CMPE 492

A Software Project Requirement Tracking Analysis Tool

Ecenur Sezer

Kadir Ersoy

Advisor:

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.

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

The present milestone entails two primary objectives. The first objective involves developing and refining the initial prototype to transform it into the final product. To achieve this goal, the trace-building process will be enhanced, and a procedure for graph creation based on the traces will be established. The second objective involves conducting the testing of the tool's functionality, usability, and reliability to determine its efficacy.

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

Success Criteria will be documented in further phases of the project.

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.

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 as-

sociated 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 meta-

data 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

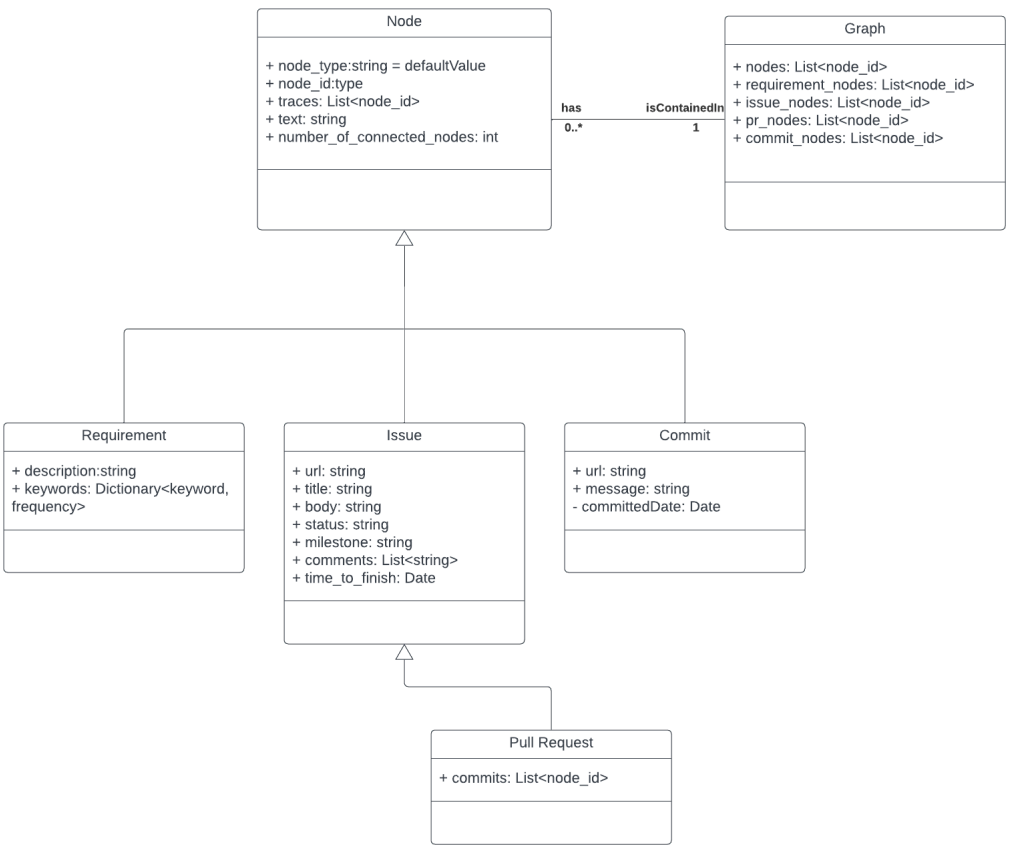
6.1. Information Structure

Information Structure will be documented in further phases of the project.

6.2. Information Flow

Information Flow will be documented in further phases of the project.

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 will be documented in further phases of the project.

7. IMPLEMENTATION AND TESTING

7.1. Implementation

7.1.1. Initial Prototype

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.

Initially, the sets of issue nodes that were found for each keyword were merged to ob-

tain the related issues for a requirement. However, it was observed that unrelated issues were present in the traces, resulting in a high noise level. An algorithm to decrease the noise level was developed. A threshold of 10 was set for the length of the matched issues list. Keywords with more than 10 matching issues were pruned by removing the issues that match only that frequent keyword. This algorithm significantly reduced the noise level.

Nevertheless, some requirements that had extracted weird keywords faced problematic matchings. The success of keyword extraction appeared to be critical. The depth of pruning was increased for keywords that had matched more than 20 by forcing those issues to have at least 2 other existing keywords. However, the desired improvement was not achieved.

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

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

```
313  1.2.3.2.1 The system shall allow searching for users by their username.
314
315  {'searching': 5, 'username': 24}
316  424 Frontend: Adding Form Validations for Sign Up Form
317  718 Milestone-2: Status of Deliverables of Milestone 2
318  845 Backend: User Search API
319  870 Frontend: User Search Component
320  955 Final Milestone: Web Application Scenario
321  ------------------------------------------------------------
```

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.

1) API Layer

The API Layer demonstrated a limited performance with regard to longer sentences. While single-keyword sentences were accurately identified, phrases and sentences with multiple keywords were not, mainly due to the inclusion of English stopwords such as "shall" and "can" in the results. Additionally, the API's free tier limit of 300 requests per month was insufficient for the purposes of our project.

2) sPacy

Spacy is a robust natural language processing tool utilized in industry settings. It features pre-installed word vectors and keyword extractors. One advantageous feature is the ability to group words by their grammatical category, such as "noun" or "verb", which can be useful for selection purposes. In addition, verbs are extracted in their base form, which is essential for implementing the "exact match" strategy.

Furthermore, Spacy allows for the creation of a custom keyword extraction algo-

rithm. However, it should be noted that the tool only returns individual words as results, as it is unable to extract phrases..

3) RAKE Nltk

The Rapid Automatic Keyword Extraction (RAKE) algorithm is a domain-independent method for identifying keywords in a text. This is accomplished by analyzing the frequency of word occurrence and its co-occurrence with other words within the text. One of the primary strengths of RAKE is its ability to extract phrases while also remaining flexible with regard to stopwords.

4) YAKE

YAKE! is an unsupervised, lightweight approach for automatic keyword extraction that utilizes statistical features extracted from individual documents to identify the most significant keywords within the text. The inclusion of a stopwords list in YAKE! produces highly precise results. Moreover, YAKE! permits the display of the "significance score" of each keyword.

5) Summa and MonkeyLearn

While the summa is not flexible for pipelines, MonkeyLearn is not affordable, even though it has useful results.

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*.

Among the six considered, YAKE and RAKE Nltk emerge as potential candidates for keyword extraction.

Both algorithms were used in the development of an initial prototype.

Graph Visualization

The automatically obtained traces will be transferred to a graph data model to visualize the graphical structure for the rest of the term.

7.2. Testing

7.3. Deployment

Deployment diagram, building instructions, docker/kubernetes, readme, system manual and user manual

8. RESULTS

Results will be documented following the completion of the project.

9. CONCLUSION

Conclusion will be documented following the completion of the project.

REFERENCES

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

2. 2, B. G., "Requirements", `https://github.com/bounswe/bounswe2022group2/home`.

3. 3, B. G., "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

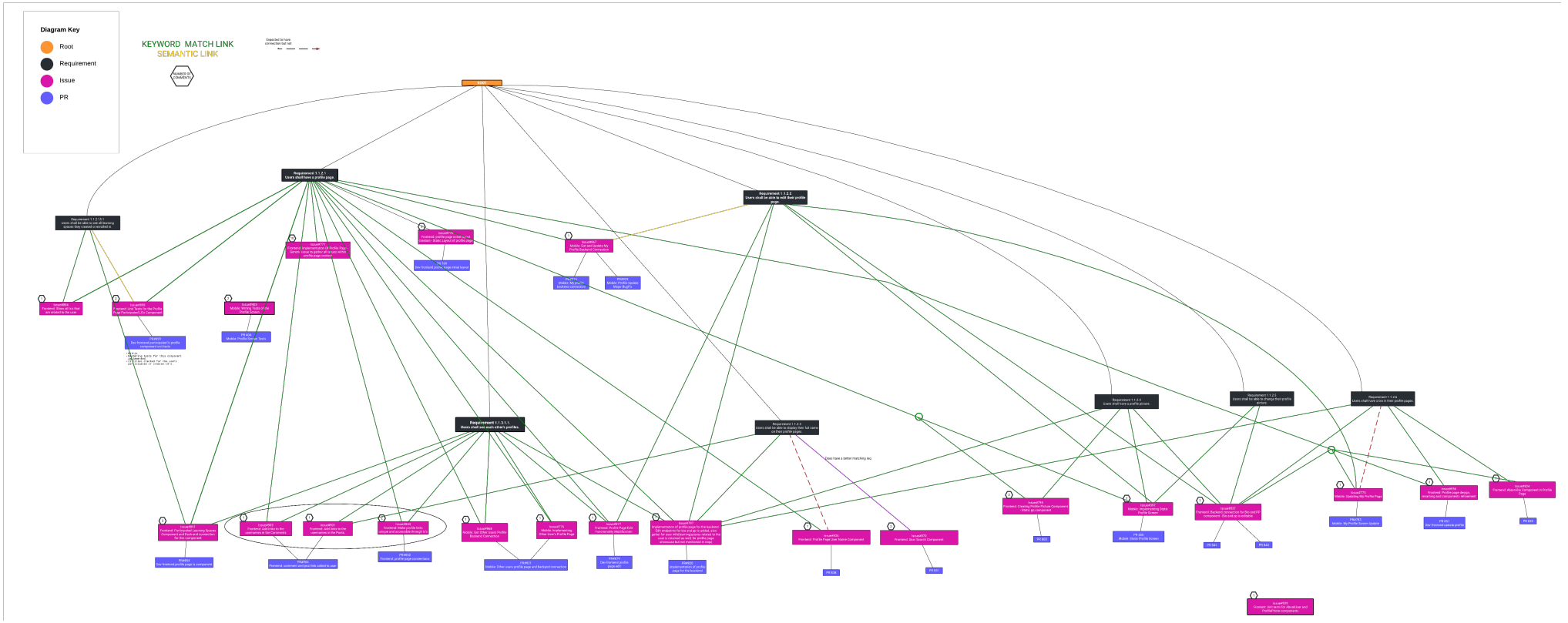


Figure A.1. Manually generated trace graph of Learnify repository

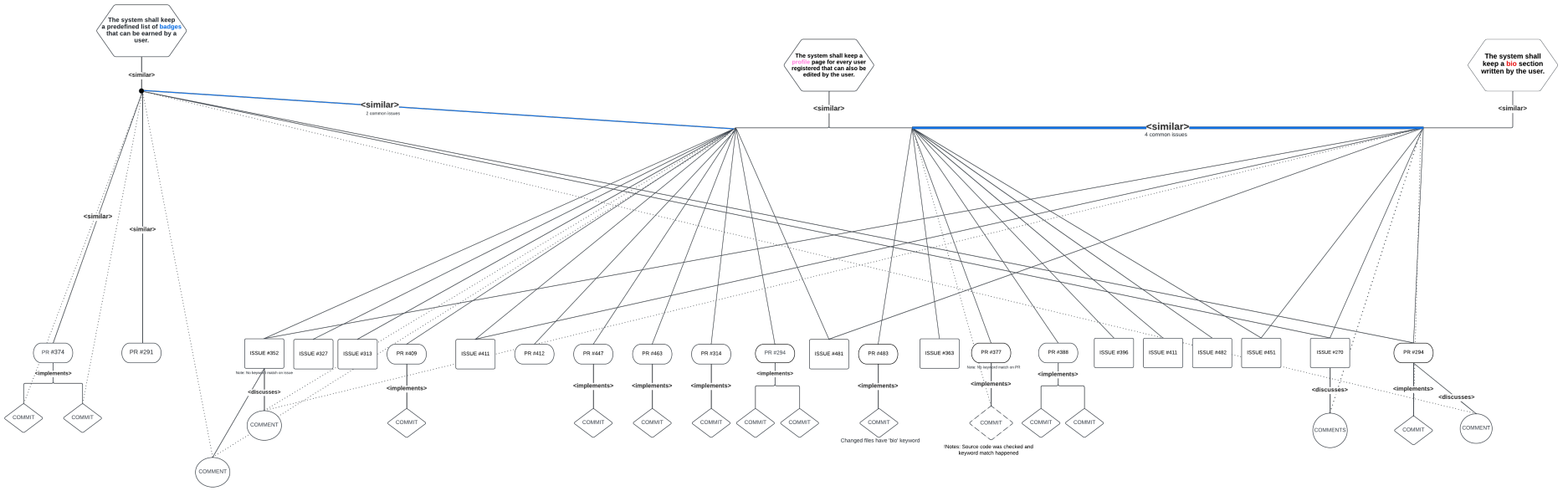


Figure A.2. Manually generated trace graph of BUcademy repository