an atomic way can be more easily compared with other requirements, and also be more clearly associated with software. Furthermore, despite the disagreement on the usage of controlled natural languages, requirements expressed with a clear structure in which it is easy to identify input, output and processing activity are also considered easier to compare. The usage of synonyms, generally discouraged in requirements, can also make requirements similarity evaluation harder. Finally, requirements similarity also depends on the relationship between the individual requirement and the system feature that is associated to it—i.e., requirements that are dissimilar in terms of surface text may be considered similar because they participate in the same feature.

When it comes to software reuse practices, the focus group participants also observed that requirements similarity is not the only aspect that comes into play when

deciding reuse opportunities. At the PPC team, a recommender system is being developed for identification of reuse [3], but the participants also use other strategies. Specifically, they look into test specifications, and participate into discussions to collectively recall previous experiences, and retrieve reuse-relevant software. Therefore, identification of reuse is highly dependent on the experiences of the engineers and experts' knowledge of the domain.

RQ2. According to practitioners, requirements sisimilarity *must* correspond to software similarity. Otherwise, this is regarded as a smell of poor tracing. Factors affecting the evaluation of similarity are the quality of the requirements (atomic and clear structure, absence of synonyms), and the relation between the individual requirement and the conceptual feature to which it belongs. Requirements similarity is also not sufficient to enable reuse; other processes- and knowledge-related aspects come into play when deciding software reuse.

Opportunities for Research in Similarity Computation. The observations raised in the focus group trigger a series of opportunities for further research. In particular, the association between similar requirements and similar software can be improved by enhancing the way similarity is computed among requirements.

The similarity evaluation approaches that we considered in our evaluation compute requirements similarity on the whole text without looking for inputs, conditions, processing steps, and outputs. The conceptual-level details, such as features and structural information, are also ignored. To improve similarity computation, requirements should be tagged for inputs, outputs, and conditional statements, and should be enhanced with meta-data identifying, e.g., their feature or their category. For novel requirements, one could expect to perform this task manually. For existing requirements, the task should be addressed through automated means.

In this regard, efforts have been made to **extract entities** from requirements' text for model extraction. For example, actors, use-case names, post-condition [78], and domain entities [8] can be extracted based on heuristics. Such approaches can be adapted to extract input, outputs, and conditions from requirements for meaningful similarity computation. Furthermore, these approaches could also be extended for automated structuring requirements' text into the Given, When, Then or the Easy Approach to Requirements Syntax (EARS) [54] template, as done, e.g., by Arora *et al.* [6].

Conceptual information such as the mapping between the system features and requirements could be considered for the identification of similar requirements in the context of reuse. In this sense, approaches for **requirements classification** [42, 49] can be used to automatically tag requirements based on their feature, and thus producing meta-data that can be used for similarity computation.

Synonymy is another relevant issue to address to facilitate similarity computation. Previous work has been conducted on identifying ambiguity related to **synonyms**, e.g., by Dalpiaz *et al.* [24], who, in line with Shaw and Gaines [76], refer to the usage of synonyms as *correspondence*. The literature also includes the use of machine learning for synonym resolution in the context of trace link recovery [82].

Similar atomic and concise requirements are perceived to be better associated with similar software. Non-atomic requirements might end up being similar to many input requirements and thus might affect the association between similar requirements and similar software in the context of requirements-based code reuse. Detecting **non-atomic** or compound requirements and breaking them down into multiple requirements can improve their quality, and in turn reinforce similarity computation in a code retrieval settings. Experiences with the IBM requirements quality assistant from the automotive domain show that non-atomic requirements can be detected with good accuracy [65].

As similar requirements *must* be associated to similar software, it is also important to devise strategies that incorporate software-related information within the representation of the requirements. The extraction of entities from software and other artifacts, as, e.g., test cases, can be exploited to train language models that not only account for requirements but also for their associated software.

Observation. Research on automated recognition of requirements entities (input, output, etc.), synonym detection, non-atomic requirements identification, and requirements classification can be used to enrich requirements with meta-data, and improve their quality. This will enhance requirements similarity measures to be used in the field of software reuse.

Opportunities for Research to Improve Reuse Practices. Reuse is recognised by practitioners as a fundamental practice to reduce development and verification time, and avoid safety re-certification.

Reuse analysis is also conducted in the company to support the **bidding** for acquiring new projects and assessing the risk associated with a new call for tenders. Call for tender documents are analyzed in relation to existing requirements to identify reuse opportu-

nities across projects and compute the risk of a new project. Providing support to automate this process can make the difference in facilitating the adaptation of existing systems to the requirements of a call. Recent work in this sense uses machine learning to identify requirements within tender documents [4], and can be exploited together with similarity measures to enable reuse. Similarly, developments in the field of change impact analysis [16] can be particularly relevant to support the bidding process.

Some impediments to both manual and automated requirements-driven software reuse are also observed by the participants. Specifically, adaptations to requirements and software are required to enable reuse across different projects. This results in many functional variants of software components and requirements. Variability management, requirement standardization, and parameterization is thus fundamental to manage reuse [21]. NLP approaches for **mining commonalities and variabilities** in software requirements [35, 11] can be pursued to address this goal.

The requirements' implementation might be a small part of a larger system function, and reuse in isolation might not be possible due to project-specific dependencies. Automatic identification of **requirements dependencies** [27, 72] can help to understand which requirements can be easily reused and which ones have a too rich set of dependency links.

Traceability is also a prerequisite of requirements-driven software reuse. Many companies, including Alstom, maintain trace links manually, and this process is time consuming and error prone. Approaches for **trace link recovery** [39, 22, 51] between requirements and implementation models have been largely studied, and could facilitate also reuse. It is however important to notice that, if the traceability between requirements and their software is not maintained at the right level of granularity, the retrieved software might not be relevant for reuse. We therefore foresee that approaches for detecting inconsistencies in the granularity of trace links could improve the requirements-driven software reuse process.

Observation. The reuse process can be enhanced with improved requirements similarity computation, but also with other automated practices. These include requirements extraction from call for tenders, variability mining, extraction of requirements dependencies, and trace link recovery.

7 Threats to Validity

In this section, we present validity threats according to Runeson *et al.* [70].

Construct Validity. We based the problem of software retrieval for reuse at the requirements level and provided empirical evidence on the association between requirements similarity and software similarity. In our procedure, we used pre-trained models that are heavily dependent on the quality of the training dataset. The quality of the results might differ if different pre-trained language models are considered. To mitigate potential threats to construct validity, we selected a diverse set of approaches (see Section 3) to represent the semantics of the requirements. In addition, we designed the focus instrument using terminologies known to the participant. The instrument was refined over several iterations and through a pilot focus group session.

Internal Validity. Internal validity threats affect the validity and credibility of our results. We followed standard procedure and open source implementations of the language models to mitigate potential internal validity threats. In addition, we also involved researchers from diverse backgrounds to validate the study design and execution. We also involved a technical project manager at the company in validating our quantitative data collection procedure. In the focus group, the presence of part of the authors could bias the audience. Though this could not be entirely avoided, the focus group was conducted with a pre-defined script, which was tested and re-harsed among the authors. Finally, the results from the thematic analysis were verified by a manager at the company to ensure consistency.

External Validity. Our results are based on data provided by one company using a data set of two projects developed by one team. The qualitative results are based on the perceptions of experts from two teams of the same company. Therefore, we do not claim the generalizability of our results beyond this context. In addition, our results are only limited to one level of abstraction since we do not consider multiple levels of requirement refinement. Considering guidelines for case-based generalization [84], these results might be applicable to similar contexts, as e.g., railway, aerospace or other safety-critical domains, where similar RE practices are followed. In particular, we argue that similar results can be obtained in domains with highly structured and waterfall-like processes, as the railway one. Further studies are needed, considering other abstraction levels of requirements and in different companies and domains, to generalize the results.

Reliability. Finally, we address the threats to the reliability of our results by providing enough details on the experimental setup and implementation. We designed the study following well-established guidelines, involving authors from diverse backgrounds. In addition, we also provide the R script and the similarity values between the pairs for replication purposes[9].

8 Conclusion and Future Work

Content-based recommender systems for code retrieval typically use requirements as queries to identify previously developed requirements, and in turn, reuse their implementation. These systems take the operational assumption that similar requirements can be used as proxies to retrieve similar code that can be reused with limited adaptation. This paper presents an empirical investigation on the relationship between requirements similarity and code similarity in the context of a large railway company. The goal of the work is to explore to which extent similar requirements can be considered as a proxy to retrieve similar code. We consider two related projects in the company. We use different NLP-based language models to represent the requirements and support similarity computation. Given similar requirements, we identify the associated code, and we compute code similarity with JPLag. In addition, we conducted a focus group session to gather the perceptions of experts on the association between requirements similarity and software similarity. Our analysis shows that the correlation between requirements and code similarity is moderately positive, even in the best case. Results from the thematic analysis on the transcript shows that experts perceive an association between requirements similarity and software similarity. So, a relationship exists between the two, but there is also a need for further research on language models and similarity measurement approaches so that it can better reflect software similarity. In our specific case, the language model that reflects software similarity better is the traditional *tf-idf*.

Future work will consider a broader set of possible application scenarios of recommender systems for code reuse. Avenues that we plan to explore are as follows.

- considering terms from software and test specifications, such as function and variable names, and code comments for training language models. We believe that this can help in optimizing the correlation between requirements similarity and software similarity.
- considering demarcating the inputs, outputs and processing information within the requirement's text
- considering the original tender requirements, and identify the relationship with existing requirements and associated software, to support early evaluation during bid proposal
- considering feature or refactoring requests as input queries, to support change impact analysis [9, 16]
- consider other companies and domains other than railways to increase external validity of the results
- identify when a specific language model is more appropriate to compute similarity, given the types of relationship between the format of the queries accepted by the recommender system, the characteristics of the requirements (e.g., high- *vs* low-level, functional *vs* quality), and the type of activity that is expected to be performed with the retrieved software, which can be reused, but also correct, remove, end even validate. Indeed, similarity measures and code retrieval can also be exploited to identify incorrectly traced software or missing trace links [37, 39], as well as potentially tacit requirements that are implemented in the software but are not specified.

[9] Replication package, `https://doi.org/10.5281/zenodo.4916071`

References

1. Abbas, M., Ferrari, A., Shatnawi, A., Enoiu, E.P., Saadatmand, M.: Is requirements similarity a good proxy for software similarity? an empirical investigation in industry. In: The 27th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 3–18. Springer International Publishing (2021)
2. Abbas, M., Jongeling, R., Lindskog, C., Enoiu, E.P., Saadatmand, M., Sundmark, D.: 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. ACM, New York, NY, USA (2020)
3. Abbas, M., Saadatmand, M., Enoiu, E., Sundamark, D., Lindskog, C.: Automated reuse recommendation of product line assets based on natural language requirements. In: S. Ben Sassi, S. Ducasse, H. Mili (eds.) Reuse in Emerging Software Engineering Practices, pp. 173–189. Springer International Publishing, Cham (2020)
4. Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M.: Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empirical Software Engineering **25**(6), 5454–5497 (2020)
5. Ali, N., Guéhéneuc, Y.G., Antoniol, G.: Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. IEEE Transactions on Software Engineering **39**(5), 725–741 (2012)
6. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Automated checking of conformance to requirements templates using natural language processing. IEEE transactions on Software Engineering **41**(10), 944–968 (2015)
7. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Automated extraction and clustering of requirements glossary

terms. Transactions on Software Engineering **43**(10), 918–945 (2016)
8. Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F.: Extracting domain models from natural-language requirements: Approach and industrial evaluation. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, p. 250260. ACM, New York, NY, USA (2016). DOI 10.1145/2976767.2976769. URL `https://doi-org.ep.bib.mdh.se/10.1145/2976767.2976769`
9. Arora, C., Sabetzadeh, M., Goknil, A., Briand, L.C., Zimmer, F.: Change impact analysis for natural language requirements: An nlp approach. In: International Requirements Engineering Conference (RE), pp. 6–15. IEEE (2015)
10. Aung, T.W.W., Huo, H., Sui, Y.: A literature review of automatic traceability links recovery for software change impact analysis. In: Proceedings of the 28th International Conference on Program Comprehension, pp. 14–24 (2020)
11. Bakar, N.H., Kasirun, Z.M., Salleh, N.: Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software **106**, 132–149 (2015)
12. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. The journal of machine learning research **3**, 1137–1155 (2003)
13. Bohner: Impact analysis in the software change process: a year 2000 perspective. In: 1996 Proceedings of International Conference on Software Maintenance, pp. 42–51 (1996). DOI 10.1109/ICSM.1996.564987
14. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)
15. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering **19**(6), 1565–1616 (2014). DOI 10.1007/s10664-013-9255-y
16. Borg, M., Wnuk, K., Regnell, B., Runeson, P.: Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context. IEEE Transactions on Software Engineering **43**(7), 675–700 (2016)
17. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology **3**(2), 77–101 (2006). DOI 10.1191/1478088706qp063oa
18. Breen, R.L.: A practical guide to focus-group research. Journal of Geography in Higher Education **30**(3), 463–475 (2006)
19. Castro-Herrera, C., Cleland-Huang, J., Mobasher, B.: Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. In: International Requirements Engineering Conference, pp. 37–46. IEEE (2009)
20. Cer, D., Yang, Y., yi Kong, S., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.H., Strope, B., Kurzweil, R.: Universal sentence encoder (2018)
21. Chen, L., Ali Babar, M., Ali, N.: Variability management in software product lines: a systematic review (2009)
22. Cleland-Huang, J., Gotel, O.C., Huffman Hayes, J., Mäder, P., Zisman, A.: Software traceability: trends and future directions. In: Future of Software Engineering Proceedings, pp. 55–69 (2014)
23. Natt och Dag, J., Regnell, B., Gervasi, V., Brinkkemper, S.: A linguistic-engineering approach to large-scale requirements management. IEEE software **22**(1), 32–39 (2005)
24. Dalpiaz, F., Van Der Schalk, I., Brinkkemper, S., Aydemir, F.B., Lucassen, G.: Detecting terminological ambiguity in user stories: tool and experimentation. Information and Software Technology **110**, 3–16 (2019)
25. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American society for information science **41**(6), 391–407 (1990)
26. Deshpande, G., Arora, C., Ruhe, G.: Data-driven elicitation and optimization of dependencies between requirements. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), pp. 416–421. IEEE (2019)
27. Deshpande, G., Motger, Q., Palomares, C., Kamra, I., Biesialska, K., Franch, X., Ruhe, G., Ho, J.: Requirements dependency extraction by integrating active learning with ontology-based retrieval. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 78–89. IEEE (2020)
28. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
29. Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M.: On-demand feature recommendations derived from mining public product descriptions. In: International Conference on Software Engineering, pp. 181–190 (2011)
30. Eyal-Salman, H., Seriai, A.D., Dony, C.: Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In: 2013 IEEE 14th International Conference on Information Reuse & Integration (IRI), pp. 209–216 (2013)
31. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. Transactions on Software Engineering **39**(1), 18–44 (2011)
32. Felfernig, A., Falkner, A., Atas, M., Franch, X., Palomares, C.: OpenReq: Recommender systems in requirements engineering. In: RS-BDA, pp. 1–4 (2017)
33. Fernández, D.M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.T., Greer, D., Lassenius, C., et al.: Naming the pain in requirements engineering. Empirical Software Engineering **22**(5), 2298–2338 (2017)
34. Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., Gnesi, S.: Natural language requirements processing: a 4d vision. IEEE Annals of the History of Computing **34**(06), 28–35 (2017)
35. Ferrari, A., Spagnolo, G.O., Dell'Orletta, F.: Mining commonalities and variabilities from natural language documents. In: Proceedings of the 17th International Software Product Line Conference, pp. 116–120 (2013)
36. Ferrari, A., Spagnolo, G.O., Gnesi, S.: Pure: A dataset of public requirements documents. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 502–505 (2017). DOI 10.1109/RE.2017.29
37. Gervasi, V., Zowghi, D.: Supporting traceability through affinity mining. In: International Requirements Engineering Conference (RE), pp. 143–152. IEEE (2014)
38. Gethers, M., Dit, B., Kagdi, H., Poshyvanyk, D.: Integrated impact analysis for managing software changes. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 430–440. IEEE (2012)

39. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically enhanced software traceability using deep learning techniques. In: International Conference on Software Engineering (ICSE), pp. 3–14. IEEE (2017)
40. Hariri, N., Castro-Herrera, C., Cleland-Huang, J., Mobasher, B.: Recommendation systems in requirements discovery. In: Recommendation Systems in Software Engineering, pp. 455–476. Springer (2014)
41. Hatcher, E., Gospodnetić, O., McCandless, M.: Lucene in action. Manning Greenwich (2005)
42. Hey, T., Keim, J., Koziolek, A., Tichy, W.F.: Norbert: Transfer learning for requirements classification. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), pp. 169–179. IEEE (2020)
43. Irshad, M., Petersen, K., Poulding, S.: A systematic literature review of software requirements reuse approaches. IST Journal **93**, 223–245 (2018)
44. Ito, K., Ishio, T., Inoue, K.: Web-service for finding cloned files using b-bit minwise hashing. In: 2017 IEEE 11th International Workshop on Software Clones (IWSC), pp. 1–2. IEEE (2017)
45. Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., Zhao, L.: Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. Multimedia Tools and Applications **78**(11), 15169–15211 (2019)
46. Jolliffe, I.T., Cadima, J.: Principal component analysis: a review and recent developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **374**(2065), 20150202 (2016)
47. Kassab, M., Neill, C., Laplante, P.: State of practice in requirements engineering: contemporary data. Innovations in Systems and Software Engineering **10**(4), 235–241 (2014)
48. Krueger, C.: Easing the transition to software mass customization. In: International Workshop on Software Product-Family Engineering, pp. 282–293. Springer (2001)
49. Kurtanović, Z., Maalej, W.: Automatically classifying functional and non-functional requirements using supervised machine learning. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 490–495. Ieee (2017)
50. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International conference on machine learning, pp. 1188–1196 (2014)
51. Lin, J., Liu, Y., Zeng, Q., Jiang, M., Cleland-Huang, J.: Traceability transformed: Generating more accurate links with pre-trained bert models. In: ICSE 2021, to appear (2021). URL https://arxiv.org/abs/2102.04411v2
52. Lops, P., De Gemmis, M., Semeraro, G.: Content-based recommender systems: State of the art and trends. In: Recommender systems handbook, pp. 73–105. Springer (2011)
53. Manning, C.D., Schütze, H., Raghavan, P.: Introduction to information retrieval. Cambridge university press (2008)
54. Mavin, A., Wilkinson, P., Harwood, A., Novak, M.: Easy approach to requirements syntax (ears). In: 2009 17th IEEE International Requirements Engineering Conference, pp. 317–322. IEEE (2009)
55. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
56. Narasimhan, K., Reichenbach, C., Lawall, J.: Cleaning up copy–paste clones with interactive merging. Automated Software Engineering **25**(3), 627–673 (2018)
57. Ninaus, G., Reinfrank, F., Stettinger, M., Felfernig, A.: Content-based recommendation techniques for requirements engineering. In: 2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), pp. 27–34. IEEE (2014)
58. Nyamawe, A.S., Liu, H., Niu, N., Umer, Q., Niu, Z.: Automated recommendation of software refactorings based on feature requests. In: International Requirements Engineering Conference (RE), pp. 187–198. IEEE (2019)
59. Nyamawe, A.S., Liu, H., Niu, N., Umer, Q., Niu, Z.: Feature requests-based recommendation of software refactorings. Empirical Software Engineering **25**(5), 4315–4347 (2020)
60. Palomares, C., Franch, X., Fucci, D.: Personal recommendations in requirements engineering: the openreq approach. In: International working conference on requirements engineering: foundation for software quality, pp. 297–304. Springer (2018)
61. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on knowledge and data engineering **22**(10), 1345–1359 (2009)
62. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshynanyk, D., De Lucia, A.: How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 522–531. IEEE (2013)
63. Pawlik, M., Augsten, N.: Rted: A robust algorithm for the tree edit distance. Proceedings of the VLDB Endowment **5**(4) (2011)
64. Ponte, J.M., Croft, W.B.: A language modeling approach to information retrieval. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, pp. 275–281 (1998)
65. Post, A., Fuhr, T.: Case study: How well can ibm's" requirements quality assistant" review automotive requirements? In: REFSQ Workshops (2021)
66. Prechelt, L., Malpohl, G., Philippsen, M., et al.: Finding plagiarisms among a set of programs with jplag. J. UCS **8**(11), 1016 (2002)
67. Ragkhitwetsagul, C., Krinke, J., Clark, D.: A comparison of code similarity analysers. Empirical Software Engineering **23**(4), 2464–2519 (2018)
68. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pp. 45–50. ELRA (2010)
69. Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T. (eds.): Recommendation Systems in Software Engineering. Springer (2014)
70. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)
71. Rus, V., Lintean, M., Banjade, R., Niraula, N.B., Stefanescu, D.: Semilar: The semantic similarity toolkit. In: Proceedings of the 51st annual meeting of the association for computational linguistics: System demonstrations, pp. 163–168 (2013)
72. Samer, R., Stettinger, M., Atas, M., Felfernig, A., Ruhe, G., Deshpande, G.: New approaches to the identification of dependencies between requirements. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), pp. 1265–1270. IEEE (2019)
73. Shatnawi, A., Seriai, A., Sahraoui, H., Ziadi, T., Seriai, A.: Reside: Reusable service identification from software families. JSS **170**, 110748 (2020)

74. Shatnawi, A., Seriai, A.D., Sahraoui, H.: Recovering software product line architecture of a family of object-oriented product variants. Journal of Systems and Software **131**, 325–346 (2017)
75. Shatnawi, A., Ziadi, T., Mohamadi, M.Y.: Understanding source code variability in cloned android families: an empirical study on 75 families. In: 2019 26th Asia-Pacific Software Engineering Conference (APSEC), pp. 292–299. IEEE (2019)
76. Shaw, M.L., Gaines, B.R.: Comparing conceptual structures: consensus, conflict, correspondence and contrast. Knowledge acquisition **1**(4), 341–363 (1989)
77. Tang, W., Chen, D., Luo, P.: Bcfinder: A lightweight and platform-independent tool to find third-party components in binaries. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp. 288–297. IEEE (2018)
78. Tiwari, S., Ameta, D., Banerjee, A.: An approach to identify use case scenarios from textual requirements specification. ISEC'19. ACM, New York, NY, USA (2019). DOI 10.1145/3299771.3299774
79. Walker, A., Cerny, T., Song, E.: Open-source tools and benchmarks for code-clone detection: past, present, and future trends. ACM SIGAPP Applied Computing Review **19**(4), 28–39 (2020)
80. Wang, B., Peng, R., Li, Y., Lai, H., Wang, Z.: Requirements traceability technologies and technology transfer decision support: A systematic review. Journal of Systems and Software **146**, 59–79 (2018)
81. Wang, M., Wang, P., Xu, Y.: Ccsharp: An efficient three-phase code clone detector using modified pdgs. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pp. 100–109. IEEE (2017)
82. Wang, W., Niu, N., Liu, H., Niu, Z.: Enhancing automated requirements traceability by resolving polysemy. In: 2018 IEEE 26th International Requirements Engineering Conference (RE), pp. 40–51. IEEE (2018)
83. White, M., Tufano, M., Vendome, C., Poshyvanyk, D.: Deep learning code fragments for code clone detection. In: International Conference on Automated Software Engineering (ASE), pp. 87–98. IEEE (2016)
84. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Science of computer programming **101**, 136–152 (2015)
85. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. ACM Comput. Surv. **54**(3) (2021). DOI 10.1145/3444689. URL `https://doi.org/10.1145/3444689`
86. Ziadi, T., Frias, L., da Silva, M.A.A., Ziane, M.: Feature identification from the source code of product variants. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 417–422. IEEE (2012)