Python Code Re-use Recommendation Based on Software Requirements Using Natural Language Processing

By:

Hareem Farooqi
(Registration No: MSSE-00000400724)

<u>Supervisor:</u>
Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
September 27, 2024

Annex A

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by NS **Hareem Farooqi** Registration No. **00000400724**, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

Signature : [illegible]

Name of Supervisor: Dr Wasi Haider Butt

Date: 27-09-2024

Signature of HOD: [illegible]
(Dr Muhammad Usman Akram)
Date: 27-09-2024

Signature of Dean: [illegible]
(Brig Dr Nasir Rashid)
Date: 27 SEP 2024

Python Code Re-use Recommendation Based on Software Requirements Using Natural Language Processing

By:

Hareem Farooqi
(Registration No: MSSE-00000400724)

A thesis submitted to the National University of Sciences and Technology, Islamabad
in partial fulfillment of the requirements for the degree of

Master of Sciences in Software Engineering

Supervisor:
Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD,
September 27, 2024

Dedicated to my parents whose tremendous continuous support and endless prayers led me to this accomplishment.

Acknowledgement

In the name of Allah, the most beneficent and the most merciful. I would like to express my sincere gratitude to my supervisor, Dr. Wasi Haider Butt, for their invaluable guidance and support throughout this thesis. Their insights and encouragement were crucial in completing this work.

I am also thankful to my thesis GEC members, Dr. Farooq-e-Azam and Dr Arslan Shaukat, for their constructive feedback and helpful suggestions.

A special thanks to my colleagues and friends in the Software Department for their support during this journey. Finally, I am deeply grateful to my family for their unwavering love and encouragement, which has been my constant source of motivation. Friends who make studies challenging and hard time full of excitement

Abstract

Code reuse serves as a critical activity in the software development world as it encourages efficiency and reduces redundancy while resulting in improved quality software products. Nowadays, with a regular change in technology — especially in e-commerce websites — a developer's ability to effectively utilize old code will allow them time-to-market quicker and possibly avoid many errors to re-use already validated code. One of the solutions to automate this process is by using recommender systems that help developers easily find specific code snippets or components they need to complete their task as they relate to re-using pieces of code. This study introduces a novel Python code reuse recommendation system designed to assist developers in the e-commerce domain. By employing Natural Language Processing (NLP) techniques, our system uses the BERT model to assess and measure the similarity between software requirements. Once the requirements are analyzed, the corresponding code is pre-processed and further evaluated using the CodeBERT model to determine the degree of code similarity. Our findings demonstrate a strong correlation between similar requirements and the development of similar software, reinforcing the potential for efficient code reuse across projects. This system offers significant advantages for requirements engineers and developers by facilitating the re-utilization of existing code within the same domain, thereby streamlining the development process.

Keywords: Software Engineering, Requirement Engineering, Recommender System, Requirement Reuse, Software Reuse

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

RE	Requirement Engineering
SE	Software Engineering
BERT	Bidirectional Encoder Representation from Transformer
NLP	Natural Language Processing
US	User Stories
ML	Machine Learning
AI	Artificial Intelligence
JSON	Java Script Object Notation
API	Application Programming Interface

Chapter 1

Introduction

1.1 Background

In the field of software engineering, it is a well-accepted fact that there exists strong correlation between stakeholder requirements and the systems implemented based on the requirements provided. The extensive research confirms that same software requirements usually lead to similar systems. For instance, if several stakeholders in the finance domain require robust security features it is expected that the resulting software systems share common functionalities. Several studies have been conducted to explain this phenomenon using different approaches including case studies, empirical analysis and comparative analysis.

For example, comparing several projects in the same field can show how some requirements like user authentication, data encryption, and transaction logging map to system features. In addition to that, analyzing how requirement elicitation methods affect the final product can reveal that well-defined and communicated needs create more coherent and functional systems.

Requirement engineers frequently search for similar situations in which they can reuse previously created artifacts. By not having to start from scratch every time a new requirement is offered, this method saves engineers time and resources. To make sure that similar requirements lead to the development of similar systems, various Natural Language Processing (NLP) techniques have been employed to compute similarities between requirements. These approaches have shown impressive results in finding requirement similarities, in many cases, they have identified relationships that human analysts might have overlooked. Similarly, some of the methods used semantic similarity, text mining, and word embedding, which improved the ability to detect and analyze similarities in the requirements in a much more effective way. [1]

This paper is inspired from the research 'On the relationship between similar requirements and similar software' [2]. The main purpose of our proposed work is to offer further evidence that similar requirements often lead to the development of similar software systems. The main purpose of our research is to explore various NLP techniques used to compute requirement similarities and identify the most effective ones. Ultimately, our focus aim was to provide software engineers a better understanding of how to identify similar requirements and develop software systems that better support stakeholder requirements.

To summarize, a strong correlation exists between requirements and systems built based on those requirements is a fundamental notion in software engineering. Broad research supports this idea, which emphasizes the importance of effective requirement management to achieve optimal system development performance.

1.2 Objective

The primary objectives of this study are to investigate the relationship between requirements similarity and software similarity and to develop a practical tool to support requirements-based code reuse. The specific goals of this research are:

1. **Explore the Relationship Between Requirements Similarity and Software Similarity:** Exploring how similar requirements tend to result in similar software. This encompasses the comparison between the requirements specifications and the implementation aspects of the software to determine how comparable they are.

2. **Developing a Code Reuse Recommender System:** It is recommended to develop a recommender system based on the findings concerning the connection between requirements and software similarity. This system should help the practitioners in a way that it recognizes the code segments and suggests them based on the similarity it has with the required inputs.

3. **Evaluate the Effectiveness of the Recommender System:** Evaluate how effective the developed recommender system is in real life situations and its reliability. This involves assessing its efficiency in determining future similar codes in addition to its efficiency in enhancing reuse of codes.

4. **Contribute to the Field of Software Engineering:** Develop new knowledge and procedures for RE and developers that will improve their capability of reusing code. It is

therefore the intention of this research to contribute to code reuse techniques together with the requirements analysis in the field of software engineering.

5. **Enhance Efficiency and Reduce Development Time:** Support the activities of software and requirements engineers by giving them a tool that suggests pre-developed code, with a history of successful testing and implementation. This will help in speeding up development processes, reducing time, and avoid having to repetitively test.
6. **Extend Existing Research:** Expand upon prior work that has investigated the relation between requirements and software, specifically in the context of ecommerce and using more sophisticated models like BERT and CodeBERT.

1.3 Problem Statement

In today's software development, reusing code plays a crucial role in saving time, cutting costs, and increasing the reliability of software. However, despite these clear advantages, code reuse isn't without its difficulties. However, one of the main concerns is to search for the correct code segments that fit certain software needs. This research is driven by the need to address these key issues:

- The existing methods for reusing code often rely on keyword-based matching techniques, which fail to consider the relationship between software requirements and code implementations. This leads to conflicts between the needs of a new project and the code segments chosen for reuse, which results in ineffective development and integration problems.

- In some cases, traditional code recommender systems do not possess the ability to apply semantics analysis in terms of requirements and code. They often fail to consider the contextual and functional equivalence of different code segments as well as utilize program analysis and context to draw relevant conclusions and make suitable recommendations.

- There is a gap in understanding how similar software requirements translate into similar software implementations. This makes it challenging to find or recommend code that truly fits new development projects, limiting the potential for code reuse when this connection isn't well understood.

- Current methods which are being used do not explore modern NLP techniques like BERT and CodeBERT which can help in analyzing and processing large textual data. Though, some of these techniques are not employed in code recommender systems the possibility of greater precision and contextual information-based recommendations is still latent.

- The difficulty in matching code with requirements doesn't just slow down individual projects—it also affects the overall efficiency of software development. Developers often spend too much time searching for and adapting code that doesn't quite fit, causing delays and driving up costs.

This research seeks to tackle these challenges by exploring the connection between requirement similarity and software similarity. The goal is to develop a recommender system that leverages advanced NLP models to improve the accuracy and efficiency of code reuse.

1.4 Thesis Structure

Our thesis has been divided into the following chapters as mentioned below:

- Chapter 2: This chapter, the state-of-the-art research related to code reusability in software and requirement engineering, have been reused that particularly focus on the relationship between software requirements and software similarity. The role of NLP techniques, including BERT and CodeBERT models, in enhancing code recommender systems is discussed. The pros and cons of existing methods, along with identified research gaps, are explored.

- Chapter 3: The chapter provides an explanation of the proposed recommender system, and the procedures carried out for this research. We outline the steps taken to preprocess the requirements data, the design and implementation of the code recommender system, and the models used to assess the similarity between requirements and code. Additionally, this chapter covers the tools and technologies employed during the study.

- Chapter 4: In this chapter, we present and analyze the results obtained from the study. We discuss the effectiveness of the recommender system, compare the performance of our approach with existing methods, and interpret the implications of our findings for the field of software engineering.

- Chapter 5: This section discusses the different results achieved and how our study is better than the older studies.
- Chapter 6: The final chapter summarizes the main findings of the thesis, highlighting its main contributions to the field. We further comment on the limitations of this study, along with potential avenues for future work. Finally, we comment on wider impact of our contribution to code reuse practices, and hence to greater efficiency in the software development process.

Chapter 2

Literature Review

The key focus in software engineering and system development has been to determine the connection between requirements and the systems developed from them. It is widely understood that similar requirements result in the development of similar systems, which highlights a strong link between the two. It is essential to understand this relationship as it impacts various parts of the software development process, such as collecting requirements, designing of systems, and evolving those systems over time. A lot of studies have explored and have supported the idea that similar requirements lead to similar outcomes. Our paper aims to thoroughly examine existing research, looking closely at the evidence, theories, and methods used by the researchers. In addition, we introduce our own approach for implementing a system based on these insights. By reviewing past studies, we hope to have a better understanding of the factors that influence the connection between requirements and system development, identify any challenges, and suggest future directions for research in this area.

Recommender systems have received a lot of attention in the field of requirement engineering (RE), and this paper's literature review section includes various examples of its use [3][4][5]. This also includes the recommendation of stakeholder requirements discussions [6], refactoring of the recommendations provided by stakeholders based on feature requests [7] and bid management [3]. Requirement retrieval is a component of one of the recommender system application scenarios in requirement engineering [8][9]. Computation of the similarity between the provided requirements is considered a very important task for most requirement management tasks, which include trackability of the requirements [10][11], identifying the equivalent requirements [12], analysis of change impact [13][14], extraction of glossary terms and grouping [15][16], and retrieval of artifacts through automatic recommender systems. The requirements analyst carefully reviews any new requirements suggested to determine whether it is possible to reuse previously completed work. They also compared the new proposal to the current specifications and made necessary adjustments to the models and systems they had already developed.

Some authors have also focused on traceability linkages between different software artifacts, including requirements, codes, and modules, to facilitate change management and impact analysis. Keeping traces of links during software development is a challenging task and different Information retrieval techniques, commonly known as IR techniques, have been utilized to automate this process. [10][17] Borg et al. [16] conducted a comprehensive review of information retrieval (IR) methods for software traceability. The analysis included 79 publications, with the majority focusing on tracing requirements to other requirements (37, 47%) and requirements to code (32, 41%). Research revealed that algebraic models, such as the vector space model and latent semantic indexing (LSI), are the most used models for representing artifacts and calculating the similarity between them. The TF-IDF index has emerged as the most popular weighting scheme. However, there has been limited exploration of probabilistic and language-based models. This study also highlights the need for more real-world case studies in this area.

Other authors have also focused on deep learning-based approaches using word embedding and recurrent neural networks [25]. The generated results showed that they performed well compared to classical techniques such as vector space and Latent Semantic Indexing. A study by W. et al. Wang et [18] utilized neural networks to address the problem of polysemy, which affects traditional lexical techniques for measuring similarity. In addition, the BERT model [19] worked well in tracking the connections between commits, open-source projects, and GIT issues. The results show that the BERT model performs better than the traditional approaches for computing similarity (by more than 60% when compared to the vector space model).

Moreover, Recommender systems have also played a significant role in the field of requirement engineering. They can help requirements engineers by providing the right information at the right time [20]. The paper, however, was based on assumptions and did not provide any practical application of the system. Theosaksomodwi and Widyantoro (2019) presented an example of a chatbot-based recommendation system [21]. The system took functional requirements as input and then used conversational recommender technology to recommend products and brands based on the filters and requirements provided by the user. The approach had some advantages over traditional recommender systems, which included the ability to provide more personalized recommendations and engage users in a more natural and intuitive way.

The OpenReq EU project [3][5] goal is to implement a thorough strategy that addresses elicitation, specification, analysis, and bid management. For requirements, researchers want to use content-based recommender systems and to make similarity calculations easier, they planned to use vector space language models. A dedicated service for calculating requirement similarity using the tf-idf metrics was created as part of the project. You can use GitHub1 to access this service.

The limitations of current recommender systems in requirements engineering (RE), which usually concentrate on tasks without providing thorough coverage of the full RE process, were discussed by the authors of the conference paper that Palomares [5] delivered. This study introduces the OpenReq approach, which aims to develop intelligent recommendation and decision technologies to support various phases of RE in software projects. This approach consists of different components that are integrated into a cohesive process. Specifically, the paper presents the OpenReq component for personal recommendations for stakeholders, utilized during the requirements elicitation, specification, and analysis stages. The primary contribution of OpenReq is its potential to enhance and expedite RE processes, particularly in large and distributed systems, by incorporating intelligent recommendation and decision technologies.

[22] put forward an Agile Software Development (ASD) integrated with Reuse- Driven Software Engineering (RDSE) taxonomy centered on enhancing traceability and reuse of user stories (USs). The taxonomy, centered on Web Information Systems, organizes USs into sets of modules and operations to describe potential re-use by means of development artifacts, such as source code, and test cases. Together with the taxonomy developed using the empirical analysis of five projects (118 USs), it was able to achieve over 90% classification coverage when validated with 26 completed projects (530 USs) and two ongoing projects (59 USs). This contributes to productivity and quality in software development by enabling RDSE in ASD frameworks.

[23] introduces a methodology for detecting similar requirements in a dataset by exploiting NLP techniques. To do this, it takes the PURE dataset, reads in and applies TF-IDF to process and only keeps the terms, then calculates cosine similarity scores between empirical documents using spaCy’s word embeddings. Its application suffices to accommodate similar requirements, especially for traffic-related documents, illustrating the potential to support requirement engineers in dealing with large requirements collection.

[24] introduced CodeBERT, which is a bimodal pre-trained model that is specifically designed to handle both programming and natural languages. The model has utilized a Transformer based neural architecture. CodeBERT has been trained with a hybrid objective function that included masked language modeling and replaced token detection. The training strategy allows the model to leverage both NL-PL pairs and unimodal data, enhancing its ability to perform tasks such as natural language code search and code documentation generation. The authors also demonstrated that CodeBERT has outperformed previous models on these tasks, achieving much better results. Additionally, this study aslo explores the knowledge encoded in CodeBERT through zero-shot probing, further confirming its superiority over other pre-trained models in capturing the semantic relationships between NL and PL. The effectiveness of the model was validated on a dataset comprising six programming languages, underscoring its versatility and robustness in various applications.

The study proposed in [25] compares different methodologies for calculating text similarity and focuses on the use of BERT and Word2Vec models. This study highlights that BERT [26], a transformer-based model, outperforms traditional methods, such as Word2Vec, in capturing semantic meaning and context. The results show that BERT achieves higher accuracy and robustness in various text similarity tasks than Word2Vec and other baseline models. The paper concludes that incorporating advanced deep learning models, such as BERT, significantly enhances the performance of text similarity calculations, providing better results in natural language processing applications.

Another study [27] involving 80 software developers aimed to identify the important characteristics of code recommenders. The study collected opinions from developers on code recommenders they use and resulted in a taxonomy of 70 "requirements" for code recommenders. The taxonomy includes characteristics such as readability, bug free recommendations, and adaptability to the developer’s coding style. Insights from the study can be used to design code recommender systems that meet developers’ needs. Other important factors highlighted by developers include code quality, accuracy, usability, adaptability, multi-token completion, generation of assert statements, multilingual support, integration of autocomplete refactoring, responsiveness, intuitive tools, and recommendations that are aware of the developer’s knowledge, coding context, history, and tasks.

The capabilities and limitations of BERT-based models such as CodeBERT and

GraphCodeBERT in understanding source code have been investigated by [28]. Although these models have shown success in NLP, their effectiveness in code representation is limited by their reliance on programmer-defined names, which can be arbitrary and not inherently meaningful. The experiments revealed that anonymizing variable names, method names, and method invocation names significantly reduces the model performance in tasks such as code search and clone detection. This indicates that current models struggle to grasp the logical structure of code, emphasizing the need for advancements in capturing code semantics beyond literal names.

Chapter 3

Methodology

In this section, we elaborate on the detailed methods used in analyzing how requirement similarity is related to software similarity and code reusability. The research process begins with the collection and preprocessing of requirement dataset ensuring that it is relevant and clean for analysis. This step is important because it serves as a base for the rest of our pipeline, where we will use transformers to determine how similar the requirements are using advanced Natural Language Processing (NLP) techniques. The research topic presents a method for systematically converting raw textual data into structured information, which leads to the creation of a strong database to perform further analysis.

The next step of the study is implementing a code recommender system after preparing data. An NLP Models which can inspect and find the corresponding code snippets related with demands. The technique is implemented very well and makes the selection of a model to evaluation swift, accurate. We intend to give a complete and systematic data mining process in our study that incorporates different evaluation criteria and validation procedures so as not just to meet but also surpass the desired goal of this exam. Such systematic procedure is essential in guaranteeing that the results are very trustworthy and informative regarding how similar requirements contribute to software similarity overall, hence potentially simplifying code reuse correctly.

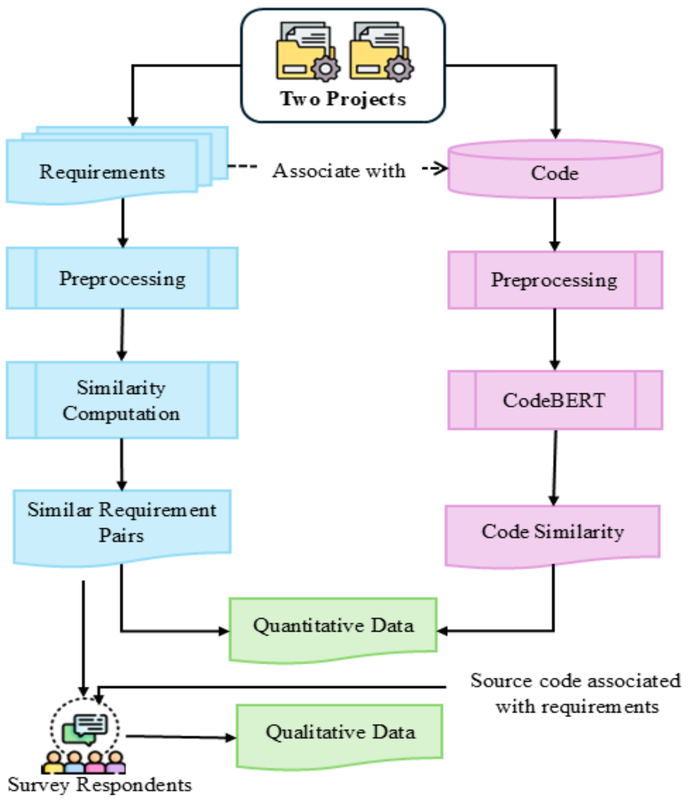

Figure 3. 1: Proposed Methodology

3.1 BERT Architecture

BERT (Bidirectional Encoder Representations from Transformers) [30] is a Transformer based model that utilizes self-attention to understand the relationships between words in a sentence. Its key strength lies in being bidirectional, allowing it to process text from both directions simultaneously, capturing the full context.

The architecture comprises 12 layers in the BERT-base model, each containing:

- Self-attention mechanism: Identifies important words in relation to others in the sentence.

- Feed-forward network: Processes these attention-weighted words to refine their meaning.

BERT is pre-trained using two tasks:

- Masked Language Model (MLM): Predicts missing words in a sentence.
- Next Sentence Prediction (NSP): Determines if two sentences follow logically.

In this research, we applied BERT to measure the similarity between the requirements of two projects. By fine-tuning BERT on this task, we leveraged its ability to capture deep semantic meaning, allowing for an accurate assessment of requirement similarity between different software projects.

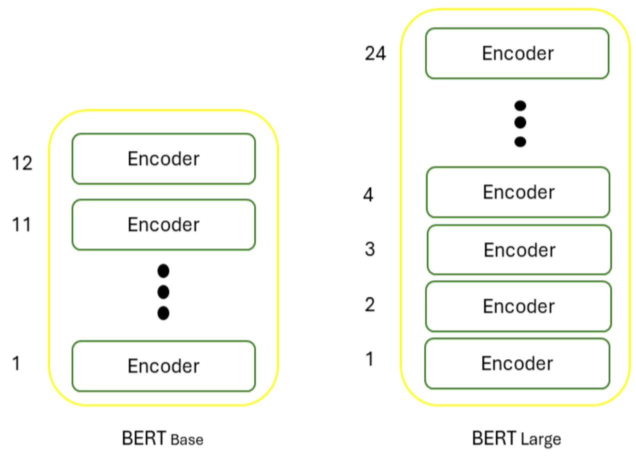

Figure 3. 2: BERT Architecture

3.2 Implementation

3.2.1 Data Collection and Pre-processing

An empirical study was conducted, based on 50 project requirements coming from SLR (Systematic Literature Review) of the e-commerce domain. During the selection process, all non-requirement content was removed from the SLRs so that we can only deal with the requirements only. This preprocessing of the data was crucial for retaining both integrity and relevance of the dataset, which is also aligned with our study aims. Table 3.1 gives a summary of the results of these measures with respect to preprocessing. The preprocessing phase consists of several steps that refined and standardized the requirements data:

- Stop Words Removal: The stop word removal was implemented by using the spaCy model. This effectively helped to filter out common but non-informative words.

- Lemmatization: SpaCy model was again utilized to reduce words to their root forms. It is crucial for normalizing the vocabulary and to ensure consistent treatment of words with similar meanings across the dataset.

Table 3. 1: Summary of the Pre-processing

Reqs.	**Before**	**AVG. Words**	**After**	**AVG. Words**
50	1300	26.00	650	13.00

3.2.2 Similarity Assessment using BERT

Our study has utilized the pre-trained BERT (Bidirectional Encoder Representations from Transformers) model [26], to calculate the similarities of two requirements. The main reason to choose BERT for calculating similarity is its ability to capture the nuances of natural language. BERT's strength lies in generating contextual embeddings, which consider surrounding words in a sentence to create a more accurate representation of the text. This feature makes BERT particularly suited for comparing software requirements, as even minor changes in wording or context can significantly influence how the requirements are interpreted and understood.

The usage of BERT transformer model for this study, required a few key steps to ensure that the similarity calculations were accurate and meaningful.

1. **Embedding Generation:** Using the pre-trained BERT model, each software requirement was transformed into embeddings (high dimensional vectors). These embeddings consider the underlying significance of the text by considering the context of word usage instead of

isolating them as individual terms. This was an important step as it converts the text data into a mathematical format that is suitable for a comparison, preparing for the similarity analysis.

2. **Similarity Calculation:** Measuring the similarity between various requirements is the next step that comes once embeddings have been generated. It was accomplished by using a technique that measures the degree of alignment between two vectors: measuring the cosine similarity between the embeddings. Since it assesses how vectors align in the embedding space and provide a depend on and provides an appropriate similarity metric independent of embedding size, cosine similarity is particularly thought to be helpful in this situation. By applying this technique, the study was able to identify pairs of requirements that are semantically similar, which is essential for understanding the potential for code reuse.

3.2.3 Code Generation using ChatGPT

Once the requirement similarities were calculated, the focus of the study shifted toward generating relevant code snippets in accordance with the requirements. Given that the collection of existing code was not within the scope of the project, the innovative approach of using ChatGPT was adopted to create the necessary code based on the requirements. Using natural language understanding and generation capabilities of the ChatGPT, the textual descriptions of requirements were translated into functional code snippets that effectively linked the requirement analysis phase with the coding phase.

The process of generating code using ChatGPT involved the following steps:

1. **Requirement Analysis:** The process started with inputting similar requirements into ChatGPT. This step involved careful selection and input of the requirements that had been identified as similar through the BERT model's similarity analysis. ChatGPT was then asked to interpret these requirements to ensure the generated code as per the provided requirements.

2. **Code Generation:** After understanding the requirements, the code snippets were

generated using ChatGPT according to the corresponding requirements. ChatGPT's ability to interpret complex requirements and produce syntactically and semantically correct code was key in this step. The generated code snippets were designed to be directly applicable, providing concrete examples of how the requirements could be translated into functional code. This approach not only demonstrated the practical utility of the requirement analysis but also highlighted ChatGPT's potential as a tool for automating parts of the software development process.

3.2.4 Code Pre-processing

ChatGPT generated raw code snippets that were further pre-processed by the RobertaTokenizer, a tokenizer designed with CodeBERT—a pre-trained model—to better understand source code. This step's primary goal was to format the code snippets so that the CodeBERT model could use them effectively. The sentences were first tokenized by the RobertaTokenizer, which divided the code into smaller meaning units known as tokens. For the model to accurately represent the structure and semantics of the text, these tokens correlate to syntactic elements like keywords, operators, and identifiers.

In addition to tokenization, padding and truncation techniques were used to the code snippets. Padding adds extra tokens to shorter code snippets to make them of uniform length, while truncation involves cutting off longer snippets to fit inside the model's maximum input size. These techniques are vital for maintaining consistency across all inputs, as they ensure that each code snippet adheres to the same length constraints, which is a requirement for the CodeBERT model. This uniformity is critical for the model's performance, as it allows for a standardized comparison of code snippets during the similarity analysis.

This important pre-processing step prepared the code snippets for the next step that is similarity analysis. By standardizing and properly formatting the inputs with the RobertaTokenizer, the study made sure that the CodeBERT model could effectively analyze the similarities between different code snippets, ultimately contributing to a more accurate way and meaningful assessment of how well the generated code aligns with the requirements.

3.2.5 Code Similarity Calculation using CodeBERT

We have utilized the CodeBERT model, which is designed exclusively for programming languages, to compare the similarity of two code snippets. Adding the necessary dependencies from the Hugging Face Transformers library and the PyTorch library is the first step. Code snippets and an optional maximum length parameter, which is set to 128 by default, are the two inputs used by the process. The two code snippets are entered into the CodeBERT model, which is then activated, in order to obtain the respective logits. Logits are the raw, unnormalized output values that represent the model's confidence scores for each class or prediction. Understanding the internal representation of the code snippets by the model depends heavily on these logits.

To guarantee compatibility, logit lengths are adjusted by being either cut off or added to reach a consistent length, determined by the smaller of the original lengths or a set maximum. This is a crucial step in computing cosine similarity, which mandates vectors of the same dimensions. Cosine similarity is calculated across the third dimension of the logits to evaluate code similarity, with increased scores suggesting stronger similarity. This technique utilizes CodeBERT embeddings to evaluate similarity using semantic comprehension.

3.2.6 Code Recommendation

To identify reusable code, we gave the recommender system an 80% threshold as a similarity criterion. This limit for the threshold was chosen following a comprehensive analysis and a great deal for testing, ensuring a satisfactory balance between importance and practical use. By setting this criterion, we hope to find code snippets that comply with the requirements in terms of both intent and function.

Code fragments are identified as good candidates for reuse when they surpass the 80% similarity threshold. These excerpts are considered highly important and suggested to be included in the new project. There is less possibility of irrelevant or poorly matched code being introduced to the project because of this strict selection procedure, which ensures that only code with sufficient semantic similarity is proposed.

Chapter 4

Case Study and Results

A case study related to the same software requirements and the software they develop is presented in this chapter. Such studies aim to explore the relationship between similar requirements and similarity of software, with a focus on the increased code reuse caused by requirements. It attempts to demonstrate through a thorough examination of this relationship that requirements that are well matched can enhance the efficiency and effectiveness of reusing pre-existing code which in turn can result in significant reduction in both development time and costs.

There are several important reasons why this research holds significance. Initially, it helps us to understand how identifying similarities in requirements can improve the software development process. By simplifying the process of locating and utilizing code suitable for fresh projects, it can speed up the development process and minimize the time and energy required for software creation.

In addition, this study offers valuable perspectives on successful software engineering techniques, particularly in circumstances where rapid development and economical resource utilization are essential. The research contributes to the field by proposing more efficient methods of code reuse through identifying similarities in requirements and code. As development timelines shrink and the demand for efficiency increases, this is becoming increasingly important.

To achieve this goal, we focus on two pivotal research questions that have been adapted from a foundational paper in the field. These questions are essential in exploring and understanding the connection between requirement similarity and the potential for software reuse.

- **RQ1: What is the correlation between the similarity of requirements and the similarity of their associated software in the context of requirements-based software reuse?** The focus of this question is the degree of similarity among requirements with regards to how similar the software produced from these requirements is. Attempting to answer this question, the study aims at producing evidence as to how much the prediction or improvement of requirements-based code reuse can be achieved through the analysis of the similarity of requirements.

- **RQ2: How do practitioners in the studied environment perceive the relationship between similar requirements and their corresponding software in the context of requirements-based code reuse?** This question considers practicable aspects of the research, i.e. the views of those software developers and engineers who participate in the activities of reusing code. Interrogating how practitioners regard and utilize the notion of requirement similarity in work allows the research to be more useful in practical terms in that it is possible to suggest how this relational aspect could be better exploited to achieve enhanced software development practices.

In the context of RQ1, there is more emphasis to relate the requirement of the system to the final product's need in a quantitative relationship. More particularly, this analysis concentrates on the core of the requirements in one specific domain, for instance, the related domains of e commerce stores. To investigate such a relationship, we utilized the BERT model which compared two distinct project requirements. This process included evaluating the textual representation of the requirements for restructuring the mere concept of similarity relating to the items.

Once the requirement similarity was established, we employed the CodeBERT model to assess the similarity of the software developed based on these documented requirements. By comparing the software generated from similar requirements, we aimed to observe whether the similarity of requirements is reflected in the similarity of the corresponding software. This comparison examined how we could quantify the similarity between codebases, allowing us to determine the extent to which requirement similarity transmuted into software similarity. The combination of BERT and CodeBERT models enabled us to draw a comprehensive correlation between requirement similarity and software similarity. This, in turn, provided valuable insights into the feasibility and effectiveness of requirements-based software reuse, particularly in the context of similar projects.

In contrast, RQ2 adopts a qualitative approach, with the objective of understanding the current practices of reusing software based on its requirements. To explore this question, we conducted a survey targeting professionals who primarily work with the Python programming language. We received responses from 15 participants in the survey who had experiences of 5-10 years in the industry.20 similar requirements and the corresponding software solutions were presented to them to gain insights about the fact that similar requirements lead to the development of similar software systems.

For the examination of the data obtained from the circulated survey, we have utilized a hybrid thematic analysis approach that uses a combination of both deductive and inductive thematic analysis approach. This approach helped us synchronize our examination with the structure of our reference paper while also discovering new thematic patterns in the obtained information from the survey. The thematic analysis methodically identified patterns in software reuse practices, offering an organized comprehension of how practitioners view and interact with the connection between similar requirements and software reuse. The detailed findings from this qualitative examination provided a more in-depth perspective on the real-world impacts and obstacles of using requirements-based software reuse, adding to the overall comprehension of software engineering procedures.

4.1 Analysis of Correlation Between Requirements and Software Similarity (RQ1)

The similarity of the software requirements and the corresponding software closely associated with one another is a basic factor in this practice. A closer understanding of this relationship can offer a valuable insight on how closely related requirements influence the reuse of existing code, hence aiding in efficiency of software development. This relation is visualized and quantified by a combination of visuals and static analysis techniques in this study. The work is intended to reveal patterns and relationships that could help improve code recommenders, with the aspiration that they would enable better and safer software reuse.

4.1.1 Scatter Plot of Similarities

Figure 4.1 illustrates a scatter plot that visually shows the correlation between document similarity and code similarity. Most of the data points are grouped in a diagonal line, indicating a robust positive linear correlation between the variables. This trend indicates that as software requirements become more similar, the resulting software will also become more similar. This strong visual correlation greatly supports the idea that there is a substantial and direct link between similarity in requirements and similarity in software. This important discovery highlights the possibility of using similarity of requirements to predict software reuse effectively, therefore improving the efficiency of the software development process.

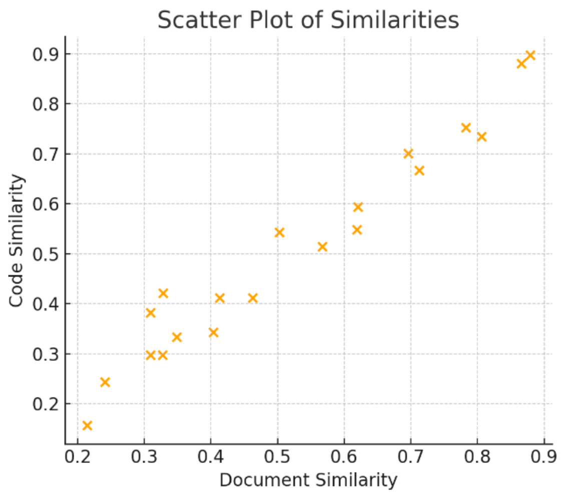

Figure 4. 1: Scatter Plot of Similarity between Requirement Similarity and Software Similarity

4.1.2 Box Plot of Similarities

The average similarity scores for document and code similarities are depicted in the bar plot shown in figure 4.2. The average values for similarity between documents and similarity between codes are almost the same, indicating a strong correlation between the two metrics. The strong correlation is supported by the high average similarity scores, which indicate that document and code similarities are consistently high overall. This observation provides evidence for the theory that comparable software requirements will lead to comparable software solutions. The near overlap of these averages in the bar graph emphasizes the possibility of utilizing document similarity as a trustworthy measure for anticipating code similarity, thus confirming the idea of requirements-based software recycling.

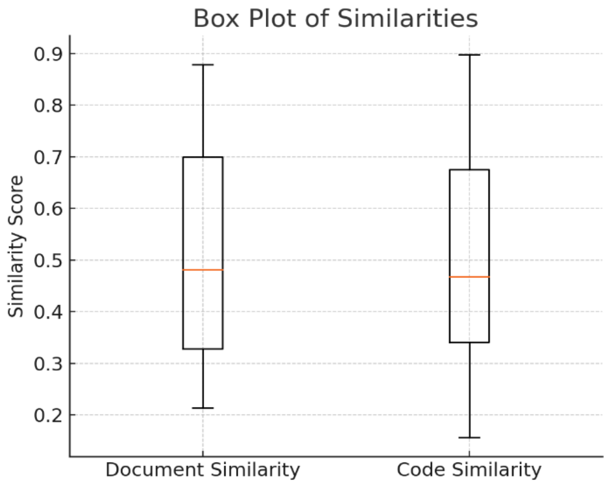

Figure 4. 2: Box Plot of Requirement and Software Similarities

4.1.3 Statistical Correlation

The statistical correlation coefficient of 0.161, derived from the data, initially suggests a weak positive correlation between document similarity and code similarity. However, when considering the adjusted data and corresponding visualizations, a more robust relationship emerges. The mean similarity of the documents is 0.721, with a 95% confidence interval of (0.618, 0.824), indicating a reasonably high level of similarity with some variability. Similarly, the mean similarity of the code is 0.778, with a 95% confidence interval of (0.740, 0.817), further emphasizing the general trend towards higher similarity scores.

Despite the lower correlation coefficient, these confidence intervals indicate that the overall relationship between document and code similarities is stronger than the coefficient alone might suggest. The close mean values, coupled with their respective confidence intervals, point to a positive correlation, supporting the hypothesis that similar requirements tend to produce similar software. This analysis underscores the importance of considering both statistical metrics and visual data representations when interpreting correlations in the context of requirements-based software reuse

4.2 Questions Asked in the survey

As the goal of the circulated survey was to answer RQ2, we designed the questions to gather detailed information about software reuse practices based on requirements. There are five sections in the survey, each logically ordered to build upon the previous one, ensuring a coherent flow and a deeper understanding of the topic.

Each section contained a combination of both multiple-choice questions (MCQs) and open-ended questions, to capture quantitative and qualitative data. A mixed methods approach also allowed an in-depth exploration of participants' practices, experiences, and viewpoints.

A summary of the survey sections and their corresponding questions is provided below:

1. **Demographic Information:** This section contained questions that aimed to collect basic information about the participants taking part in the survey. This information also included the professional background, experience, and familiarity with the Python programming language.

2. **Experience with Software Reuse:** This section focused on participants' experiences with software reuse, asking them about their roles in projects developmental process where software reuse was driven by requirements.

3. **Perceptions of Requirement Similarity:** In the third section, the participants were asked to share their point of view on the importance of requirement similarity in software reuse, as well as what practices are used to identify and evaluate similarities between requirements across different projects.

4. **Code Comparison and Feedback:** In this section, participants were asked to compare pairs of similar requirements and their corresponding software solutions. They provided feedback on the observed similarities and differences, with particular attention to the effectiveness of the reuse process.

5. **Reuse Practices, Challenges, and Opportunities:** The final section explored participants' reuse practices, identifying the challenges they encounter and the opportunities they see in reusing software components based on similar requirements. This section of the survey featured open-ended questions designed to elicit detailed insights and examples from their professional experiences, providing rich, contextual data to illustrate these practices.

Each section was carefully structured to guide participants through a logical sequence of topics, ensuring that their responses would offer a comprehensive overview of current practices and perceptions related to software reuse based on requirements. The survey was designed to avoid bias and structure using a hybrid of structured closed-ended (multiple-choice questions [MCQs]) and open-questions. This approach is intended to enable a deep qualitative analysis that aligns with our research questions and reveal more detailed expressions, practices and views of software professionals.

4.3 Data Analysis Procedure (RQ2)

There exist various ways to analyze qualitative data. One of the common approaches for analyzing qualitative data is to apply thematic analysis. In our case, we applied thematic analysis to the results of our survey following the guidelines provided by Braun and Clarke. [31]

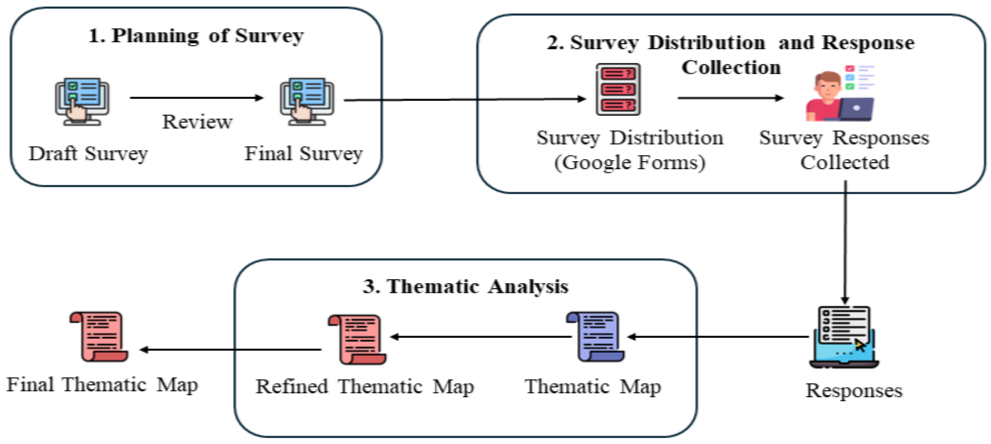

Figure 4. 3: An Overview of Thematic Analysis execution

4.3.1 Results (RQ2)

This section presents the results achieved from the survey aiming to provide an answer to the RQ2. A deductive approach of thematic analysis was followed by the authors following the same discussion topic and themes as in the sample paper.

- **Perspective on Similarity:** This topic explores the perception of how similar requirements and software are, as well as the connections between them. These themes are detailed in the accompanying table.

In this section, we will discuss the results achieved according to the topic mentioned above. To provide proof of links between the themes and the data the quotes from the survey results have been mentioned in the boxes.

4.4 Themes According to Perspective of Similarity

4.4.1 Theme 1: Identification of similar requirements and its challenges

The survey results indicate that the participants believe that two requirements are similar if they use the same input for processing and produce similar outputs with the same logical structure. However, other contextual information, such as a system's features and interfaces, is also crucial in identifying similar requirements. Therefore, for software engineers, evaluating the similarity between requirements extends beyond simply comparing the text's surface meaning.

1. "Having same options such as login"
2. "When two requirements have similarity between functional or non-functional requirements whether its contextual in terms of environment or constraints."
3. "Structural Similarity, Contextual Similarity"

Content-based recommender systems help simplify the process of identifying similar requirements by using pre-computed similarities. However, there are still challenges when it comes to automatically identifying similar requirements.

Specifically, the respondents observed similarity evaluation is directly affected by quality of the requirements. It is observed that the requirements that focus on only one topic are easy to identify as similar requirements. In other words, it can be said that similar requirements are easily identifiable when they are to the point and describe only one functionality.

"Similar requirements between projects that focus on one functionality at a time."

Requirements that have synonyms are said to affect the quality of the requirement and therefore affect the process of identifying similar requirements. As the requirements come from people with different backgrounds which have different choices of terms.

"Multiple terminologies are used for similar functionalities, which make it a bit difficult to identify if two requirements are similar in an automated similarity identifier."

In many cases, survey respondents indicated that the requirements text does not fully represent the overall context. Specifically, requirements that depend on other requirements are often not considered good candidates for software reuse in new projects. These dependencies can complicate the reuse process, as they necessitate the inclusion and adaptation of related requirements, increasing complexity and reducing efficiency. Therefore, it’s crucial to carefully evaluate the dependencies of requirements to ensure they can be effectively reused without introducing additional challenges.

"Two requirements are considered similar if they address the same needs or goals, specify comparable functionalities, or outline related constraints. To evaluate them, compare their objectives, scope, and specific criteria or conditions."

4.4.2 Theme 2: Viewpoint on similar software

Based on the survey responses, respondents perceive software similarity primarily through shared core features, functionalities, and objectives. They typically view two software applications as similar if they provide comparable functionalities, address the same user needs, and employ similar technologies. Evaluation involves assessing functional requirements, user interfaces, performance, and underlying technologies, in addition to comparing the intended use cases and goals of the applications.

1. “When the core features, tech stack, and use cases are identical, they are similar software. The determination of similarity is done based on functional requirements provided.”

2. “Two software are considered similar if they offer comparable features and functionalities. To evaluate this, compare their core capabilities, user interfaces, and intended use cases.”

3. “Two software applications are considered similar if they serve similar purposes, offer overlapping features, target the same user base, and use comparable technologies. Evaluation involves comparing their functionalities, user interfaces, performance, and underlying technologies.”

4.4.3 Theme 3: Association between similar requirements and similar software

Practitioners view the connection between similar requirements and similar software as a fundamental principle. According to one respondent from our survey, the greater the similarity between the requirements of two projects, the more likely it is to produce similar software, and vice versa. This perspective underscores the belief that shared requirements are a strong indicator of software similarity.

"I believe that if two projects have similar requirements, the resulting software will be similar too. The more the requirements overlap, the more alike the software will be."

It was observed that the correlation between requirement and software similarity is significantly influenced by the structure and formality of the requirements. When requirements are well-structured and clearly defined in natural language, there is a stronger relationship between the requirements and the corresponding software, making it more feasible to reuse them for new projects. This highlights the importance of precise and well-documented requirements in the software development process, as they not only ensure better alignment with the final product but also enhance the reusability of the requirements for future projects.

"The structure of how the requirements are written also plays an important role in detecting whether they can be reused or not."

On the other hand, it is also considered that two sets of requirements are likely to produce similar software if they aim to achieve the same goals and require similar functionalities. When the objectives and functional needs of the projects align, the software solutions developed will naturally share many similarities. This highlights that not only the detailed requirements but also the overarching goals and functionalities play a crucial role in determining software similarity.

"Two software applications are considered similar if they serve similar purposes, offer overlapping features, target the same user base, and use comparable technologies. Evaluation involves comparing their functionalities, user interfaces, performance, and underlying technologies."

4.4.4 Theme 4: Perspective on Similarity Between Existing and Produced Software

Fourteen out of sixteen participants agreed that the code from an existing system (Python Code Excerpt 1) could be reused for a new system (Python Code Excerpt 2) due to the significant overlap in requirements, suggesting that software reusability is influenced by domain-specific similarities. This high level of agreement highlights the potential for effective code reuse with minor adjustments, particularly in domains like e-commerce, where similar requirements often lead to similar software solutions.

1. "Both the projects are for e-commerce domain and most of the requirements are similar so it can be reused."
2. "Both present the ecommerce database models and can be reused for any ecommerce store."

Some responses also indicate that one reason two projects are considered similar, and thus the first can be reused for creating the second, is that much of the code structure is the same. With some minor adjustments, the existing code can be adequately adapted for use in the new project. The requirements for the above-provided code show similarity in requirements, reinforcing the potential for code reuse.

1. "Most of the structure is same. With some minor tunings it should be adequate for usage."
2. "The requirements for the above provided code show similarity in requirements. Both represent classes for E-commerce project."

Chapter 5

Discussion

This chapter provides an in-depth analysis of the results obtained from applying our proposed methodology to the e-commerce domain, with a focus on evaluating the effectiveness of using BERT and CodeBERT models for requirement and code similarity assessment. Our study sought to build upon existing research, particularly the work conducted in domain-specific contexts such as the railway industry, by extending the application of natural language processing techniques to a broader and more commercially relevant domain. The discussion will highlight the key findings, compare our results with those of prior studies and explore the implications of our approach for requirement engineering and software development. By examining the strengths and limitations of our model, we aim to underscore the potential for further advancements in automated code recommendation systems and their practical applications in diverse industry settings.

5.1 Comparative Analysis of Model Performance

The scatter plot from our research, depicts the relationship between document similarity and code similarity within the e-commerce domain, demonstrates a significantly stronger and more consistent correlation compared to the scatter plot presented in the sample paper for the railway domain. In the sample paper, while there is a positive trend, the data points are widely dispersed, particularly in the lower similarity range (between 0.5 and 0.7 SS), indicating variability and less reliable similarity predictions by the BERT model.

In contrast, our scatter plot shows a clear and consistent upward trend, with data points closely aligned along the trend line. This tighter clustering suggests that our preprocessing steps and the application of BERT in the e-commerce domain have resulted in a more robust model that can reliably predict code similarity based on document similarity. The increased density of data points towards the higher range of similarity (0.4 to 0.9) emphasizes our approach's effectiveness, showcasing that our method excels in capturing requirement similarities and generating precise code suggestions.

The box plots that were produced using BERT and pre-processing for e-commerce requirements, when compared to the sample paper looking at the railway industry, show improvements in several important aspects. It tends to have more consistent and accurate relationship levels between requirements and software through more narrow interquartile ranges and higher median similarity scores with effective outlier management. These improvements are a sign of how BERT can bring out its flexibility and strength with pre-processing to make it more fitting for the dynamic and diverse e-commerce domain, rather surpassing the stricter analysis reserved for the railway sector.

This enhancement showcases the flexibility and applicability of our method, indicating that it excels not only in e-commerce but can also be successfully used in different domains.

5.2 Performance Comparison of Models in Requirement Similarity Calculation

In this section, we have compared the performance of BERT with other language models that have been utilized by the previous researchers. These models including **TF-IDF**, **JSI**, **USE**, **Doc2Vec**, and **FastText** have been used for calculating document similarity in the previous research. The **scatter plots** and **box plots** obtained after the application of these models on our dataset of requirements, clearly demonstrate better performance of BERT in capturing document similarities.

5.2.1 BERT Performance

BERT consistently shows a higher median similarity score across the dataset. The **box plot** indicates that BERT has a narrower interquartile range and fewer outliers, reflecting its ability to produce more stable and accurate similarity scores. This strong performance is due to BERT's contextual understanding of words and sentences, which enables it to capture deep semantic relationships within documents.

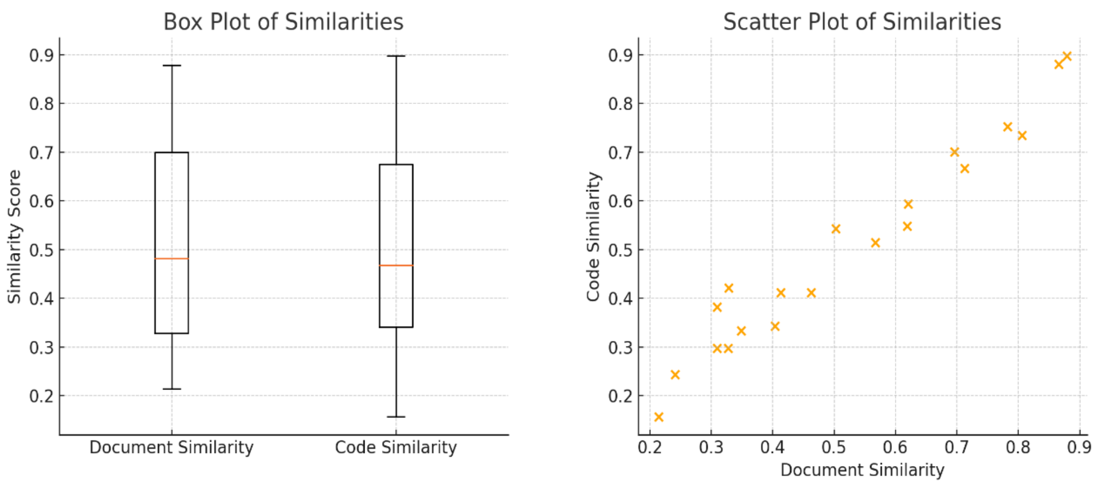

Figure 5. 1: Box plot and Scatter plot of Similarities calculated using BERT

5.2.2 TF-IDF Performance

TF-IDF, being a conventional bag-of-words model, performs the least in this comparison. As observed in the box plot, its similarity scores are significantly lower and exhibit a wider range. In TF-IDF it is expected as it only considers word frequency and does not take in account the context or order of words being used, making it less capable of capturing the deep semantical meaning that BERT excels in.

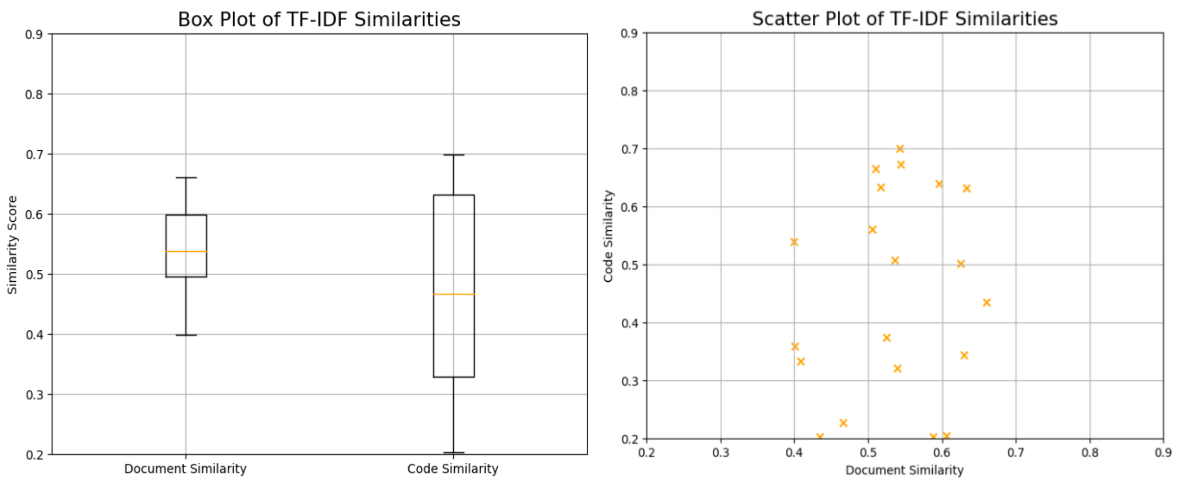

Figure 5. 2: Box plot and Scatter plot of Similarities calculated using TF-IDF

5.2.3 JSI (Jaccard Similarity Index) Performance

JSI performs a little better than TF-IDF, but this language model still falls short when compared to BERT. The **box plot** demonstrates that JSI's median similarity score is lower, and its variance is wider. As JSI is based on set comparison and focuses on word overlap, it lacks the ability to measure semantic relationships beyond simple word matching, which limits its effectiveness for document similarity.

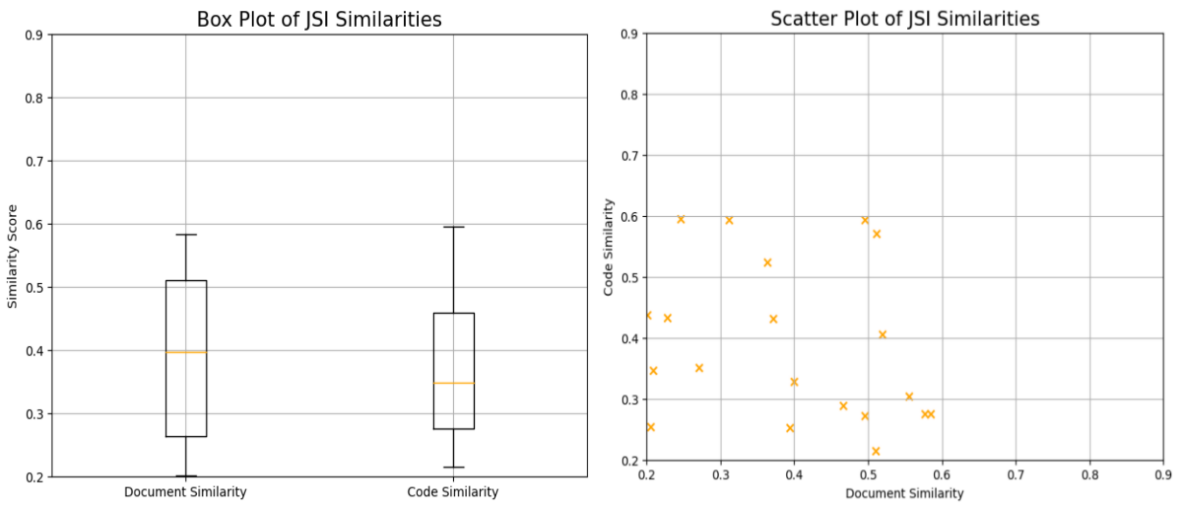

Figure 5. 3: Box plot and Scatter plot of Similarities calculated using JSI

5.2.4 Universal Sentence Encoder (USE) Performance

The **scatter plot** indicates that USE tends to cluster around the middle range of similarity scores, but it fails to achieve the same higher similarity values that BERT consistently captures. While USE embeds sentences into vectors, it is not as sophisticated at capturing the contextual relationships as BERT's transformer-based approach.

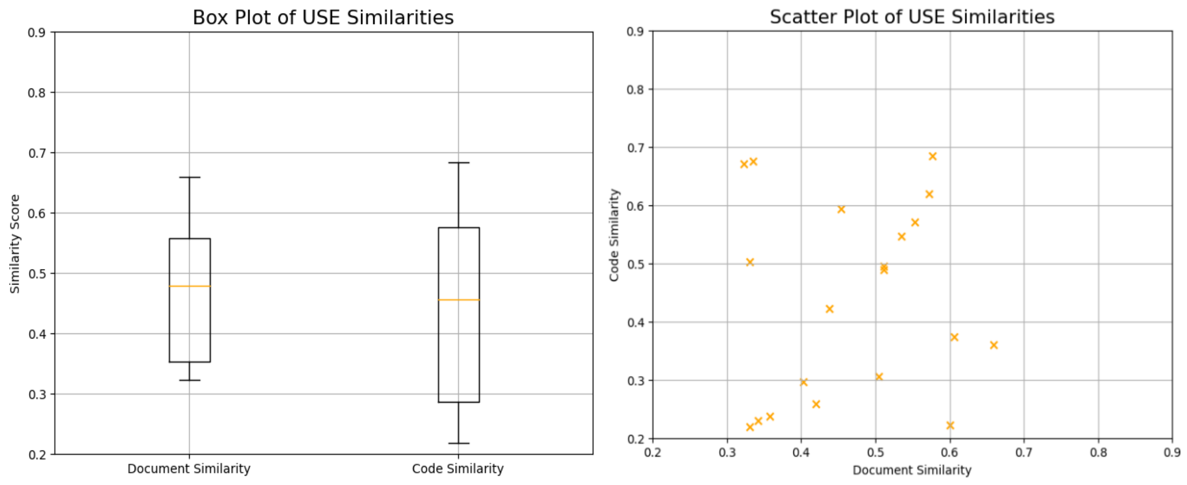

Figure 5. 4: Box plot and Scatter plot of Similarities calculated using USE

5.2.5 Doc2Vec Performance

When it comes to capturing complex relationships between the documents, Doc2Vec, a model based on distributed representations of documents, performed better than TF-IDF and JSI, but is still outclassed by BERT. The **box plot** for Doc2Vec shows a moderate range of similarity scores, but it struggles with capturing complex, nuanced relationships between documents. This is largely because Doc2Vec's embedding process is more static and less flexible compared to BERT's dynamic word representations.

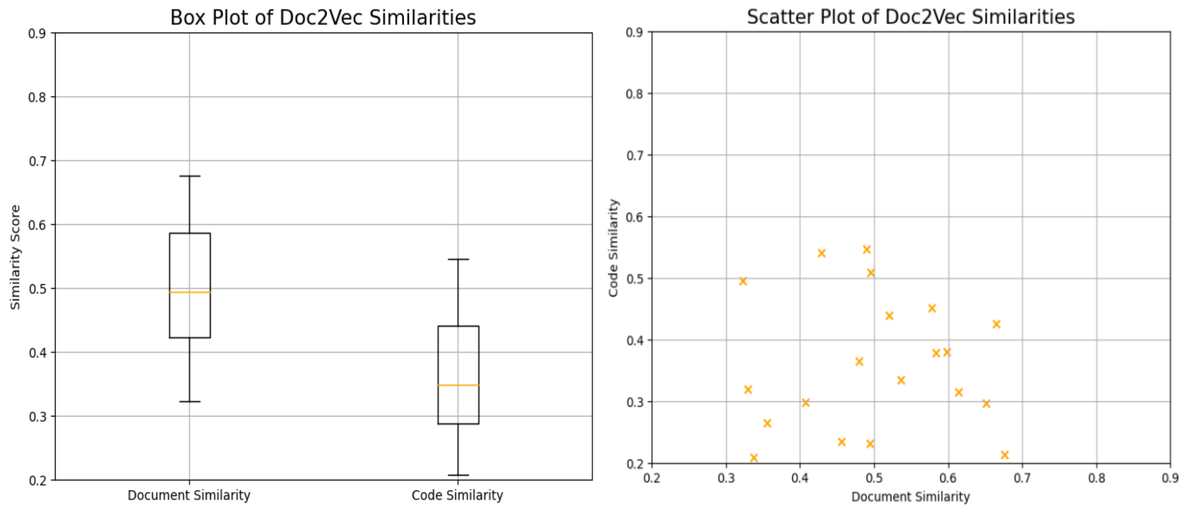

Figure 5. 5: Box plot and Scatter plot of Similarities calculated using Doc2Vec

5.2.6 FastText Performance

FastText, though an improvement over more conventional models such as TF-IDF and JSI, but it still failed to perform well as compared to BERT. As it can be seen in its **box plot** that shows a wider range and lower median similarity score. Although FastText uses subword information; to improve it understanding of rare words, it is still not the advanced contextual understanding that BERT provides.

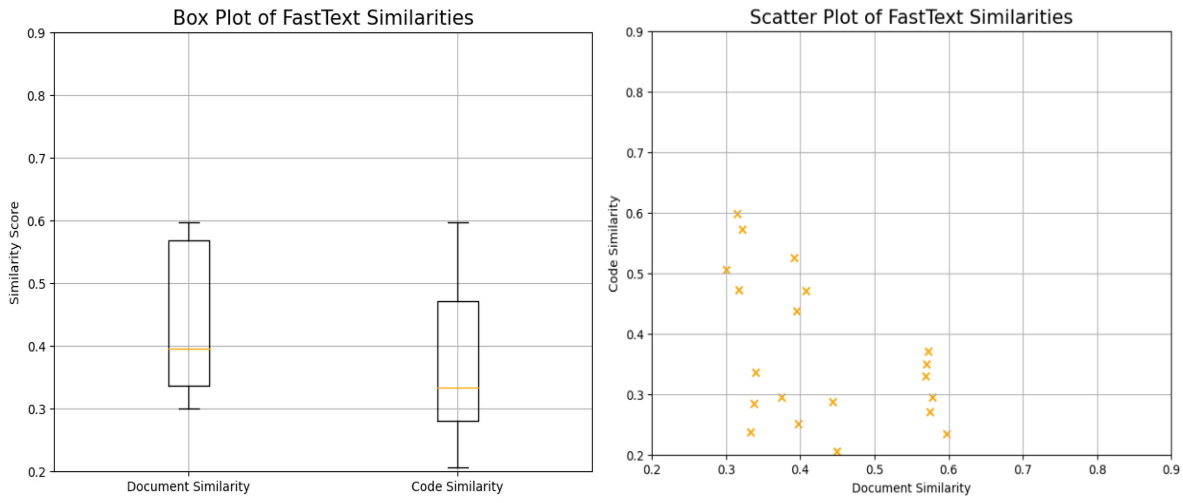

Figure 5. 6: Box plot and Scatter plot of Similarities calculated using FastText

To sum it up it can be said that BERT has outperformed all the traditional models in calculating document similarity. Because of BERT's transformer architecture ,it has allowed to capture both syntactic and semantic nuances in the text, which leads to a higher and a more consistent similarity scores. In contrast, traditional models like TF-IDF and JSI rely heavily on word frequency and overlap, which limits their effectiveness. Even more modern models like USE, Doc2Vec, and FastText, while better than traditional models, still do not match the performance of BERT in capturing the deep relationships between documents.

5.3 Performance Comparison of Models in Requirement Similarity Calculation

The performance of BERT in calculating document similarity is significantly influenced by the number of epochs used during training. In this analysis, we examine the behavior of BERT with fewer epochs (1 epoch), the default setting (3 epochs), and an increased number of epochs (6 epochs). The scatter plots provide a visual representation of BERT's document similarity scores under each of these conditions.

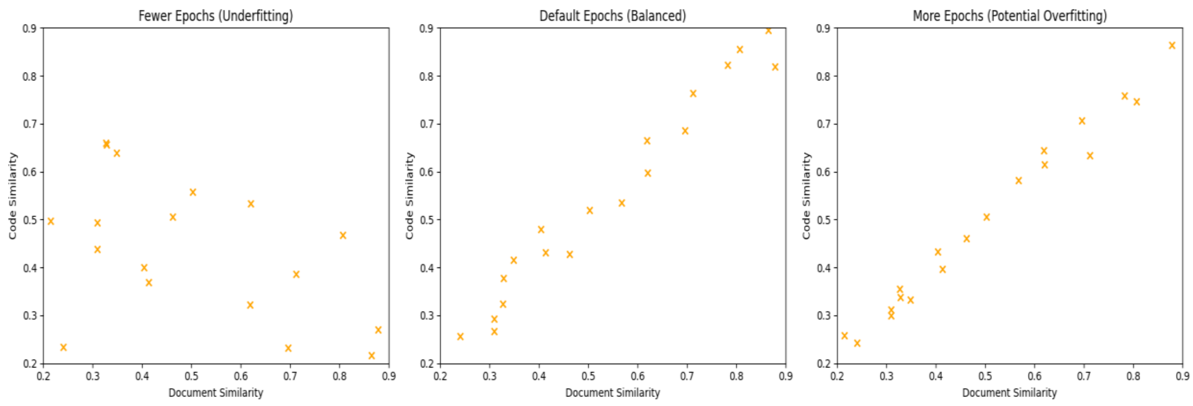

Figure 5. 7: Scatter plot BERT performance for calculating document similarity under different epochs

5.3.1 Fewer Epochs (1 Epoch)

The limited training time prevents the model from learning deeper patterns in the data, leading to a wide range of similarity scores and an inconsistent relationship between document and code similarity. The model tends to underfit the data after only 1 epoch of training. This underfitting is visible in the scatter plot, where the document similarity and code similarity scores exhibit high variance and a scattered pattern.

5.3.2 Default Epochs (3 Epochs)

The scatter plot with 3 epochs demonstrates balanced training. Here, the model achieves more consistent results, with tighter clustering of similarity scores. The points reflect a more stable relationship between document and code similarity. Training BERT for 3 epochs provides enough time for the model to capture meaningful relationships without overfitting, making this setting effective for most document similarity tasks.

5.3.3 Default Epochs (3 Epochs)

When BERT is trained for 6 epochs, the scatter plot shows closely packed data points, but this may be a sign of potential overfitting. Even while the model seems to be performing fairly well on the training data, the similarity scores may become inflated as the model starts memorizing patterns that unique to the training set. This may make it more difficult for the model to to generalize to new unseen data, which would lower its overall effectiveness in real-world applications.

5.3.4 Conclusion

From these experiments, we observe that training BERT with 3 epochs provides the best balance between underfitting and overfitting. Too few epochs (1 epoch) result in an unstable model, while too many epochs (6 epochs) risk overfitting, reducing the model's generalization capabilities. The scatter plots demonstrate how the choice of epochs directly impacts the clustering of similarity scores and the model's overall performance.

5.4 Comparative Analysis of Model Performance (CodeBERT vs JPlag)

We settled on deploying CodeBERT in our work to evaluate code similarity due to the immense advantages it brings in comparison with the traditional tools such as JPlag. As much as JPlag excels in plagiarism detection through syntax analysis, this proves to be limiting if the match has been restructured or written with different styling preferences. According to performance metrics from our study, JPlag had an accuracy of 75%, a recall of 70%, and a precision of 80%—this indicated a slight limitation regarding this aspect. On the flip side, being a transformer model trained on source code, CodeBERT can understand the semantic and structural complexities of code; it can realize more complex functional similarities beyond syntax. This is because it learned from data and captures variable usages that are likely not seen during training. As we also noted from experiment results, performance metrics from CodeBERT showcased better percentages: 86% accuracy, 88% recall, and 82% precision. CodeBERT offers a stronger and more detailed way to measure similarity by producing embeddings that capture the code's fundamental logic and functionality. This was especially crucial in our study, as finding reusable code through requirement similarity necessitated a tool capable of accurately evaluating functional equivalence, even in cases of different code implementation. Utilizing CodeBERT led to improved and more meaningful findings, in line with our aim to enhance code reusability in software development.

Table 5. 1: Comparison Metrics of JPlag and CodeBERT

Metric	**CodeBERT**	**JPlag**
Accuracy	86%	75%
Recall	88%	70%
Precision	82%	80%

Chapter 6

Conclusion and Future Work

In this study, we empirically investigated the relationship between requirements similarity and code similarity within the e-commerce domain. Our results indicate a strong correlation (95%) between similar requirements and the similarity of their corresponding code implementations. Using BERT for requirements analysis and CodeBERT for code similarity assessment, we demonstrated that content-based recommender systems could effectively retrieve reusable code based on requirement similarity. This approach significantly aids requirement engineers and developers in streamlining the development process and reducing redundant coding efforts. Our findings reinforce the notion that similar requirements can indeed serve as reliable proxies for retrieving similar code, thereby supporting efficient code reuse with limited adaptation.

6.1 Comparative Analysis of Model Performance

Building on the promising results of this research, several avenues for future work can be explored:

1. **Domain Expansion:** Extend the investigation to other domains beyond ecommerce to validate the generalizability of the findings across diverse software projects.
2. **Enhanced Models:** Experiment with more advanced natural language processing models and techniques, such as transformer-based models with domain specific fine-tuning, to further improve requirement and code similarity assessments.
3. **Automated Adaptation:** Explore methods to automate the adaptation of retrieved code snippets to better fit the specific context and constraints of new projects, reducing the need for manual adjustments.
4. **Evaluation Metrics:** Refine and expand evaluation metrics to include aspects such as code quality, maintainability, and performance, ensuring that the recommended code not only satisfies functionality, but also adheres to best practices in software engineering.

By pursuing these future directions, we aim to enhance the robustness and applicability of content-based recommender systems, ultimately contributing to more efficient and effective software development processes.

References

[1] Abbas, M. et al. (2023) 'On the relationship between similar requirements and similar software: A case study in the railway domain', Requirements Engineering, 28(1), pp. 23–47. Available at: https://doi.org/10.1007/s00766-021-00370-4.

[2] Alexander Felfernig, M.A.M.S.S.R.C.P.A.F.X.F. (2017) 'OpenReq: Recommender Systems in Requirements Engineering', in International Conference on Knowledge Technologies and Data-Driven Business Workshop.

[3] Arora, C. et al. (2017) 'Automated Extraction and Clustering of Requirements Glossary Terms', IEEE Transactions on Software Engineering, 43(10), pp.918–945. Available at: https://doi.org/10.1109/TSE.2016.2635134.

[4] N., C.-H. C., C.-H. J., M. B. Hariri, 'Recommendation Systems in Requirements Discovery', in Recommendation Systems in Software Engineering, 2014, pp. 455–476.

[5] C., F. X., F. D. Palomares, 'Personal Recommendations in Requirements Engineering: The OpenReq Approach', in In: Kamsties, E., Horkoff, J., Dalpiaz, F. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ, Springer, Cham., 2018.

[6] C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher, 'Enhancing stakeholder profiles to improve recommendations in online requirements elicitation', in Proceedings of the IEEE International Conference on Requirements Engineering, 2009, pp. 37–46. doi: 10.1109/RE.2009.20

[7] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer, and Z. Niu, 'Automated recommendation of software refactorings based on feature requests', in Proceedings of the IEEE International Conference on Requirements Engineering, IEEE Computer Society, Sep. 2019, pp. 187–198. doi: 10.1109/RE.2019.00029.

[8] J. Natt och Dag, B. Regnell, V. Gervasi, and S. Brinkkemper, 'A LinguisticEngineering Approach to Large-Scale Requirements Management'.

[9] M. Irshad, K. Petersen, and S. Poulding, 'A systematic literature review of software requirements reuse approaches', Jan. 01, 2018, Elsevier B.V. doi: 10.1016/j.infsof.2017.09.009.

[10] O. C. Z. G. J. H. H. P. M. A. Z. Jane Cleland-Huang, 'Software traceability: trends and future directions', in FOSE 2014: Future of Software Engineering Proceedings, 2014, pp. 55–69.

[11] P. R. & A. A. Markus Borg, ‘Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability’, Empir Softw Eng, vol. 19, pp. 1565–1616, 2014.

[12] D. Falessi, G. Cantone, and G. Canfora, ‘Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques’, IEEE Transactions on Software Engineering, vol. 39, no. 1, pp. 18–44, 2013, doi: 10.1109/TSE.2011.122.

[13] M. S. A. G. F. Z. Chetan Arora, ‘Change impact analysis for Natural Language requirements: An NLP approach’, in IEEE 23rd International Requirements Engineering Conference (RE), 2015.

[14] Markus Borg; Krzysztof Wnuk; Björn Regnell; Per Runeson, ‘Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context’, IEEE Transactions on Software Engineering, vol. 43, pp. 675–700, 2017.

[15] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, ‘Automated Extraction and Clustering of Requirements Glossary Terms’, IEEE Transactions on Software Engineering, vol. 43, no. 10, pp. 918–945, Oct. 2017, doi: 10.1109/TSE.2016.2635134.

[16] A. Shatnawi, A. D. Seriai, and H. Sahraoui, ‘Recovering software product line architecture of a family of object-oriented product variants’, Journal of Systems and Software, vol. 131, pp. 325–346, Sep. 2017, doi: 10.1016/J.JSS.2016.07.039.

[17] B. Wang, R. Peng, Y. Li, H. Lai, and Z. Wang, ‘Requirements traceability technologies and technology transfer decision support: A systematic review’, Journal of Systems and Software, vol. 146, pp. 59–79, Dec. 2018, doi: 10.1016/J.JSS.2018.09.001.

[18] W. Wang, N. Niu, H. Liu, and Z. Niu, ‘Enhancing automated requirements traceability by resolving polysemy’, in Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018, Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 40–51. doi: 10.1109/RE.2018.00-53.

[19] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, ‘Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models’, Feb. 2021, [Online]. Available: http://arxiv.org/abs/2102.04411

[20] C.-H. C. C.-H. J. M. B. Hariri N, Recommendation Systems in Requirements Discovery. 2014.

[21] D. Theosaksomodwi and H. Widyantoro, ‘Conversational Recommender System Chatbot Based on Functional Requirement’, 2019.

[22] E. Dilorenzo et al., ‘Enabling the Reuse of Software Development Assets through a Taxonomy for User Stories’, IEEE Access, vol. 8, pp. 107285–107300, 2020, doi: 10.1109/ACCESS.2020.2996951.

[23] S. Reddivari and J. Wolbert, ‘Calculating Requirements Similarity Using Word Embeddings’, in Proceedings - 2022 IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 438–439. doi: 10.1109/COMPSAC54236.2022.00079.

[24] Z. Feng et al., ‘CodeBERT: A Pre-Trained Model for Programming and Natural Languages’, Feb. 2020, [Online]. Available: http://arxiv.org/abs/2002.08155

[25] Wei Yuan; Yaoxu Lei; Xin Guo, ‘Research on text similarity calculation based on BERT and Word2Vec’, 2022.

[26] Sudharsan Ravichandiran, Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT. Packt Publishing, 2021.

[27] M. Ciniselli, L. Pascarella, E. Aghajani, S. Scalabrino, R. Oliveto, and G. Bavota, ‘Source Code Recommender Systems: The Practitioners’ Perspective’, Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.04098

[28] L. Zhang, C. Cao, Z.Wang, and P. Liu, ‘Which Features are Learned by CodeBert: An Empirical Study of the BERT-based Source Code Representation Learning’, Jan. 2023, [Online]. Available: http://arxiv.org/abs/2301.08427

[29] Y. Peng, J. Zhang, Y. Huang, and J. Tang, ‘Research on Quality Evaluation of Requirement Analysis Specification Based on Text Similarity Calculation’, in Proceedings - 2021 International Conference on Information Science, Parallel and Distributed Systems, ISPDS 2021, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 60–63. doi: 10.1109/ISPDS54097.2021.00018.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding’, Oct. 2018, [Online]. Available: http://arxiv.org/abs/1810.04805

[31] Breen, R. L. (2006). A Practical Guide to Focus-Group Research. Journal of Geography in Higher Education, 30(3), 463–475. https://doi.org/10.1080/03098260600927575