Is Requirements Similarity a Good Proxy for Software Similarity? An Empirical Investigation in Industry[⋆]

Muhammad Abbas[1,2] ✉, Alessio Ferrari[3], Anas Shatnawi[4], Eduard Paul Enoiu[2], and Mehrdad Saadatmand[1]

[1] RISE Research Institutes of Sweden, Västerås, Sweden, {first.last}@ri.se
[2] Mälardalens University, Västerås, Sweden, {first.last}@mdh.se
[3] CNR-ISTI, Pisa, Italy, alessio.ferrari@isti.cnr.it
[4] Berget-Levrault, Montpellier, France, {first.last}@berget-levrault.com

Abstract. [**Context and Motivation**] Content-based recommender systems for requirements are typically built on the assumption that similar requirements can be used as proxies to retrieve similar software. When a new requirement is proposed by a stakeholder, natural language processing (NLP)-based similarity metrics can be exploited to retrieve existing requirements, and in turn identify previously developed code. [**Question/problem**] Several NLP approaches for similarity computation are available, and there is little empirical evidence on the adoption of an effective technique in recommender systems specifically oriented to requirements-based code reuse. [**Principal ideas/results**] This study compares different state-of-the-art NLP approaches and correlates the similarity among requirements with the similarity of their source code. The evaluation is conducted on real-world requirements from two industrial projects in the railway domain. Results show that requirements similarity computed with the traditional *tf-idf* approach has the highest correlation with the actual software similarity in the considered context. Furthermore, results indicate a *moderate positive* correlation with Spearman's rank correlation coefficient of more than 0.5. [**Contribution**] Our work is among the first ones to explore the relationship between requirements similarity and software similarity. In addition, we also identify a suitable approach for computing requirements similarity that reflects software similarity well in an industrial context. This can be useful not only in recommender systems but also in other requirements engineering tasks in which similarity computation is relevant, such as tracing and categorization.

Keywords: Requirements Similarity · Software Similarity · Correlation

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

1 Introduction

Recommender systems have been widely studied in requirements engineering (RE) [14, 28, 19], and several diverse applications of this paradigm have been proposed in the literature. These include stakeholder recommendation for requirements discussions [8], refactoring recommendation based on feature requests [27] and also bid management [14]. One typical application scenario of recommender systems in RE is related to *requirements retrieval* [20, 9] in reactive software product line engineering (SPLE) [35, 22]. With SPLE, companies manage software reuse in a structured way to satisfy multiple variations of customer requirements while minimizing development effort [29]. Specifically, in a reactive SPLE context [22], 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. This can be supported by content-based recommender systems [24], 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 semantic requirements similarity, and the recent emerging of novel NLP language models provides promising options [38]. 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. This paper aims to empirically study the problem in the context of the requirements of Bombardier Transportation AB (BT), a world-leading railway company. The main objective of this study is to improve the requirement-based software retrieval process in the studied setting. To study the relationship between requirements similarity and software similarity, 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* [25], and the more advanced Doc2Vec [23], FastText [5], and Bidirectional Encoder Representations from Transformers (BERT) [10]. Surprisingly, 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 [30]. 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 but also suggests that there is further space for research about novel methods to retrieve similar software that goes beyond requirements similarity.

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

2 Related Work

In software engineering, several approaches rely on similarity measurements to analyze relationships between different software artifacts. Typical goals include feature identification [39], feature location [12], architecture recovery [35], reusable service identification [34] and clone detection [36]. In the RE field, similarity computation normally involves using NLP techniques to represent the requirements [38], as these are typically written in NL [15]. Computation of similarity is key for many typical requirements management tasks, including traceability [6, 18, 17], identification of equivalent requirements [13], clustering [3], and also recommender systems based on Information Retrieval (IR) approaches [2, 9, 14, 28, 8, 27, 11, 19, 32]. As our research is focused on this latter group of applications, we will compare our work with representative ones in this area. One of the seminal contributions is the work by Natt och Dag *et al.* [9], where the *tf-idf* language model and cosine similarity are used to support retrieval of previous requirements on a large industrial dataset. Dumitru *et al.* [11] propose an approach for feature recommendation based on online product descriptions, with the support of association rule mining and kNN clustering. This type of clustering is also used by Castro-Herrera *et al.* [8], who proposes to recommend relevant stakeholders to requirements discussion forums based on their expertise. The OpenReq EU project [28, 14] aims to take a more holistic perspective, with recommendations in elicitation, specification, and analysis, and also includes a proposal for bid management. Similarly to our work, the researchers plan to use content-based recommender systems for requirements and adopt vector-space language models to support similarity computation. On a different note, but still using *tf-idf* to support similarity computation, Nyamawe *et. al.* [27] recommend refactoring based on new feature requests. Finally, in a recent contribution [2], we used requirements descriptions to recommend the reuse of their implementation for new requirements.

Compared to our previous work [2], 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. With respect to other related studies, the work presented in this paper is the first one that, while focusing on requirements and software similarity, compares the most recent state-of-the-art NLP techniques to support similarity computation and applies these techniques in an industrial context. This is particularly relevant also for the whole NLP for the RE field, as the recent survey of Zhao *et al.* [38] clearly highlights limited experimentation with advanced NLP techniques in RE research.

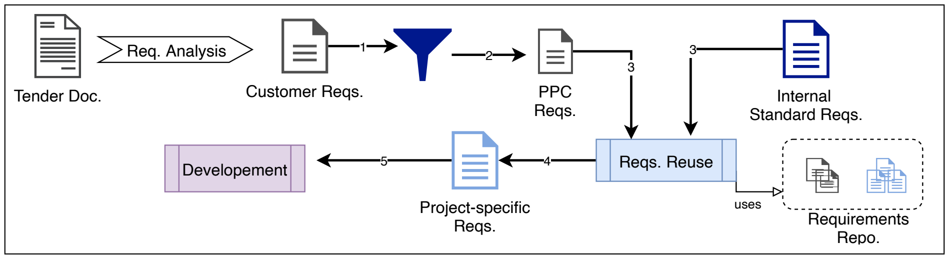

Fig. 1: The Overall Process of Receiving Requirements from a Customer

3 Study Design

This section outlines the research method used to obtain the results. This work can be regarded as an *exploratory* case study [33], oriented to understand the relationship between requirements and their associated code and exploit this relationship for software retrieval in the specific context of a railway company. We designed this study following the guidelines of Runeson *et al.* [33] for conducting and reporting case studies by providing an overview of the context, our objective, and research questions, followed by the data collection and analysis procedures.

3.1 Study Context

We have studied the PPC software development team of BT. In this team, the software is developed by reusing existing components from the assets base [1]. The development of a new product starts after receiving customer requirements from different teams at the company. 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. The customer requirements relevant to the propulsion system are received by this team. 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 and, by exploiting traceability information, 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 [2]. Like most RE recommender systems, VARA is also based on the assumption that similar requirements can be used as proxies to retrieve similar software.

3.2 Objective and Research Questions

Our main goal is to improve the software retrieval process in the studied setting. To this end, we need first to check the typical assumption of content-based recommender systems for requirements, i.e., that similar requirements can be used as proxies to retrieve similar software. In other terms, we want to check if a relationship can be identified between requirements similarity and software similarity so that similar requirements can be assumed to point to similar code. Then, we want to check which NLP approach performs best in exploiting this similarity. To achieve these objectives, we define the following research question.

RQ: To which extent can we use requirements similarity, automatically computed through different language models, as a proxy for software similarity?

This research question aims at exploring the relationship between requirements similarity and software similarity. Language models are commonly used to compute the similarity among requirements. Therefore, this research question also aims to identify the most effective language model in our specific case for computing requirements similarity that correlates well with the software similarity and can be better exploited in the given setting.

The case under study is the relationship between requirement similarity and software similarity in the considered industrial setup. The unit under analysis in our case are the projects developed in the Power Propulsion Control software at Bombardier Transportation.

3.3 Data collection

We collected data from two projects at Bombardier Transportation AB, developed by the Power Propulsion Control software team. Due to limited access to the company's repository, the projects were selected based on convenience by a company's project manager. The requirement documents were subjected to cleaning to remove all non-requirements such as headings and definitions. This resulted in a final set of 254, selected out of 265 entries. In data collection, one project manager from the company was involved in validating our procedure. Table 1 outlines the data about two projects with information on requirements and lines of code.

We conducted the investigation for our dataset, both with and without stop-words. This is because some language models can utilize stop-words, suffix, and prefix information for learning. We use a pre-processing pipeline to remove all the English stop-words and lemmatize the words of the requirements to their roots. An example requirement from the PURE dataset before and after pre-processing is shown as follows [16].

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

Before Pre-Processing: The number of block movement in incremented of 1. The difference of time of the block movement and the previous recorded time is recorded.
After Pre-Processing: number block movement incremente 1 difference time block movement previous record time record

In the studied projects, the requirements are realized in Simulink models, and code is generated from the models for deployment. Besides, there are not many available tools for computing similarity between Simulink models. Therefore, to mimic the studied setting, we used Simulink Embedded Coder[5] with MinGW64 gmake tool-chain to generate code from the models. The related code realizing each requirement was traced and moved to directories tagged with the requirement's identifiers.

3.4 Language Models for Requirements Similarity

Language models are used to derive feature vectors from the requirements' text. Various similarity metrics are used on the vectors to compute similarity among them. The cosine similarity metric is based on the cosine angle between the vectors and is heavily used in the area of NLP. The effectiveness of the similarity computed with cosine is heavily dependent on the choice of language model used for computing feature vectors. In addition, some language models are sensitive to pre-processing, such as removal of stop-words and lemmatization. This is why we selected some of the most seminal language models and fed them the dataset with and without pre-processing applied. Particularly, we considered tf-idf (TF), Doc2Vec (DW), FastText (FT), and BERT. In addition, to see the effect of pre-processing, we combined these language models with pre-processing (pTF, pDW, pFT, and pBERT). Note that for DW, FT, and BERT, the hyper-parameters are not in our control and are coming from the original pre-trained models. In our case, the input to each language model is the requirements from two projects, and the output is vectors of requirements. Given the total number of requirements, we select the top 50 similar pairs of requirements using cosine similarity to fulfill the sample size requirement. A pair is created by retrieving the most similar requirement from project B for each requirement in project A. The similarity between each pair of requirements' vectors is calculated using

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

the cosine similarity metric implementation available in scipy [31]. In this subsection, we first present the pre-process pipeline, then the different language models used to generate vectors to compute the similarity between requirement pairs.

Pre-Process. 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 requirements text is tagged with Part-of-speech (POS) tags to guide the lemmatization. The pre-trained spaCy model[6] 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 section, the names of language models starting with “p” are the model variants where pre-processing is applied.

TF 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 the frequencies are treated as values of the features. Minimum and maximum term frequencies can be defined to drop irrelevant features such as potential stop-words. The 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 in cases where the requirements share common terms. In our case, the model is configured to build the term-document matrix on project B and then uses Principal Component Analysis (PCA) [21] 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 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. In our case, the pre-trained Doc2Vec model available in Gensim data[7] 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 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 in 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. In our case, 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, resulting in a vocabulary

[6] https://spacy.io/
[7] https://github.com/RaRe-Technologies/gensim-data

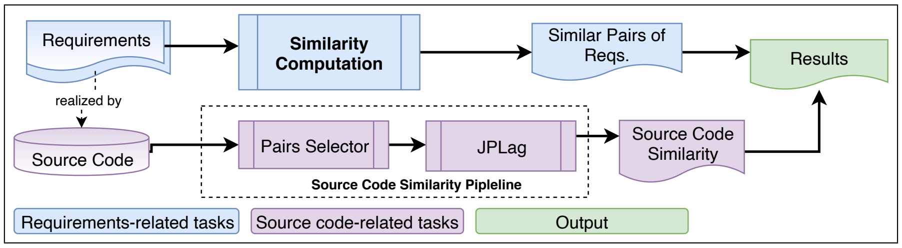

Fig. 2: Execution procedure overview

size of 2,519,370. Both FT and DW are based on the skip-gram neural network architecture [26], known for contextual word prediction.

BERT is a recent breakthrough in language understanding research. 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. In our case, we use the uncased pre-trained BERT model by Google Research [10]. The model has 12 layers and a vector size of 768. We use the BERT implementation available in BERT-as-a-service[8].

3.5 Software Similarity Pipeline

Our software similarity pipeline takes pairs of requirement's identifiers as input. It copies each pair's code to separate folders [9]. The pipeline then uses JPLag to compute the similarity between the pair of source code. To compute the similarity between the source code of the two requirements, we use the JPLag's Java ARchive (JAR) with C/C++ as a language parameter [30]. 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 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 code. The output of this pipeline is the software similarity values between 0 and 100, later converted to range between 0 and 1 for the input pairs.

3.6 Execution

Figure 2 shows a high-level view of the execution procedure followed to obtain the results. We started with two requirement documents as input to all the language

[8] Xiao Han, https://github.com/hanxiao/bert-as-service
[9] In our case, each folder for a pair contains two sub-folders with code of each requirement.

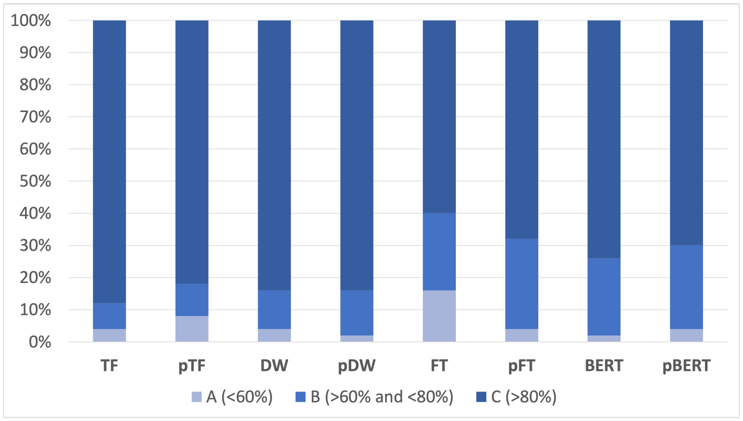

Fig. 3: Software similarity distribution in the top 50 similar requirement pairs

models presented in Section 3.4. Each language model outputs vectors of the requirements that are used to select the 50 most similar pairs of requirements based on cosine similarity. For each model, the `Pairs Selector` searches, selects and structures the code of the requirements for JPLag. The pipeline then uses JPLag to compute the similarity between each input pair of the source code and produces the software similarity values for each language models' result.

3.7 Data Analysis

First, we visualize the data in bar and scatter plots to provide descriptive statistics on the software similarity percentages among the identified pairs by using each language model. Then, we apply the correlation analysis to quantify the relationship between the two variables using R Studio [10]. 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 Results

In this section, we quantitatively answer our posed research question. First, we present the descriptive statistics, and then we present the correlation analysis.

Descriptive Statistics. To understand the results, we divided the similar pairs of the requirements—computed based on different language models—against the actual software similarity into three classes. The first class represents the cases where the retrieved software shares less similarity ($< 60\%$ software similarity, A). The second class represents cases where the retrieved software share moderate similarity (between 60 and 80% between the software of the pairs, B), finally, third class represent cases where the retrieved software shares high similarity

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

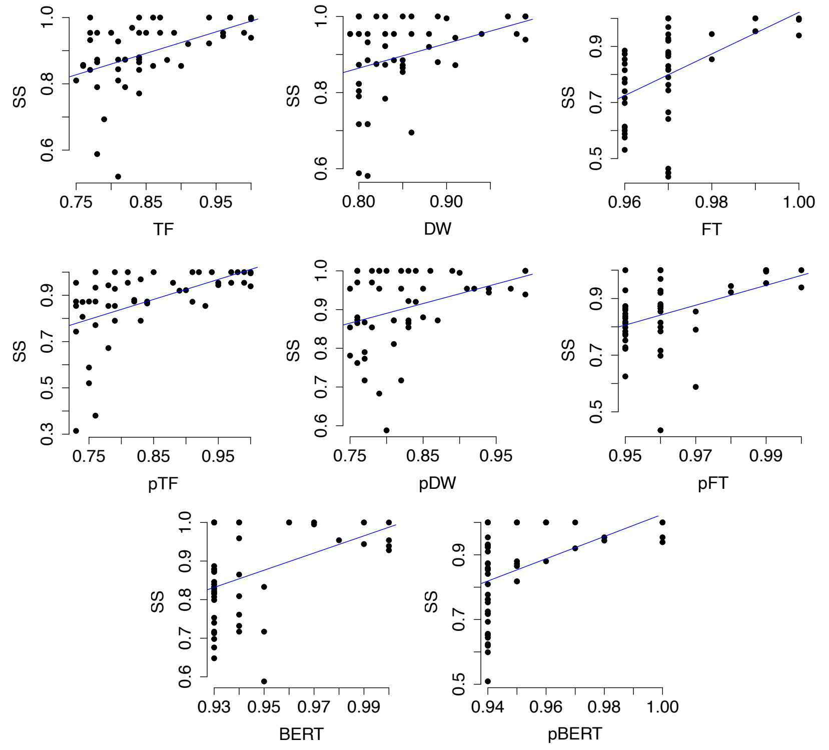

Fig. 4: Scatter plots of the requirements and software similarity

(> 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. Figure 3 shows the distribution of software similarity among the top 50 similar pairs of requirements based on each language model. As it can be seen, in all cases, in at-least 60 percent of the pairs, the software similarity stays above 80 percent (in class C).

In addition, Figure 4 presents a holistic view of the association between the requirements similarity and software similarity. The requirements similarity (on X-Axis) is calculated using different language models. The software similarity is plotted on Y-Axis and is calculated using our JPLag-based pipeline, shown in Figure 2. 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 positive association between the two variables. Besides, we also visualize the interquartile range (IQR), mean, and outliers in our variables in Figure 5. As can be seen from Figure 5, the software similarity for most requirement pairs stays above 70%.

Correlation Analysis. We applied correlation analysis to quantify the relationship and find the most suitable approach toward requirements similarity

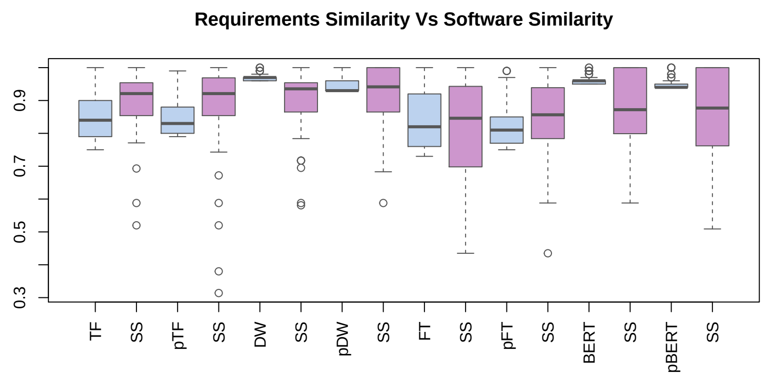

Fig. 5: Requirements Similarity (blue) and their corresponding Software Similarity (SS, purple) for all pipelines

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

	TF	**pTF**	DW	pDW	FT	pFT	BERT	pBERT
rho	**0.5089**	**0.5927**	0.2642	0.3104	**0.5718**	0.4676	0.3865	**0.5575**
p-value	0.0001	5.753e-06	0.0636	0.0282	1.439e-05	0.0006	0.0055	2.594e-05

computation. We measure the correlation between the similarity of the top 50 most similar pairs of the requirements and their source code similarity. We choose the top 50 pairs because it is a suitable number for a sample size (for applying statistical tests) and, at the same time, not a large number of pairs compared to the total requirements.

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 positive association between the requirements similarity and software similarity for all the language models.

5 Discussion

From the results shown in Figure 3, it can be seen that even in worst cases, the 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. Based on the descriptive statistics, we can make the following conclusion.

Requirement-level similarity can be used as proxy for retrieving relevant software (sharing at-least 80% software similarity) for reuse in at-least 60% of the cases.

In addition, the trendlines in Figure 4 also shows that the results from all the language models could have a positive association with software similarity. However, in some cases these language models can produce inaccurate results. As it can be seen in Figure 5, there are some outliers in the retrieved software. Besides it can also be seen from Figure 5, the variance in the software similarity across the pairs is high in case of FT and BERT, suggesting that these models tend to capture more nuanced semantic similarities in requirements, which may point to more fine-grained variations of the software. 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 representation 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. Figure 5 also shows that similarity ranges largely vary between language models (e.g., BERT and DW have 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.

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 moderate positive correlation between the requirements similarity computed with tf-idf, FastText and BERT (shown in bold text in Table 2). Results also show that pre-processing improves the correlation for all language models except FastText.

Our results indicate that term-frequency inverse document frequency (tf-idf)-based language model with pre-processing shows a moderately positive correlation (with rho of 5.92) to software similarity.

Surprisingly, the decades-old *tf-idf* performs better than the new state-of-the-art language models. This can be explained by the limited vocabulary and high similarity of terms used in the requirements of the two projects, as typical in the RE domain [16], where synonyms are not recommended, and company practices encourage uniform terminology. 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, could be more evident. The worst performance is obtained with Doc2Vec. 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.

It is worth remarking that our final objective is to improve requirements-based software reuse. At this stage, our results can be used to build recommender systems that use requirements similarity to retrieve candidate software for reuse. The actual in-field assessment of the quality of the retrieval—or, in other terms, the answer to the question: *does the retrieved software satisfy my requirement?*—will need to be addressed with the involvement of human operators.

6 Threats to Validity

In this section, we present validity threats according to Runeson et al. [33].

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 3.4) to represent the semantics of the requirements. We did not consider similarity as assessed by human subjects, as our goal is to use language models for automatic similarity computation. However, different results may emerge if human subjects are involved in the assessment.

To mitigate potential internal validity threats, we followed standard procedure and open source implementations. In addition, we also involved researchers from diverse backgrounds to validate the study design and execution. Finally, we also involved a technical project manager at the company in validating our data collection procedure.

Our results are based on data provided by one company using a data set of two projects developed by a team. 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 [37], these results might be applicable to similar contexts, where similar RE practices are followed. Further studies are needed on other abstraction levels of requirements and in different companies and domains to generalize the results.

Finally, we address the threats to the reliability of our results by providing enough details on the experimental setup and implementation. In addition, we also provide the R script and the similarity values between the pairs for replication purposes[11].

7 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

[11] Replication package, https://doi.org/10.5281/zenodo.4275388

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 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. Our analysis shows that the correlation between requirements and code similarity is moderately positive, even in the best case. So, a relationship exists between the two, but there is also a need for further research on language models and similarity measurement approaches that 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 include: (1) considering the original tender requirements, and identify the relationship with existing requirements and associated software, to support early evaluation during bid proposal (2) considering feature or refactoring requests as input queries, to support change impact analysis [4, 7] (3) consider other companies and domains other than railways to increase external validity of the results (4) involve domain experts in the assessment of similarity measurements, as well as in the empirical evaluation of requirements-based software retrieval for reuse (5) 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 [17, 18], as well as potentially tacit requirements that are implemented in the software but are not specified.

References

1. 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)
2. Abbas, M., Saadatmand, M., Enoiu, E., Sundamark, D., Lindskog, C.: Automated reuse recommendation of product line assets based on natural language requirements. In: Ben Sassi, S., Ducasse, S., Mili, H. (eds.) Reuse in Emerging Software Engineering Practices. pp. 173–189. Springer International Publishing, Cham (2020)
3. 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)

4. 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)
5. 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)
6. 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). https://doi.org/10.1007/s10664-013-9255-y
7. 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)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. 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)
14. Felfernig, A., Falkner, A., Atas, M., Franch, X., Palomares, C.: OpenReq: Recommender systems in requirements engineering. In: RS-BDA. pp. 1–4 (2017)
15. 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)
16. 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). https://doi.org/10.1109/RE.2017.29
17. Gervasi, V., Zowghi, D.: Supporting traceability through affinity mining. In: International Requirements Engineering Conference (RE). pp. 143–152. IEEE (2014)
18. 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)
19. 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)
20. Irshad, M., Petersen, K., Poulding, S.: A systematic literature review of software requirements reuse approaches. IST Journal **93**, 223–245 (2018)

21. 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)
22. Krueger, C.: Easing the transition to software mass customization. In: International Workshop on Software Product-Family Engineering. pp. 282–293. Springer (2001)
23. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International conference on machine learning. pp. 1188–1196 (2014)
24. 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)
25. Manning, C.D., Schütze, H., Raghavan, P.: Introduction to information retrieval. Cambridge university press (2008)
26. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
27. 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)
28. 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)
29. Pohl, K., Böckle, G., van Der Linden, F.J.: Software product line engineering: foundations, principles and techniques. Springer Science & Business Media (2005)
30. Prechelt, L., Malpohl, G., Philippsen, M., et al.: Finding plagiarisms among a set of programs with jplag. J. UCS **8**(11), 1016 (2002)
31. 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 (May 2010)
32. Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T. (eds.): Recommendation Systems in Software Engineering. Springer (2014)
33. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)
34. Shatnawi, A., Seriai, A., Sahraoui, H., Ziadi, T., Seriai, A.: Reside: Reusable service identification from software families. JSS **170**, 110748 (2020)
35. 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)
36. 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)
37. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Science of computer programming **101**, 136–152 (2015)
38. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing (nlp) for requirements engineering: A systematic mapping study. arXiv preprint arXiv:2004.01099 (2020)
39. 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)