CMPE 492

A Software Project Requirement Tracking Analysis Tool

Report

Ecenur Sezer

Kadir Ersoy

Advisors:

Ph.D. Suzan Üsküdarlı

Ph.D. Fatma Başak Aydemir

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Broad Impact

The development of a software project for Requirement Traceability is potentially significant for the software development process. The project incorporates a tool that can automatically generate graphs showing the relationship and similarity between software requirements and software artifacts like issues, pull requests (PRs), and commits. With the help of this tool, developers can more easily identify dependencies and track the evolution of requirements throughout the development process, resulting in more efficient development and improved software quality. Moreover, the tool can provide stakeholders with valuable insight into the project's progress, enabling them to make informed decisions about resource allocation and prioritization. Ultimately, this project aims to enhance the efficiency, effectiveness, and quality of software development processes, which can have a wide-ranging impact.

1.2. Ethical Considerations

The development of a software project for Requirement Traceability has significant ethical implications. The project requires access to software artifacts, which necessitates a requirement specification document and GitHub repository, raising concerns about data privacy and access control. It is essential to develop the project with appropriate security measures to safeguard the confidentiality of the data. Furthermore, the use of a graphical database for visualization necessitates careful attention to data accuracy, integrity, and ownership. The ethical consequences of utilizing this database must be thoroughly evaluated to ensure that all stakeholders are aware of the potential risks and benefits associated with the project. The project should also be developed and implemented in a fair and impartial manner, taking into account the diversity and inclusivity of its users.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

Requirement Traceability refers to analyzing a requirement's life and its relations with other items in both ways (forwards and backward) in an application. [1] The significance of requirements traceability cannot be overstated, as it enables stakeholders to monitor the fulfillment of requirements and allows system engineers to track the progress and evolution of functionalities, especially in large-scale software projects. In this context, this project endeavors to develop an automated tool that leverages similarity techniques on software artifacts like issues, PRs, and commit messages to trace the lifespan of software requirements. The resulting traces are presented in a graph structure, replete with distinct nodes for software artifacts. Different types of links are employed to demonstrate the nature of the relationships between these artifacts, such as similarity links from requirements to issues, PRs, or commits, and implementation links from PRs to issues. The graph construction process is automated, with traces and artifact nodes generated automatically using data obtained from GitHub repositories and the requirement specification document. Moreover, the nodes store the number of software artifacts associated with each requirement, as well as various properties of issues, PRs, and commits, such as title, description, message, comments, and status. Ultimately, the graph structure serves as a clear and concise visual representation of the lifespan of requirements and the interdependencies of software artifacts in project development.

The entire documentation and the source code are present in GitHub Repository.

2.2. Project Planning

There are five milestones distributed along the lifetime of the project to set a roadmap. These include; Literature Research, Development of the Initial Prototype, Midterm Report, Evaluation of the Results, and Documenting the Research. To facilitate tracking of the project's progress, a GitHub repository is utilized, where every issue created is linked to its corresponding milestone. This system allows for streamlined monitoring of the project's status.

Literature Research

During the first week, the team's main objective was to review and condense research studies pertaining to Requirement Analysis, Requirements Quality, and Requirements Traceability. They utilized keyword searches and snowball methodology to conduct their research. The team analyzed a total of 20 research papers and summarized 13 of them on the project repository's wiki. The team also presented their findings during their weekly meetings and discussed the most appropriate topic to develop a software tool that would enhance software quality. After careful consideration, the team opted to concentrate on Requirements Traceability as it remains an unresolved problem that requires automated solutions.

During the second week, the research focused on two related topics: "Requirements Traceability" and "Traceability Recovery". The purpose was to gain a better understanding of the current knowledge and practices related to tracing software requirements. The team reviewed various articles that provided different definitions of traceability and explored several approaches to establish trace links between software artifacts and requirements. Some of the articles discussed UML design approaches, while others focused on workbenches or editors used in the industry for this purpose. The team also looked into automated traceability efforts and information retrieval techniques that can be used to identify traces between software artifacts. After discussing and documenting these different approaches, our team decided to focus on tracing the link between software artifacts and requirements to observe their lifecycle. This will help us better understand how requirements evolve over time and how they are translated into software artifacts.

Development of the Initial Prototype

During the third week, the team conducted a feasibility analysis by examining open-source projects to determine what is possible for traceability and what kind of information is available in repositories to achieve it. They tested open-source repositories with various engines, to measure the semantic similarity between different artifacts, such as requirements, design documents, issue titles, commit messages, and pull requests. The team also performed a manual evaluation to identify traces between artifacts and assess how well the requirements were met. The ultimate goal is to automate this evaluation process. Furthermore, the team documented the performance of the tools used in the repository wiki for future reference.

During week 4, the team examined two open-source software projects - Learnify [2] and BUcademy [3] - which were developed for the Introduction to Software Engineering course at Bogazici University. The team had access to the projects' requirements specification documents as well as related development artifacts, such as issues, pull requests, and commits. The primary goal of the analysis was to identify evidence of requirements within the software artifacts and to assess the overall completion of those requirements. Additionally, the team observed the relationship between the quality of the requirements and the presence of evidence within the artifacts. To document their findings, the team conducted a simple keyword search of the titles and descriptions of the artifacts and then tracked the lifecycle of the requirements. This approach allowed the team to observe the natural language traces of requirements and to document the development of those requirements over time. Upon reviewing the results of the analysis, the team decided to create a graph of the traces to facilitate further analysis of the repository. Detailed analysis can be found in our wiki page.

During the fifth week of the project, the team had a discussion about the design of a traceability graph. They decided on a graph model that would include different types of nodes representing software artifacts and various types of links that would show the relationships between them. The team created graphs to represent the traces that had been recorded during the previous week for two different projects: Learnify (Figure A.1) and BUcademy (Figure A.2). These graphs made it easy to detect relationships between artifacts that were directly linked to each other. In addition, nodes that had overlapping links provided a more comprehensive understanding of the relationships between the artifacts. The quality of the requirements was also a topic of debate among the team members, as some requirements had more complex trees and were linked to more artifacts than others. The manually conducted graphs can be found in our wiki page.

During the sixth week, the team directed their efforts toward automating the construction of traceability graphs. These graphs are typically created manually during the previous week. The team's initial objective was to gather the necessary data and quickly automate the trace-finding process in order to determine the possibilities and limitations of automation. The team planned to handle the graph construction aspect at a later stage, after developing a working prototype that recovers trace links. The development of the initial prototype was carried out in three phases: Keyword extraction, Data parsing, and Trace building. For further information regarding the implementation of the automated graphs, please refer to section 7.1.

During the seventh week, the team focused on exploring various methods to enhance the accuracy and reduce noise in keyword extraction libraries. They discussed the utilization of POS tagging and dependency parsing to identify specific word groups and initiated the development of a customized pipeline for keyword extraction.

In the eighth week, the team established a connection with Neo4j, a graph database, but encountered the absence of relationships within the database. They deliberated on visualizing the preliminary results using the graph database and made efforts in that direction. Additionally, they implemented the keyword extractor pipeline; however, due to the noise originating from extracted nouns and verbs, they decided to capture noun-phrases instead.

During the ninth week, the team obtained the initial results from the Neo4j database, which were found to be quite noisy. Consequently, they discussed the incorporation of different types of relationships between artifacts to enhance the visualization quality. The keyword extractor was finalized, and the focus of discussion shifted more towards the Neo4j database.

In the tenth week, the team created new relationships between pull requests (PRs) and commits while reducing the direct traces to requirement nodes. They improved the methods for capturing traces using word vectors and keyword extractors. To evaluate the accuracy of trace record capture, they developed a ground truth set. Additionally, a dashboard to display statistical data about the repository was created to help users understand the current status of the project analyzed with the tool.

In the eleventh week, the keyword extraction method was enhanced to include dependency parsing links involving prepositions, resulting in improved accuracy of the tool. However, the recall and precision values of the word vector model indicated that it was not complex enough to achieve useful accuracy. Therefore, the team switched to using the tf-idf algorithm, which produced better results.

During the twelfth week, the team prepared the final report and poster for the concluding presentations. This documentation was the result of effective teamwork and ensured the team's readiness for the evaluation process.

In the thirteenth week, the team documented the article in IEEE format to submit it to the Software Traceability workshop.

Midterm Report

During the seventh week, the primary goal was to prepare the midterm report. Additionally, the team reviewed and discussed the initial prototype. Suggestions were made for enhancing the prototype, including the addition of dependency parsing to the keyword extraction process, as well as parsing the requirements using headers.

Evaluation of the Results

Once the development of the tool is concluded in accordance with the specified requirements, the evaluation phase will commence, encompassing the assessment of the tool's performance using diverse methodologies and the provision of statistical data pertaining to its accuracy. In addition to empirical data, comprehensive analysis will be conducted to address various aspects, including the successful completion of the project, the developmental roadmap followed, the current status of the requirements, and an assessment of the quality and potential challenges associated with individual requirements. The evaluation of these aspects is contingent upon the specific objectives and expectations of the user in utilizing the tool. However, it is still possible to evaluate the results in terms of our own intrinsic motivation. By considering these factors and conducting a thorough analysis, a comprehensive evaluation of the tool's effectiveness and overall success can be achieved.

Documenting the Research

The documentation of research milestones involves the systematic recording and presentation of progress and findings at a specific stage of a research project. To adhere to the IEEE Paper Format [4], an academic article detailing these findings must be written before the milestone deadline, which is set for June 8th. This date has been chosen for two reasons. Firstly, the presentation of the bachelor theses at Bogazici University is scheduled for June 9th. Additionally, 31st IEEE International Requirements Engineering Conference - Software and Systems Traceability Workshop [5] also has a submission deadline around this time, and the team plans to submit their project to this workshop. By meeting this milestone deadline, the team can ensure timely dissemination of their research findings and maximize their chances of showcasing their work at academic forums.

2.2.1. Project Time and Resource Estimation

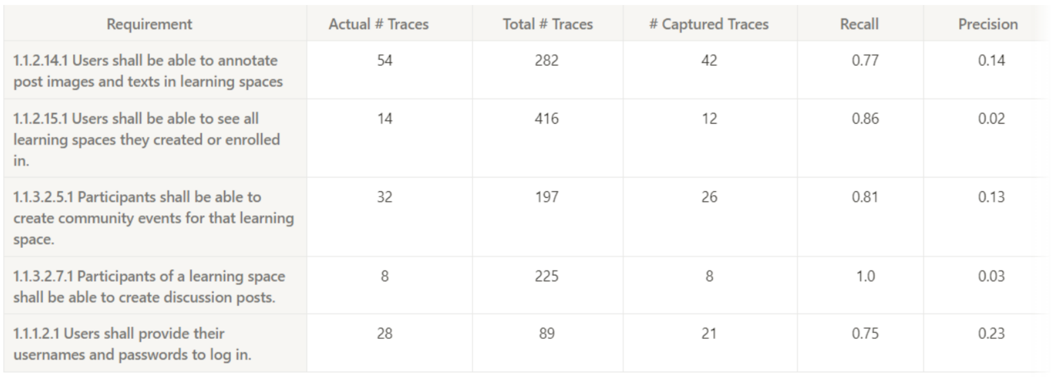
The project is expected to be completed within the 13-week duration of the semester. Any additional work can be conducted subsequently to further enhance the existing project. The timeframe has been divided into five milestones, as explained in the project planning section.

To conduct this project, the following resources may be required:

(i) Research materials: A thorough research of requirements analysis, requirements traceability and recovery, and automated traceability is essential.
(ii) Data: Software artifact data from a project needs to be collected to build and test traceability.
(iii) Software and tools: It may be necessary to utilize natural language processing tools to analyze and relate the textual data of each software artifact. Furthermore, a decision needs to be made regarding the software and tools to be used for storing and visualizing the data as a graph.

2.2.2. Success Criteria

Our main goal in this project is to create a roadmap for requirements traceability. Requirements traceability allows stakeholders to monitor the progress of requirements and helps system engineers track the effort and workload needed to fulfill those requirements. Therefore, our measure of success is to improve the traceability of requirements in software projects through automation. To assess the effectiveness of our methods, we have manually created sets of traces from requirements in the Learnify project artifacts. These sets are then used to calculate recall and precision values for our automated methods of capturing traces. At this stage, our measure of success branches into two categories: improved recall and improved precision values. Depending on the user's needs, the preferred method may vary, and all methods within this context are considered successful.



Requirement	Actual # Traces	Total # Traces	# Captured Traces	Recall	Precision
1.1.2.14.1 Users shall be able to annotate post images and texts in learning spaces	54	282	42	0.77	0.14
1.1.2.15.1 Users shall be able to see all learning spaces they created or enrolled in.	14	416	12	0.86	0.02
1.1.3.2.5.1 Participants shall be able to create community events for that learning space.	32	197	26	0.81	0.13
1.1.3.2.7.1 Participants of a learning space shall be able to create discussion posts.	8	225	8	1.0	0.03
1.1.1.2.1 Users shall provide their usernames and passwords to log in.	28	89	21	0.75	0.23

Figure 2.1. Keyword extraction method, ground truth values and recall-precision values

2.2.3. Risk Analysis

Potential risks:

- Not being able to acquire necessary data for requirement traceability. Most of the documents that this project aims to trace like requirements specifications, issues, and pull requests are usually confidential that companies do not share publicly.

- Requirement specification and issues having problematic descriptions and language. Since the tool will rely on software artifacts that accomplish similar jobs to have similar words and meanings, artifacts that work on the same subject but are written and explained with different words and contexts will not be linked. This can occur due to terminology differences or abbreviations etc.

- The ground truth set was not created by professional analysts, and was created by our team, which are the developers of the Learnify project. This creates a bias, hence we are capable of guessing the traces the automated tool will or will not capture.

2.2.4. Team Work (if applicable)

The teamwork approach taken by the partners in their research project is noteworthy for its meticulous documentation and tracking of progress. The partners meet on a weekly basis with their supervising professors to discuss their research progress, and action items are documented for later reference. Subsequently, the partners use GitHub issues to track their individual work and review each other's contributions. This collaborative effort, characterized by regular communication and thorough documentation, reflects a commitment to transparency, accountability, and effective project management. By adopting this teamwork approach, the partners are able to stay on track with their research objectives and produce high-quality output.

3. RELATED WORK

Introduction

A literature survey is an important tool for exploring existing research and knowledge on a particular topic. The purpose of our literature survey is to provide a comprehensive overview of the current state of research and practice on a particular topic. In this paper, we present a literature survey on Requirement Analysis and later stages, Requirements Traceability, with the goal of synthesizing and evaluating the current research and practice in this area. The summaries of related work present in this section are superficial. More detailed summaries of the following studies are present at Github Repository Wiki page.

Results

Analyzing Quality of Software Requirements; A Comparison Study on NLP Tools [6]

Afrah Naeem - Zeeshan Aslam - Munam Ali Shah, ICAC - 2019

The article compares five existing tools for requirements engineering, and proposes a new tool called the Requirement Assessment Tool (RAT) to assess the quality of requirement and specification documents. While some tools check for grammatical errors, natural language can be ambiguous and unclear, leading to multiple interpretations. RAT follows ISO/IEEE standards and identifies weak parts of the document, and provides recommendations to requirement engineers. The article also proposes quality standards for requirements, such as using imperatives and avoiding vague words or option phrases. RAT outperforms the other tools in all categories of quality indicators, but future work will involve evaluating more tools and generating more satisfactory tools that look for semantic errors as well.

An analysis of the requirements traceability problem [1]

O.C.Z. Gotel and C.W. Finkelstein - Proceedings of IEEE International Conference on Requirements Engineering, 1994

The article discusses techniques for providing requirement traceability (RT) and identifies diverse definitions and conflicts as reasons for the persistence of the RT problem. The importance of pre-RS (pre-requirement statement) traceability is highlighted as it can yield quality improvements and economic leverage. Additional computer metaphors can help produce more pre-RS information. The article concludes that to improve the RT problem, research efforts should focus on pre-RS traceability and continuous modeling of the social infrastructure involved in the requirement production process.

Implementing Requirements Traceability: A Case Study [7]

B. Ramesh, T. Powers, C. Stubbs, M. Edwards - Proceedings of 1995 IEEE International Symposium on Requirements Engineering, 1995

The study describes a case study of the use of traceability in systems development and management activities by an organization called WST. The approach to traceability varies among different actors involved in the process, such as the customer and maintenance engineer. WST uses traceability as a key component of its systems development and management activities. The organization captures various types of traceability information through source documents, such as design rationale and requirements rationale, and requirements tracing through traceability matrices. The WST identifies requirements for a proposed system from the customer's project management plan and validates them through various means. The use of traceability information throughout the project is observed through the system engineer's "Engineer's Notebook." The absence of a common and well-defined perspective on traceability contributes to the wide variation in current practices. However, the costs associated with implementing a comprehensive traceability scheme can be justified in terms of better quality of the product and the systems development and maintenance process with potentially lower life cycle costs.

PRO-ART: enabling requirements pre-traceability [8]

K. Pohl - Proceedings of the Second International Conference on Requirements Engineering, 1996

The study discusses the importance of developing a model for traceability information at the level of systems design, which should include bidirectional links to allow requirements to be traced forward from requirements to systems components, and backward from systems components to requirements. The model should also capture the rationale for design decisions and support various stakeholders in systems development activities, including the systems designer and project manager. The development of such models and reasoning mechanisms is the primary goal of ongoing research.

TraceLab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions [9]

Ed Keenan, Adam Czauderna, Greg Leach, Jane Cleland-Huang et al. - International Conference on Software Engineering (ICSE), 2012

The study describes the design and implementation of TraceLab, an experimental environment that allows researchers to design and execute experiments in a visual modeling environment using a library of reusable and user-defined components. TraceLab is constructed in .NET using the Windows Presentation Foundation (WPF) and experiments are composed of executable components and decision nodes laid out in a precedence graph on a canvas. Components can be primitive or composite, written in various memory-managed languages, and integrated into TraceLab by adding metadata information to the code. TraceLab datatypes support a wide range of primitives and community-defined data structures, as well as user-defined datatypes. The study then provides an example of an experiment using TraceLab to compare and combine the Vector Space Model (VSM), probabilistic Jensen and Shannon (JS) model, and Relational Topic Modeling (RTM) techniques. Researchers were able to construct the experiment using built-in and custom-built components, and developed new datatypes needed by the custom components.

Traceability Fundamentals [10]

J Cleland-Huang - OCZ Gotel - J Huffman Hayes - P Mäder - A Zisman, Proceedings of the on Future of Software Engineering - 2014

The article explains the concept of traceability in software development and the different elements associated with it, such as trace artifacts, trace links, and trace relations. The ability to achieve traceability depends on the creation of navigable links between software artifacts. Traceability enables processes such as change impact analysis, coverage analysis, and dependency analysis, contributing to better understanding of software and easing development and maintenance processes. Traceability strategies involve decisions about granularity, approaches for generating, classifying, representing, and maintaining artifacts, and assessing the quality and execution of traceability solutions. Traceability use includes supporting software engineering activities such as verification and validation, impact analysis, and change management. Different types of tracing are also explained, such as forward tracing, backward tracing, vertical tracing, and horizontal tracing. Finally, complexities concerning traceability are highlighted, such as the issue of trace granularity and the quality of the trace.

Improving Requirements Tracing via Information Retrieval [11]

Jane Huffman Hayes - Alex Dekhtyar - James Osborne, Proceedings 11th IEEE International Requirements Engineering Conference, 2003

The paper aims to improve requirements tracing by using information retrieval (IR) techniques. The paper presents three IR algorithms that use keyword-based retrieval and the vector model, which represents documents as vectors of keywords and their weights. The user query is also analyzed for keywords, and the relevance of documents to the query is expressed using a similarity measure. The paper evaluates the performance of these algorithms based on recall and precision metrics. The paper focuses on after-the-fact requirement tracing, where the requirements traceability matrix does not exist. They compare their model with a human analyst and an existing tracing tool, and the results suggest that the model outperforms both in terms of recall and sometimes in terms of precision. However, there are some potential issues, such as the size of the domain, implications of domain size, and query interdependence.

Automating Traceability Link Recovery through Classification [12]

Chris Mills, ESEC/FSE 2017

The paper discusses an automated approach for Traceability Link Recovery (TLR) using machine learning classification to determine the validity of links. Three types of features, including IR Ranking, QQ, and Document Statistics, are used to represent traceability links. The paper investigates several classification algorithms to evaluate the results and to minimize false positives. The results indicate that Random Forests have the best performance at minimizing false positives. The paper also aims to improve the approach by performing more in-depth feature engineering and addressing the class imbalance problem in future work.

Best Practices for Automated Traceability [13]

J Cleland-Huang - B Berenbach - S Clark - R Settimi - E Romanova, IEEE Computer journal, volume 40, 2007

The paper discusses the challenges of manual traceability and introduces an automated traceability method using a probabilistic network (PN) model called Poirot. Poirot returns a set of possible links for a human analyst to assess. The paper then presents best practices for automated traceability, including establishing a traceability environment, defining trace granularity, supporting in-place traceability, creating traceable artifacts with a well-defined project glossary, quality requirements, meaningful hierarchy, bridging intradomain semantic gap, and incorporating domain knowledge to improve traceability. The evaluation metric for Poirot is recall and precision, and there is a tradeoff between these two metrics.

4. METHODOLOGY

The project is currently being developed and managed through GitHub, where the team convenes on a weekly basis to provide updates on the status of the project, discuss the outcomes of the preceding week's tasks, and strategize for the forthcoming week. The records of these meetings, including the minutes of the discussions and the action items assigned to each member, are documented and stored in the project's GitHub repository.

To facilitate collaboration amongst the team members, an issue is generated in GitHub for each task, allowing for progress monitoring and providing a space for team members to contribute to the results and findings associated with the task. These contributions are then reviewed by the team.

At the conclusion of each week, the team reviews and refines the results of each task before documenting them on a relevant wiki page. These outcomes are later discussed during the team's next meeting, where an agenda is prepared to highlight the key topics for discussion.

Communication between team members and their professors is facilitated via Discord, ensuring that all members remain up-to-date and informed throughout the course of the project.

5. REQUIREMENTS SPECIFICATION

1. Functional Requirements

1.1 The system shall be able to extract software artifacts from GitHub repositories and the requirement specification document to generate a graph structure.

1.2 The system shall employ similarity techniques to capture trace links between requirements and software artifacts such as issues, PRs, and commit messages.

1.2.1 The system shall perform keyword extraction on the requirement specification document.

1.2.2 The system shall perform a keyword search on the textual data of the software artifacts.

1.3 The system shall automatically generate artifact nodes and trace links in the graph structure.

1.3.1 The system shall store the software artifacts that have keyword matches as trace links.

1.3.2 The system shall differentiate between different types of links.

1.4 The graph structure generated by the tool shall provide a visual representation of the trace links and software artifacts as nodes of a graph.

1.5 The nodes in the graph structure shall store information about the number of software artifacts associated with each requirement and various properties of issues, PRs, and commits, such as title, description, message, comments, and status.

1.6 The tool shall be customizable to accommodate different project requirements and specifications.

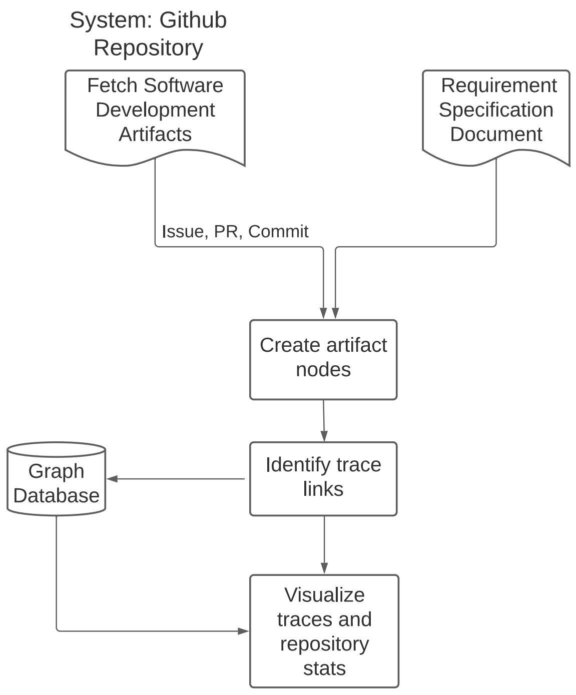
1.6.1 The stopword list used in keyword extraction should be customizable.

6. DESIGN

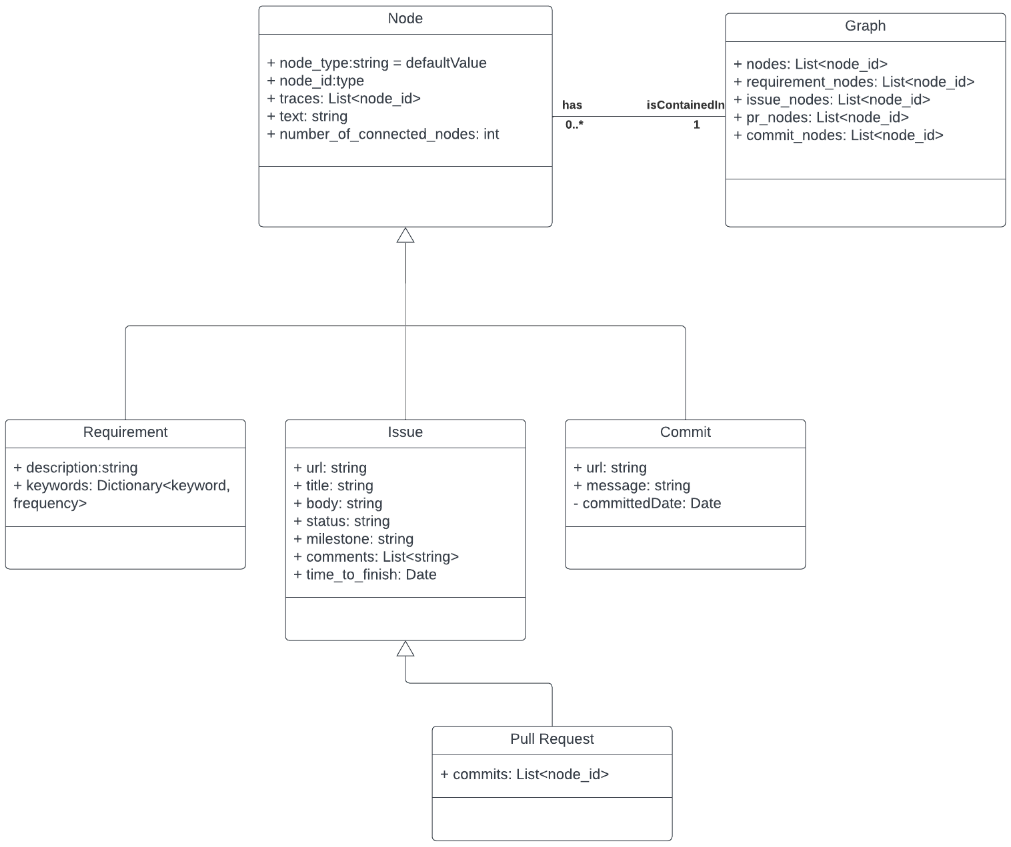
6.1. Information Structure

The tool requires software development artifacts fetched from GitHub repository and Requirement Specification Document. These information is then used to create "artifact nodes" with crucial information stored as node properties. In this step, the textual data of artifacts are concatenated and stored as "text" property for tracing. Later on, various methods are used for identifying trace links. Following the identification of traces, the data present are extracted into graph database. Overall, graph database stores artifact data as nodes and the trace links are kept as edges between nodes. In the final stage, statistical information of the repository are extracted into a dashboard for visualization of project status.

6.2. Information Flow


System: Github Repository
Fetch Software Development Artifacts
Requirement Specification Document
Issue, PR, Commit
Create artifact nodes
Identify trace links
Graph Database
Visualize traces and repository stats

6.3. System Design

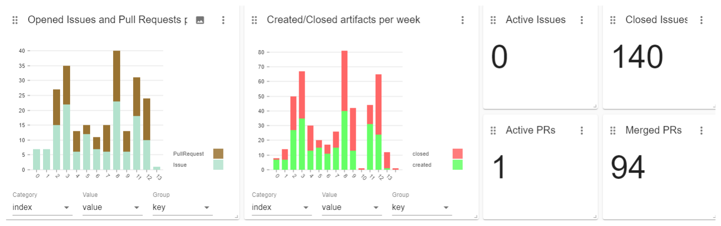


The provided class diagram illustrates the primary six classes of the tool. The $< Graph >$ class is the primary parent class, which stores nodes with their unique IDs and specific type within the class itself. Each graph object has the ability to contain a varying number of $< Node >$ objects. A $< Node >$ object can exist only within a single $< Graph >$ object.

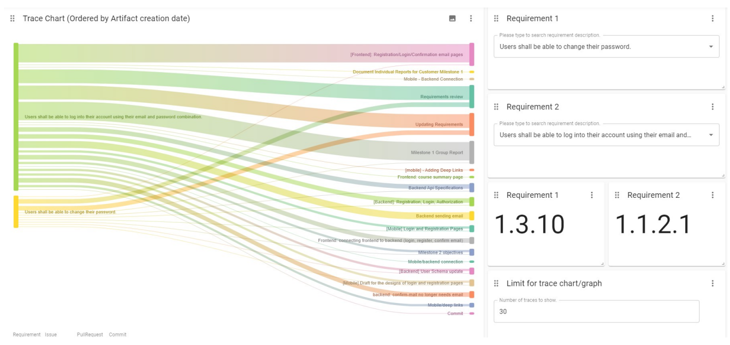
The classes $< Requirement >$, $< Issue >$, and $< Commit >$ are derived from the $< Node >$ class. In addition, the $< PullRequest >$ class is a subclass of the $< Issue >$ class.

6.4. User Interface Design (if applicable)

User Interface Design consists of dashboard where the statistical data of repository are displayed. Down below, there are examples of reports on dashboard.

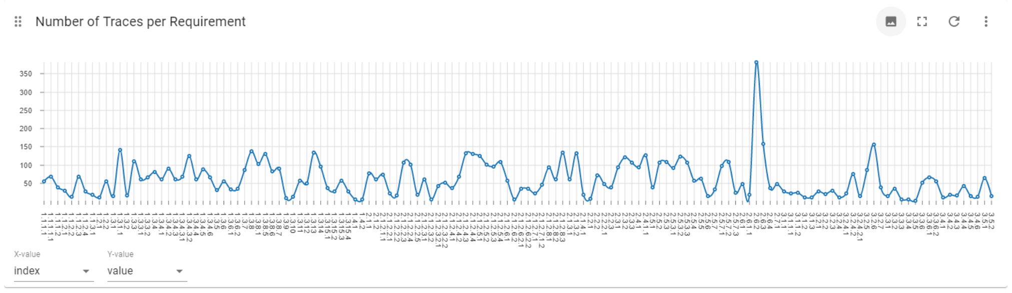


The figure above shows the opened and closed issues and PRs in a bar chart. On left, statistical data about the status of artifacts are displayed.

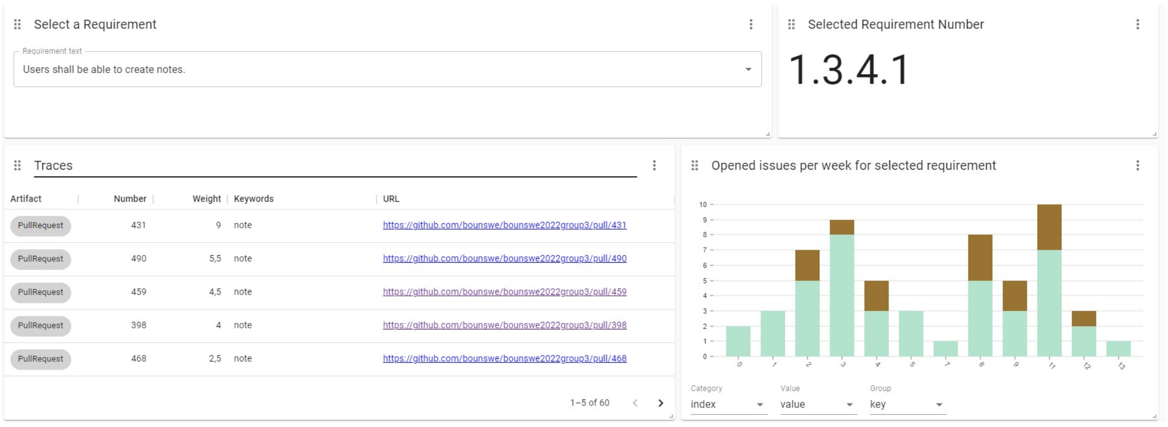


Trace Chart (Ordered by Artifact creation date)
Requirement 1
Users shall be able to change their password.
Requirement 2
Users shall be able to log into their account using their email and...
Requirement 1
1.3.10
Requirement 2
1.1.2.1
Limit for trace chart/graph
30
Requirement
Issue
PullRequest
Commit

The figure above visualizes the traces on sankey chart, where common related artifacts are shown. The thickness of links indicate the weight of keyword match.



The figure above shows the number of traces for per requirement.


Select a Requirement

Requirement text: Users shall be able to create notes.

Selected Requirement Number

1.3.4.1

Traces

Artifact	Number	Weight	Keywords	URL
PullRequest	431	9	note	https://github.com/bounswe/bounswe2022group3/pull/431
PullRequest	490	5,5	note	https://github.com/bounswe/bounswe2022group3/pull/490
PullRequest	459	4,5	note	https://github.com/bounswe/bounswe2022group3/pull/459
PullRequest	398	4	note	https://github.com/bounswe/bounswe2022group3/pull/398
PullRequest	468	2,5	note	https://github.com/bounswe/bounswe2022group3/pull/468

1–5 of 60

Opened issues per week for selected requirement

Category: index
Value: value
Group: key

The figure above shows the traces and their properties for a specific requirement in tabular format.

7. IMPLEMENTATION AND TESTING

7.1. Implementation

The implementation of the project has three major components: Keyword Extraction, Automated Graph Database, and Graph Visualization.

The initial prototype source code is present in GitHub repository.

Automated Graph Database

Graph and Nodes

To be able to build a graph from the software artifacts, having software artifacts as nodes and traces as edges, we need to, - Acquire raw data of software artifacts, - Parse them into node structures, - Build links between artifact nodes

Acquiring data

All necessary data for the construction is present and fetched from GitHub Repositories. Requirements Specification Documents recorded on GitHub wiki pages are manually converted into a .txt file. Issues, pull requests, and commits that are stored in GitHub were obtained using GitHub API.

Due to its user-friendly nature, we opted to utilize the GitHub GraphQL API in lieu of the traditional REST API. Notably, the structured approach offered by GraphQL, which allows us to specify the desired fields to extract from the API, proved highly advantageous for our purposes to fetch data for keyword searches.

Data fields fetched for each artifact are as follows:

$$\text{Issue: } < url >, < title >, < number >, < body >, < createdAt >, < closedAt > \\ < state >, < milestone >, < comments >$$

$$\text{PR: } < url >, < title >, < number >, < body >, < createdAt >, < closedAt > \\ < state >, < milestone >, < comments >, < commits >$$

$$\text{Commit: } < id >, < message >, < url >, < committedDate >$$

Parsing data

By employing GraphQL, parsing operations can be simplified to reading the structured data and generating the required nodes. The parent *Node* class contains $< id >, < text >, and < traces >$ fields, while the child classes (including *issue*, *pr*, *commit*, and *requirement*) have similar structures as their corresponding GraphQL queries. The text field is utilized to gather all text-related content of a node into a single field, facilitating easy search among nodes. For instance, the title, body, and comments of issues are concatenated and stored under the text field.

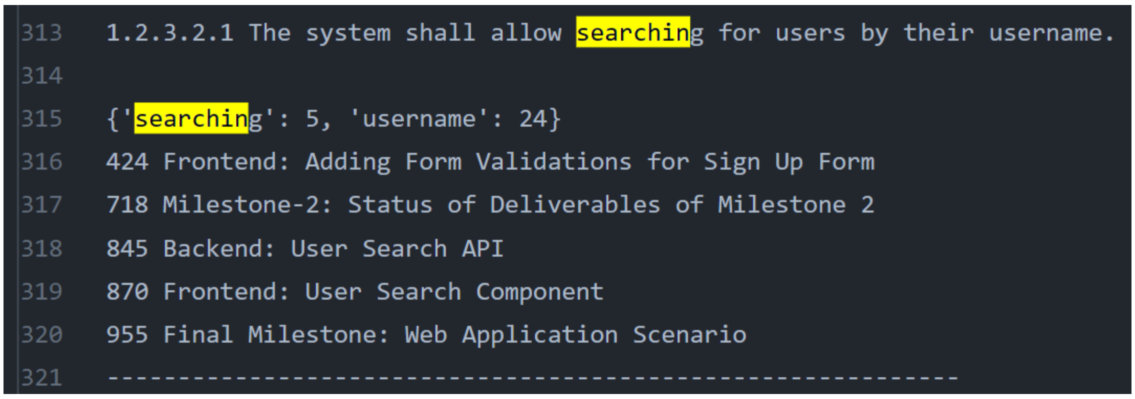
Building trace links

The keyword extraction and parsing results have been combined.

After parsing, node objects with an id and text field were created. The text field of the requirement nodes was then subjected to a keyword extraction process. The resulting list of keywords for each requirement was then matched with existing issues, by searching each keyword in them using RegEx. The nodes of the issues that had matching keywords were saved to a set.

The textual traces obtained are present in traceResults . The textual traces have the template:

$$< \textit{Requirement Number} > - < \textit{RequirementText} >$$
$$\{ \; [< \textit{keyword} >:< \textit{Number of occurences} >] \; \}$$
$$< \textit{Artifact Number} >< \; \textit{Artifact Title} >$$



```
1.2.3.2.1 The system shall allow searching for users by their username.

{'searching': 5, 'username': 24}
424 Frontend: Adding Form Validations for Sign Up Form
718 Milestone-2: Status of Deliverables of Milestone 2
845 Backend: User Search API
870 Frontend: User Search Component
955 Final Milestone: Web Application Scenario
------------------------------------------------------------
```

Figure 7.1. Example trace results for initial prototype

Keyword Extraction

The graph database comprises a Node class with four distinct subclasses as mentioned in the previous section, namely: Requirement, Issue, PR, and Commit. The textual data encapsulated within these nodes can be utilized for trace-searching purposes. Typically, the Requirement node comprises a natural language requirement specification sentence. The principal aim of identifying artifacts associated with a requirement is to execute Natural Language Processing (NLP) techniques and acquire the most relevant keywords that can be utilized to conduct a keyword search on software artifact properties, including title, description, commit message, and comments. Whenever a match is found, it is regarded as a potential trace to the requirement. To achieve this goal, four Python libraries and two external APIs were assessed to determine their efficacy in extracting keywords. These are named API Layer, sPacy, RAKE Nltk, YAKE and Summa and MonkeyLearn. However, the efficiency of these libraries on their own were not sufficient enough for our success criteria, and the noise level was too high for artifact textual data. Therefore, Part-of-Speech tagging(POS tagging) and Dependency Parsing on requirement texts were decided for use, since a custom pipeline implementing the mentioned methods were suitable to extract keywords from requirement texts.

Custom Pipeline

The implemented pipeline requires a Requirement Specification Document (RSD) and an English stopwords file to function. The purpose of the pipeline is to identify the most frequently used words in the RSD and remove the first five words from the extracted keywords, in addition to the English stopwords. This method helps reduce noise in the captured traces, as frequently used words often appear in various textual data.

The next step involves tokenizing the words in each requirement's text and capturing nouns and verbs. This task is relatively straightforward since sPacy, with its powerful libraries and methods, can identify words tagged as "VB" (verbs) or "NN" (nouns), or any combination thereof. The captured noun-verb tokens are stored in a "keyword list" as the initial candidates for extraction. The custom extraction process begins after this step.

The extracted nouns and verbs have dependency parsing links that connect them to their related objects. By following these links, "noun phrases" or "verbal phrases" can be obtained. These phrases significantly improve the accuracy of the requirements. While isolated verbs and nouns may introduce noise, the phrases result in traces that are more specific to the requirements.

Noun and Verb Analysis

In the process described, the verb tokens are examined, and their child tokens are traversed. Specifically, the child tokens that are connected to the parent verb token through the categories of "acomp" (adjectival complement), "dobj" (direct object), "comp" (complement), and "prt" (phrasal verb particle) are combined with the parent token to form a phrase. As an illustration, the verb token "edit" is associated with its direct object token "note" (dobj) to create the phrase "edit note". The parent tokens are then excluded from the list of keywords, while the child tokens are retained.

In the case of noun tokens, a similar procedure is carried out, where specific dependency parsing links are considered valid. These accepted links include "nmod" which denotes a noun modifier, "amod" which denotes an adjectival modifier, and "compound" which denotes a compound modifier. Once noun phrases are identified, both the parent and children tokens associated with them are included in the keyword list.

The subsequent step involves identifying and capturing more complex phrases that are connected by prepositions such as "of," "and," "in," "at," and so on.

The identified noun tokens, noun phrases, and verb phrases are all added to the keyword list, thereby completing the process of keyword extraction.

Results

The evaluation was conducted using the Requirements Specification Document of BUcademy [3], which is publicly accessible on their GitHub Repository. Elaborate findings of the comparative analysis of the keyword extractors are available in the Keyword Extraction section of our Github Repository. Inside the linked directory, the Requirements Specification Document(RSD) is present as *Requirements.txt*. The stopwords include the English stopwords with high-frequency words of RDS, and named *SmartStopwords.txt*. The result of the different extractors are documented with the template *Extractors_***classname***.txt*.

Word-Vector

In the word-vector approach, a document vector is generated by utilizing a pre-trained word-vector model. Word vector for each word in an artifact's text is acquired from the model and averaged, resulting in the document vector. This process is performed uniformly for every artifact.

Following the creation of word vectors, the cosine similarity between the word vector of each software development artifact and the requirement node is computed. A similarity threshold is established, and artifacts exhibiting a cosine similarity surpassing the predefined threshold are identified as potential traces.

TF-IDF Vector

In the tfidf-vector approach, a document vector is generated by tf-idf values of words within an artifact's text. This procedure is consistently applied to every artifact.

Once the tfidf-vectors are generated, the cosine similarity is computed between each requirement node and the word vector associated with the software development artifact. A predetermined similarity threshold is established, and the artifacts exhibiting a cosine similarity above this predefined value are considered as potential traces..

Graph Visualization

Our project focuses on storing requirement trace relationships between software artifacts, specifically Issues and Pull Requests (PRs), using the Neo4j graph database. The motivation behind this choice of database architecture stems from several key factors. Firstly, Neo4j is a highly efficient and scalable graph database that excels in managing complex relationships between data entities.

The database consists of various nodes, namely Requirement, Issue, PR, and Commit. Once the neo4j connection is established, the initial step involves the creation of software artifact nodes. During this process, the textual data associated with the artifacts is merged into a designated "text" field within the nodes, facilitating the capture of traces.

The Requirement nodes possess several properties including "number," "description," "parent" (which indicates the parent requirement in the requirement tree structure), and "text."

On the other hand, the Issue nodes contain properties such as "number," "createdAt" (representing the creation timestamp), "commentList" (a storage for the comments discussed under the Issue), "closedAt" (indicating the closure timestamp), "commentCount," "state," "title," and "URL."

Similarly, the PR nodes exhibit properties such as "number," "createdAt," "commitCount," "commentList" (a storage for the comments discussed under the

PR), "closedAt," "commentCount," "state," "title," and "URL."

An additional Neo4j connection is secured following the creation of artifact nodes in order to create trace relationships. There are two types of relationships in the database. They are named: *tracesTo* and *relatedTo*.

The *tracesTo* relationship, as shown in Figure 7.2., is only present between Requirement and Issue/PR nodes. The relationship possesses two distinct attributes: "weight" and "keywords". The "keywords" attribute represents a list of keywords identified through the process of Keyword extraction. On the other hand, the "weight" attribute indicates the strength or intensity of the identified keywords. Specifically, when a match is found with a "noun" or a "verb", the weight is incremented by 0.5. Similarly, when a match occurs with "noun phrases" or "verb phrases", the weight is incremented by 1. This particular attribute serves the purpose of facilitating visualization.

The *relatedTo* relationship, as shown in Figure 7.3., is only present between PRs and the Commit nodes. The Commit nodes that are related to the PR's are linked to them with this relationship. It has no properties.

Figure 7.4. shows the relationships established within the system provide users with the ability to closely monitor the progression of a requirement during its development phase. By simply clicking on a node, users gain access to a comprehensive display of all relevant properties associated with that node. Additionally, the inclusion of the URL property enables users to conveniently visit the corresponding source code hosted on GitHub. This integration facilitates a seamless and efficient workflow for users seeking to track requirement development and explore further details within the source code repository.

.

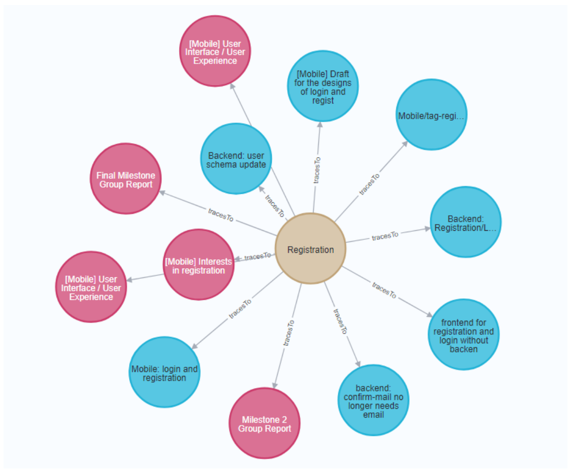

[Mobile] User Interface / User Experience
[Mobile] Draft for the designs of login and regist
Mobile/tag-regi...
Backend: user schema update
Final Milestone Group Report
Backend: Registration/L...
Registration
[Mobile] Interests in registration
[Mobile] User Interface / User Experience
frontend for registration and login without backen
Mobile: login and registration
backend: confirm-mail no longer needs email
Milestone 2 Group Report
tracesTo

Figure 7.2. *tracesTo* relationship between Requirement node(in brown) and Issue node(in pink) and PR node(in blue)

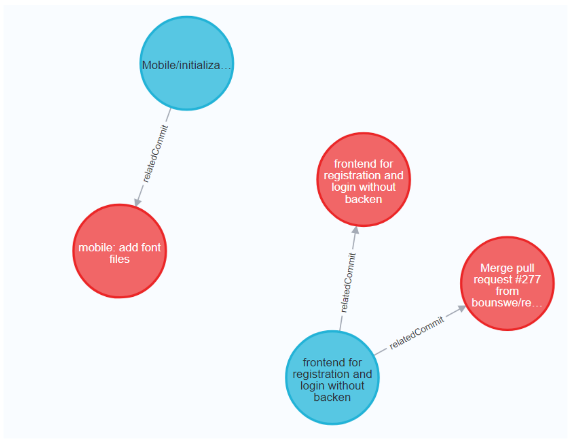

Mobile/initializa...
relatedCommit
mobile: add font files
frontend for registration and login without backen
relatedCommit
Merge pull request #277 from bounswe/re...
relatedCommit
frontend for registration and login without backen

Figure 7.3. *relatedTo* relationship between PR nodes(in blue) and Commit nodes(in red)

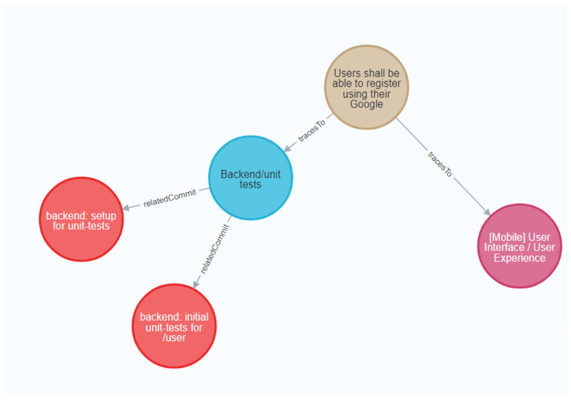

Users shall be able to register using their Google
tracesTo
Backend/unit tests
tracesTo
backend: setup for unit-tests
relatedCommit
relatedCommit
[Mobile] User Interface / User Experience
backend: initial unit-tests for /user

Figure 7.4. Relationships between Requirement node(in brown) and Issue node(in pink) and PR node(in blue) and Commit node(in red)

7.2. Testing

Testing will be documented in further phases of the project.

7.3. Deployment

The project written and tested with Python=3.10.0. It is recommended to use a virtual environment on Python using *venv* library. Clone the project using the following shell command.

```
git clone https://github.com/ersoykadir/Requirement-Traceability-Analysis.git
```

Navigate to the root directory of the project and install the required dependencies.

```
pip install -r requirements.txt
```

Navigate to the root directory of the project and run Docker to activate Neo4j server.

```
docker-compose up -d
```

A file named *.env.example* can be found as a template in the root directory. Create a *.env* file with the following content:

```
GITHUB_USERNAME=<your github username>
GITHUB_TOKEN=<your github token>
NEO4J_PASSWORD="password"
NEO4J_USERNAME ="neo4j"
NEO4J_URI = "bolt://localhost:7687""
```

Navigate to the traceGraph directory under the root.

```
cd ./traceGraph
```

Run *main.py* with two system arguments;

```
python ./main.py <search_method> <options_>
```

$<$ *search_method* $>$ indicates the method used for searching traces.

- keyword: Keyword extraction method for capturing traces.
- word-vector: Word-vector method for capturing traces.
- tf-idf vector: tf-idf vector method for capturing traces.

$<$ *options* $>$

- rt: requirement tree mode,
  - Includes parent requirements for keyword extraction, requires a file
  - Requires a file named 'requirements.txt' in the root directory of the repository
- rg: reset graph
  - Deletes the graph pickle to re-create the graph from scratch

Navigate to http://localhost:7474/ to view results on neo4j.

7.3.1. Examples for Neo4j Queries

The following query returns the traces for a specific requirement number.

```
MATCH p=(r)-[t:tracesTo]->(a)
where r.number='<req_number>'
RETURN *
```

For keyword search, the following query returns the traces which have a specific keyword match.

```
MATCH p=(r)-[t:tracesTo]->(a)
WHERE t.keyword='<keyword>'
RETURN *
```

The following query returns the traces between the requirement and the specific artifact type.

```
MATCH p=(r)-[t:tracesTo]->(a:<artifact_type>)
where r.number='<req_number>'
RETURN *
```

8. RESULTS

Method	**Recall**	**Precision**
Keyword extraction	0.865	0.212

Table 8.1. Recall - Precision Values of Keyword Extraction Method

	Word-vector		**Tf-idf vector**	
Sim.Threshold	**Rec.**	**Prec.**	**Rec.**	**Prec.**
0.05	1.0	0.043	0.839	0.121
0.15	1.0	0.043	0.573	0.256
0.25	1.0	0.043	0.244	0.430
0.35	1.0	0.043	0.095	0.392
0.45	0.965	0.071	0.025	0.125
0.55	0.865	0.100	0.013	0.121
0.65	0.294	0.300	0	0

Table 8.2. Recall - Precision Values of Vector-based Methods

The trace links for a chosen subset of requirements were manually annotated to serve as our ground truth.

Observations:

- The best recall-precision values are obtained with keyword extraction.
- Similarity results are highly dependent on the quality and consistency of how the requirements and and messages associated with software artifacts (i.e. commit messages) have been articulated.

9. CONCLUSION

There are couple of challenges and future work we would like to denote.

Future Work:

- Discovering traceability links between requirements themselves and, from issues to pull requests.
- Integrating different types of Requirement Specifications.
- Implementation of a product that could get artifacts from various platforms such as GitLab, Jira.

Challenges:

- Understanding traceability concept and deciding on what to implement.
- Finding a project to employ requirement traceability. Usually project requirements information is not shared.

We would like to thank our advisors for their invaluable support throughout the entire process.

REFERENCES

1. Gotel, O. and C. Finkelstein, "An analysis of the requirements traceability problem", *International Conference on Requirements Engineering*, 4 1994.

2. Learnify, "Requirements", `https://github.com/bounswe/bounswe2022group2/wiki/Requirements`.

3. BUcademy, "Requirements", `https://github.com/bounswe/bounswe2022group3/wiki/Requirements`.

4. "Manuscript Templates for Conference Proceedings", `https://www.ieee.org/conferences/publishing/templates.html`.

5. Jpsteghoefer, "SST'23 — Software and Systems Traceability – 11th International Workshop at the 31st International Requirements Conference (RE), September 2023", `https://sst23.xitaso.com/`.

6. Naeem, A., Z. Aslam and M. A. Shah, "Analyzing Quality of Software Requirements; A Comparison Study on NLP Tools", *International Conference on Automation and Computing*, 9 2019.

7. Ramesh, B. M., T. Powers, C. W. Stubbs and M. C. Edwards, "Implementing requirements traceability: a case study", *Requirements Engineering*, 3 1995.

8. Pohl, K., "PRO-ART: enabling requirements pre-traceability", *International Conference on Requirements Engineering*, 4 1996.

9. Keenan, Czauderna, Leach, Cleland-Huang, Shin, Moritz, Gethers, Poshyvanyk, Maletic, Hayes, Dekhtyar, Manukian, Hossein and Hearn, "TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions", *International Conference on*

Software Engineering, 1 2012.

10. Gotel, O., J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. I. Maletic and P. Mäder, "Traceability Fundamentals", *Springer eBooks*, pp. 3–22, 10 2011.

11. Hayes, J. H., A. Dekhtyar and J. P. Osborne, "Improving requirements tracing via information retrieval", *IEEE International Conference on Requirements Engineering*, 9 2003.

12. Mills, C., "Automating traceability link recovery through classification", *Foundations of Software Engineering*, 8 2017.

13. Cleland-Huang, J., R. Settimi, E. V. Romanova, B. Berenbach and S. M. Clark, "Best Practices for Automated Traceability", *IEEE Computer*, Vol. 40, No. 6, pp. 27–35, 6 2007.

APPENDIX A: GRAPHS

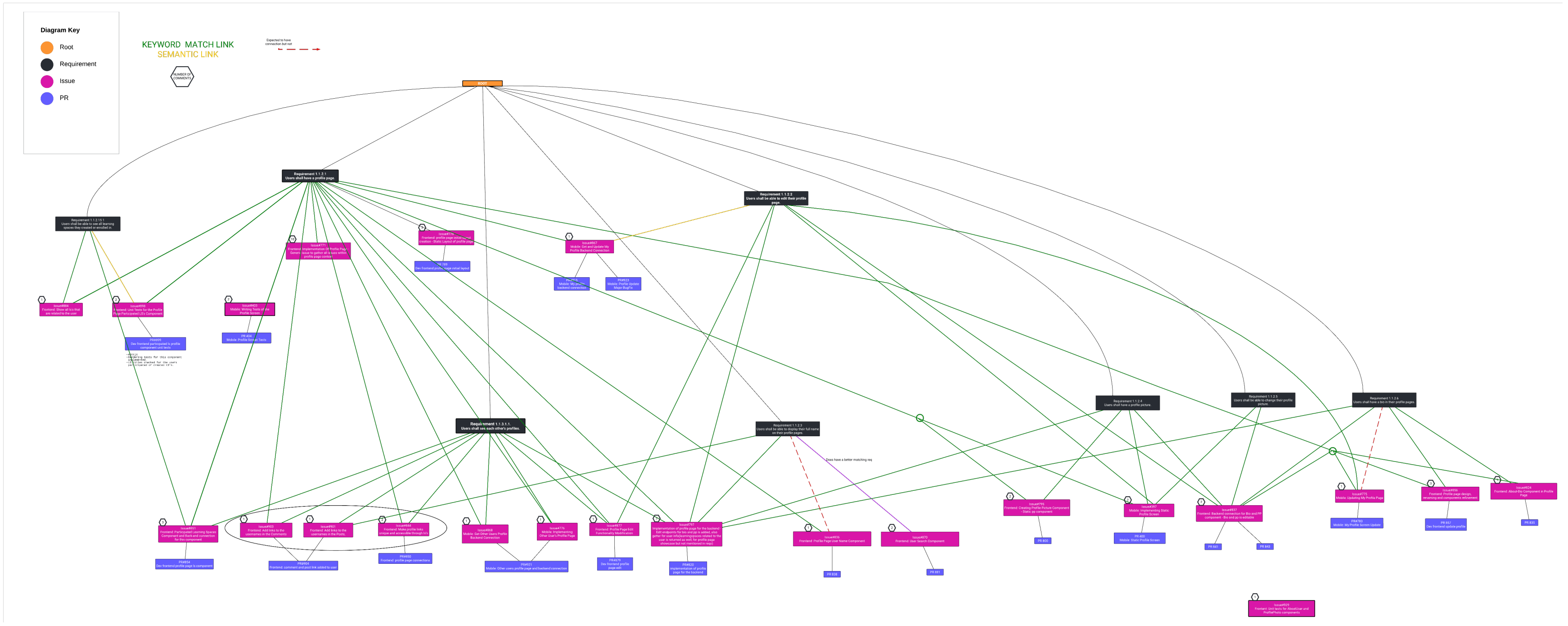


Diagram Key
Root
Requirement
Issue
PR
KEYWORD MATCH LINK
SEMANTIC LINK
NUMBER OF COMMENTS
Expected to have connection but not
Requirement 1.1.3.1.1 Users shall see each other's profiles
Requirement 1.1.2.1 Users shall have a profile page
Requirement 1.1.2.2 Users shall be able to edit their profile page
Does have a better matching req.

Figure A.1. Manually generated trace graph of Learnify repository

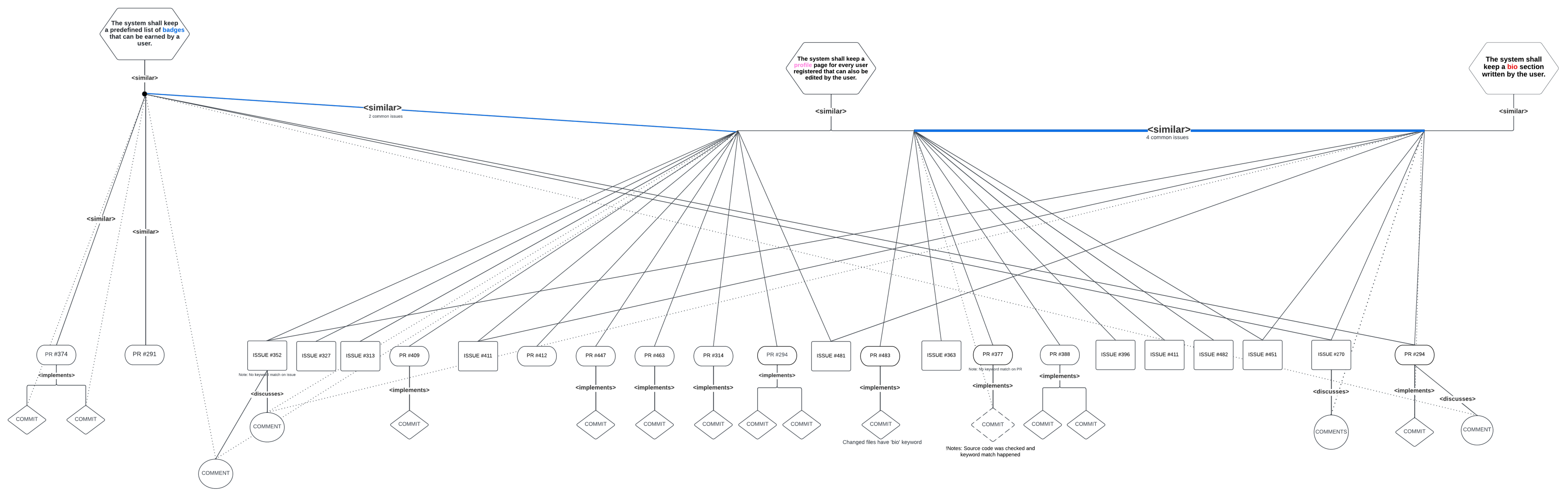

The system shall keep a predefined list of badges that can be earned by a user.
The system shall keep a profile page for every user registered that can also be edited by the user.
The system shall keep a bio section written by the user.
<similar>
<similar>
2 common issues
<similar>
4 common issues
<implements>
<discusses>
PR #374
PR #291
ISSUE #352
ISSUE #327
ISSUE #313
PR #409
ISSUE #411
PR #412
PR #447
PR #463
PR #314
PR #294
ISSUE #481
PR #483
ISSUE #363
PR #377
PR #388
ISSUE #396
ISSUE #411
ISSUE #482
ISSUE #451
ISSUE #270
PR #294
COMMIT
COMMENT
COMMENTS
Note: No keyword match on issue
Note: No keyword match on PR
Changed files have 'bio' keyword
!Notes: Source code was checked and keyword match happened

Figure A.2. Manually generated trace graph of BUcademy repository