Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A tool for visualizing software requirements traceability

Bachelor's Thesis

2024

Havriil Pietukhin

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A tool for visualizing software requirements traceability

Bachelor's Thesis

Study Programme: Computer Science
Field of Study: Computer Science
Department: Department of Computer Science
Supervisor: RNDr. Jana Kostičová, PhD.

Bratislava, 2024
Havriil Pietukhin

84378653

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Havriil Oleksiiovych Pietukhin
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: A tool for visualizing software requirements traceability
Nástroj na vizualizáciu sledovateľnosti softvérových požiadaviek

Anotácia: Sledovateľnosť požiadaviek je veľmi dôležitým pojmom v celom procese vývoja softvéru. Veľké množstvo IT projektov zlyhá práve kvôli nesprávnym požiadavkám, ktoré vznikajú pri nedodržaní princípov sledovateľnosti. Tento pojem nie je jednoduché zrozumiteľne vysvetliť študentom informatiky. Cieľom práce je navrhnúť a vyvinúť nástroj, ktorý prehľadne vizualizuje sledovateľnosť softvérových požiadaviek naprieč celým procesom vývoja softvéru a podporí tak výučbu princípov tvorby softvéru na FMFI UK.

Cieľ: - Podporiť výučbu princípov tvorby softvéru na FMFI UK poskytnutím nástroja, ktorý vizualizuje a vysvetľuje koncept sledovateľnosti softvérových požiadaviek
- Umožniť odsledovať pôvod ako aj nasledovníkov špecifickej požiadavky, vrátane nasledovníkov vo forme konkrétnych fragmentov kódu a iných relevantných výstupov procesu vývoja softvéru
- Správne vizualizovať rôzne verzie požiadaviek a s nimi prepojených položiek
- Umožniť vytváranie a validáciu hierarchií požiadaviek pre všetky bežne používané modely / metodiky vývoja softvéru

Literatúra: Ian Sommervile: Software Engineering, 10th edition, 2016
Karl Wiegers, Joy Beatty: Software Requirements, 3rd Edition, 2013
P. Bourque a R.E. Fairley, eds.: Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org

Vedúci: RNDr. Jana Kostičová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 27.10.2023

Dátum schválenia: 30.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

študent vedúci práce

84378653

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Havriil Oleksiiovych Pietukhin
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: A tool for visualizing software requirements traceability

Annotation: Requirements traceability is a very important concept in the entire software development process. A large number of IT projects fail precisely because of incorrect requirements that arise when the principles of traceability are not followed. This concept is not easy to explain clearly to computer science students. The goal of the work is to design and develop a tool that clearly visualizes the traceability of software requirements across the entire software development process and thus supports the teaching of the principles of software design at FMFI UK.

Aim:
- To support the teaching of principles of software design at FMFI UK by providing a tool that clearly visualizes and explains the concept of software requirements traceability
- To enable the user to trace both backward and forward lineage of a specific requirement, including forward lineage to specific code fragments and other outputs of software development process
- To visualize correctly different versions of requirements and linked items
- To enable the user to create and validate requirements hierarchies for all commonly used software development models / methodologies

Literature:
Ian Sommervile: Software Engineering, 10th edition, 2016
Karl Wiegers, Joy Beatty: Software Requirements, 3rd Edition, 2013
P. Bourque and R.E. Fairley, eds.: Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org

Supervisor:	RNDr. Jana Kostičová, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	27.10.2023
Approved:	30.10.2023

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.... Student

.... Supervisor

Acknowledgments: I would like to express my deepest gratitude to RNDr. Jana Kostičová, PhD. for her persistent help and guidance throughout the duration of this thesis. Her insights and expertise were crucial in the successful completion of this work. Additionally, I wish to thank to my family and friends for their support and encouragement, which were instrumental in my academic journey.

Abstrakt

Táto bakalárska práca predstavuje vývoj vizualizačného nástroja pre sledovanie softvérových požiadaviek, ktorý je určený na podporu výučby princípov softvérového inžinierstva na Univerzite Komenského. Nástroj sa zaoberá problémami sledovateľnosti softvérových požiadaviek, čo je kľúčovým faktorom vysokej miery neúspešnosti IT projektov. Zaradením požiadaviek do hierarchie a vytvorením grafového rozhrania nástroj umožňuje používateľom vizuálne sledovať vývoj každej z požiadaviek, čím zlepšuje pochopenie procesu vývoja softvéru a vznikajúcich pritom výstupov. To prispieva k efektívnejšiemu učeniu sa principam vývoja softvéru.

Kľúčové slová: sledovateľnosť softvérových požiadaviek, vizualizačný nástroj, vzdelávanie v oblasti softvérového inžinierstva

Abstract

This bachelor thesis presents the development of a visualization tool for software requirements tracking, which is intended to support the teaching of software engineering principles at Comenius University. The tool addresses the problems of software requirements traceability, which is a key factor in the high failure rate of IT projects. By placing requirements in a hierarchy and creating a graphical interface, the tool allows users to visually track the evolution of each requirement, improving understanding of the software development process and the resulting deliverables. This contributes to more effective learning of software development principles.

Keywords: software requirements traceability, visualization tool, software engineering education

Contents

Introduction

Gathering and compiling requirements is one of the most crucial and time-consuming aspects of the software development process. To successfully complete a project, it is necessary to consider the interests and constraints of all project participants, from customers to testers. Over the decades, the industry has developed standards and methodologies for business analysis and requirements management, which are embodied in a variety of tools for organizing development processes. These tools continually adapt to emerging software development approaches and improve alongside them.

However, as practice shows, the problem of requirements traceability remains a significant issue. It was found in comprehensive study [1] on software project failure factors that 76% of failed projects did not reassess, control or manage risks throughout the project, and 73% of delivery decisions were made without adequate requirements information.

This work aims to develop a visual and flexible tool that will enable FMFI UK students to understand the importance of requirements traceability and familiarize them with various approaches to improve it under different development conditions, such as waterfall and agile methodologies. To achieve this, we have examined a range of existing tools performing similar tasks and proposed our solution. For this purpose, we developed our own model for documenting requirements, design artifacts, code, and tests as a graph with semantic relationships. We have also fully documented the requirements for the application using the application itself. The specification and its glossary can be found in Appendix 1.

Chapter 1

Project background

This chapter lays the groundwork for the project by introducing key terminology, defining the primary goals, and categorizing the tool we are developing. It provides the necessary context for the rest of the thesis.

1.1 Terminology

In this section, we introduce and define the main definitions used throughout this work. These definitions are drawn from the ISO/IEC/IEEE 24765:2017 vocabulary [2], which provides a unified lexicon for all aspects of systems and software engineering.

- **Requirement:** "*a feature, constraint, or other property that a system must provide to satisfy the needs of stakeholders*".
- **Requirement specification (RS):** "*document that specifies the requirements for a system or component*".
- **Requirement traceability (RT):** "*the identification and documentation of the derivation path (upward) and allocation/flow-down path (downward) of requirements in the requirements hierarchy*".

1.2 Primary goals

In this section, we outline the primary objectives we aim to achieve with our tool. Our tool aims to address several key objectives:

- Supporting the teaching of software design principles at FMFI UK by providing a tool that clearly visualizes and explains the concept of software requirements traceability.

- Enabling users to trace both backward and forward lineage of a requirement, including links to specific code fragments and other outputs of the software development process.
- Accurately visualizing different versions of requirements and linked items.
- Allowing users to create and validate requirements hierarchies for all commonly used software development models/methodologies - waterfall, iterative-incremental, and Agile.

Existing applications in our domain do not fully meet our requirements for an educational tool. A more detailed analysis of these solutions is provided in chapter 2. In summary, the development of our application is motivated by several key factors:

- **Cost and Usability:** Commercial tools are often expensive and have a steep learning curve. They come with numerous features, many of which may be unnecessary for our purposes.
- **Graphical Representation:** There is a lack of user-friendly, highly interactive graphical representations in existing tools, which are crucial for our educational objectives.
- **Integration:** Many existing tools suffer from poor integration capabilities with other tools, limiting their effectiveness in a comprehensive requirements management ecosystem.

To address these issues, our application should offer:

- A lightweight, easy-to-use interface that reduces complexity.
- A highly interactive and visually appealing graph representation of requirements. Representing structured information in the form of a graph is the most effective approach for educative purposes [3]. Graph representation offers a visually clear and intuitive way to understand the relationships between requirements and other artifacts.
- High modularity, allowing easy integration with other tools such as version control systems, e.g., git.

By focusing on these key aspects, our tool aims to provide an accessible and effective way to teach and demonstrate the principles of software requirements traceability, while offering the flexibility to integrate with other tools and adapt to various software development methodologies. We have documented the detailed requirements for our application in appendix B. This not only serves as a comprehensive specification but also provides a practical demonstration of our tool's capabilities by creating a tangible artifact that showcases the application's functionality in action.

1.3 Tool categorization and validation of requirements

The article "An Analysis of the Requirements Traceability Problem" [4] validates the relevance of the goals we have set for our software requirements traceability visualization tool. The study, based on extensive research involving practitioners, focus groups, and literature reviews, introduces the concepts of pre-Requirements Specification (pre-RS) and post-Requirements Specification (post-RS) traceability.

The authors' recommendations for establishing pre-RS traceability, such as reliable access to information, historical data versioning, and collaboration capabilities between tools and teams, align with our tool's objectives. Furthermore, the article categorizes requirements engineering tools based on their purpose and architecture, placing our tool in category C - RT (requirements traceability) workbench.

The article also points out some common issues with RT workbench tools. They often force users to work in a strict, top-down manner and rely on fixed classification systems. These tools usually assume that the baseline requirements will not change much, which makes it hard to handle updates. They also tend to focus more on the traceability process itself rather than using it to help create better requirements specifications. Another big problem is that these tools often do not integrate well with other systems. This makes it tough to use them for traceability in the early stages of a project or to easily handle changes later on without a lot of manual work.

By being aware of these limitations, we can ensure that our application addresses these shortcomings and provides a more comprehensive and user-friendly solution for managing software requirements traceability. The analysis presented in the article not only validates our goals but also highlights the significance of the issues we aim to address.

Chapter 2

Existing solutions

Before presenting our methodological findings, we analyze existing tools from our domain that serve similar purposes. We examined various requirement management tools, including both proprietary and open-source options. We focus on three proprietary solutions — IBM Rational, CodeBeamer, and Visure — notable for their functionalities in source code integration and designing requirements hierarchies [5]. We highlight how these applications achieve better requirement traceability.

2.1 Commercial solutions

Here we examine commercially available tools addressing software requirements traceability.

2.1.1 IBM Rational

IBM Rational [6] offers a suite for requirements management in complex software and systems engineering projects. Key features include:

- Traceability linking requirements with various project artifacts.
- Collaboration across geographically dispersed teams.
- Handling of large-scale projects.
- Integration with various software development tools.

2.1.2 CodeBeamer

CodeBeamer [7], an Application Lifecycle Management (ALM) tool, integrates strongly with Software Configuration Management (SCM) systems. It offers:

- Fuzzy parsers for languages like C/C++ and Java.
- Extraction and relation of programming constructs (classes, methods, variables).
- Accommodation of preprocessor directives for enhanced C/C++ parsing.

2.1.3 Visure

Visure [8] also provides enables teams to import source code from repositories into the tool, providing:

- Creation of elements and items for each file or function.
- Facilitation of tracing requirements, tests, defects, and risks associated with specific code parts.
- Support for over forty nine programming languages, including C, C++ and C#.
- Simplification of compliance audits by exporting source code tracing to documentation formats.

2.2 Open-source requirements management tools

In this section we will analyze open-source alternatives to commercial tools. In the domain of open-source requirements management tools, three notable options are rmtoo, Doorstop, and StrictDoc, each representing a unique approach to the integration of requirements into software development processes.

2.2.1 rmtoo

Another observed open-source tool is rmtoo [9], designed primarily for Unix-based systems. It adopts a decentralized approach by storing requirements as text files. This strategy provides easy and reliable approach for creating requirement hierarchies, crucial for effective project management. Predominantly operated via a command-line interface, rmtoo includes features for creating dependency graphs. However, its lack of a GUI and limited compatibility with Windows environments may present hurdles, particularly for users less familiar with command-line tools.

2.2.2 Doorstop

Doorstop [10] model of requriement is similiar to one in rmtoo. It organizes requirements management around the concept of storing textual requirements alongside source code in a version control system. Each textual file in Doorstop represents an individual requirement, forming the basis for a linkable and reviewable set of requirements. The tool employs directories for hierarchical structuring, assigning unique, sequentially numbered identifiers to each requirement. Doorstop's approach, as outlined in its foundational papers [11], emphasizes the importance of integrating requirements directly with source code, although it primarily offers a command-line interface and lacks a comprehensive GUI, potentially limiting its accessibility to users not familiar with specific technical tools like Git and Bash.

2.2.3 StrictDoc

StrictDoc [12] emerges as a successor to Doorstop, maintaining the foundational principle of text-based requirements management but enhancing it with significant advancements. Unlike Doorstop, which stores requirements in YAML files and assumes an implicitly-defined grammar encoded ad-hoc in its parsing and validation rules, StrictDoc adopts a more explicit and flexible approach. It utilizes textX, a tool for creating Domain-Specific Languages, to define its grammar explicitly. This allows StrictDoc to encode a strict, type-safe grammar in a single grammar file, which is then used to parse documentation files. This design decision not only provides more flexibility in defining the structure of requirements but also makes StrictDoc more user-friendly. The addition of a server and a GUI in StrictDoc greatly extends its accessibility, making it a suitable option for a broader audience.

Overall, while rmtoo and Doorstop focus on command-line based, decentralized requirements management, StrictDoc is offering a more flexible, user-friendly interface, and a robust approach to defining requirements structures, thereby broadening its appeal to a wider range of users.

2.3 Challenges with existing tools

Existing tools, both commercial and open-source, present several challenges that hinder their alignment with our primary goals.

Commercial tools

Commercial requirements management tools, while being feature-rich, are not well-suited for academic purposes due to:

- Excessive functionalities that are unnecessary for demonstrative purposes, leading to complexity and a steep learning curve.
- Proprietary nature, making them less suitable for academic environments focused on demonstration rather than project management.

Open-source tools

Open-source tools like rmtoo, Doorstop, and StrictDoc also have challenges:

- **rmtoo:** Despite its capability to create a graph of requirements, rmtoo's display is not user-friendly and lacks comprehensive lineage tracking for each requirement, making it difficult to track fulfillment in the code.
- **Doorstop:** Doorstop lacks a graphical representation of requirements sets and offers limited traceability through source code due to non-integrated changes between requirements and source code. Users must manually map changes, although it does provide versioning of requirements through Version Control Systems (VCS).
- **StrictDoc:** Initially, StrictDoc appeared to be a highly promising solution. In fact, it seemed so suitable that we considered partially using it as a foundation for our application. However, significant implementation issues arose during our development process, which ultimately led us to pursue a different approach. A more detailed discussion of these challenges and our decision-making process can be found in chapter 4.

In practice, keeping software code in sync with regulatory documentation is a significant challenge. Doorstop and StrictDoc offer a solution by keeping code and documentation in Git, but this approach has its own problems. Deciding when to document changes — before, during or after software modifications — and managing draft documentation in Git, where content is either committed or not, presents a complex scenario. These issues highlight the need for a more integrated and user-friendly approach to requirements management, particularly in an academic setting.

Chapter 3

Hierarchy model and application requirements

In this chapter, we present the requirements model or hierarchy we use in our proposed tool and discuss its theoretical groundings. The model draws inspiration from the Business Analysis Body of Knowledge (BABOK) guide [13], a widely recognized standard for business analysis practitioners. The BABOK guide presents a methodology-agnostic hierarchy, making it a versatile framework that is frequently employed in both waterfall and iterative and incremental development approaches. We modified the BABOK model to address the specific needs of our application and to support various software development methodologies, including Agile. Our proposed model offers a flexible and comprehensive framework for managing requirements traceability, addressing the limitations of existing models and incorporating best practices from various development methodologies.

3.1 Existing Requirement Models

This section describes popular hierarchy models commonly used in main software development methodologies - waterfall, iterative and incremental, and Agile. To begin with, let us briefly introduce the most popular software development methodologies. This section describes popular hierarchy models commonly used in main software development methodologies:

- **Waterfall**: In the waterfall methodology, the project is split into sequential stages, each depending on the outputs of the previous one. Requirements are completely specified at the beginning and are not anticipated to change much during development.

- **Iterative and incremental**: This approach involves building the system through repeated cycles, called iterations, and smaller portions, called increments, at a time. Requirements are refined and prioritized throughout the project, enabling more flexibility and adaptability.

- **Agile**: Agile methods focus on collaboration, flexibility, and rapid iteration. Requirements are usually handled through user stories, which are brief, simple descriptions of a feature from the user's perspective. Agile teams operate in short cycles called sprints, continuously delivering functional software and obtaining feedback.

By understanding these models, we provide context for our proposed approach.

3.1.1 Traditional models

Traditional requirement models have been widely used in software development, particularly in waterfall and iterative-incremental methodologies. While there are several traditional models, we'll focus on two prominent examples: the BABOK model and the model proposed by Sommerville [14].

BABOK Requirements Hierarchy

The Business Analysis Body of Knowledge (BABOK) guide offers a comprehensive framework for organizing requirements and specifying the connections between them. It's important to note that the BABOK model is based on previous models and industry standards, synthesizing best practices in the field. BABOK categorizes requirements into four primary types [13]:

- **Business requirements:** These are high-level requirements that answer the question "why?" They describe the goals and outcomes that an organization aims to achieve through changes. Business requirements can encompass the entire enterprise, a specific division, or a particular initiative.

- **Stakeholder requirements:** These requirements detail the needs of various project participants. They serve as a bridge between business requirements and solution requirements, helping to translate general objectives into more specific needs.

- **Solution requirements:** Here we move to a more technical level. These requirements describe what exactly the solution should do to meet stakeholders' needs. BABOK divides them into two subcategories:

  - **Functional requirements:** They define the specific behavior of the system and how it should process information.
  - **Non-functional requirements:** These requirements concern the qualitative characteristics of the solution, such as performance, security, or usability.
- **Transition requirements:** These are temporary requirements that are only relevant during the transition period from the current state to the target state. They may include aspects of data migration, staff training, or ensuring business continuity during implementation.

BABOK also defines several types of relation between requirements:

- **Derive:** This relationship shows that one requirement logically follows from another. For example, a specific solution requirement might be derived from a more general business requirement.
- **Depend:** Here, one requirement depends on another. BABOK distinguishes two types of dependencies:
  - **Necessity:** When implementing one requirement only makes sense if another is also implemented.
  - **Efficiency:** When fulfilling one requirement simplifies the implementation of another.
- **Satisfy:** This relationship indicates how a specific solution element meets a certain requirement. For instance, how a system's functionality implements a functional requirement.
- **Validate:** The connection between a requirement and a method of verifying it, such as a test scenario that confirms the fulfillment of the requirement.

These relationships help establish traceability between requirements and other project elements, enabling better understanding, impact analysis, and management of requirements throughout the project lifecycle.

Model proposed by Sommerville

Sommerville [14] presents a generic requirements model similar to BABOK, but it identifies only two lower levels of requirements — user requirements and system requirements. The key difference between BABOK and Sommerville's model lies in their perspective. Sommerville's model does not include the highest level (business requirements) found in BABOK. This difference reflects a broader debate in the field about the boundaries of the software development life cycle (SDLC).

Application in waterfall and iterative-incremental development

These traditional models are commonly used in both Waterfall and Iterative-incremental development methodologies. In waterfall development, the clear categorization of requirements and well-defined relationships provided in BABOK model align well with the structured, sequential nature of the methodology. Each phase of the project is completed before moving on to the next, making the comprehensive hierarchy of the BABOK model particularly useful. Iterative-incremental development can be viewed as a series of small waterfalls, and thus can utilize all the models applicable to waterfall. However, this approach introduces an additional dimension of iterations or versions. To support this, we introduce historical versions of the requirements hierarchy in our model. This allows for tracking changes and maintaining a record of the requirements' evolution throughout the development process, accommodating the dynamic nature of requirements in iterative-incremental methodologies.

3.1.2 Agile models

In Agile development, there are several approaches to documenting and managing requirements. We'll focus on two prominent models: the Scaled Agile Framework (SAFe) and the Traceability Information Model (TIM).

SAFe / Big Picture model

The Scaled Agile Framework (SAFe) [15] is a popular framework for scaling Agile practices to enterprise-level software development. SAFe is similar to the Big Picture of Agile Requirements [16], as it organizes requirements at multiple levels — portfolio, program, and team. SAFe's requirements model includes:

- **Epics:** large, high-level requirements that span multiple releases and are broken down into features.
- **Features:** more detailed requirements that can be delivered within a single release and are further broken down into stories.
- **Stories:** small, manageable requirements that can be completed within a single iteration.

This hierarchical structure supports requirements traceability and helps ensure alignment between the work being done at the team level and the overall goals of the organization.

Traceability Information Model (TIM)

Rempel and Mäder [17] proposed a traceability information model (TIM) for Agile requirements management. This model conceptualizes traceable artifacts and trace links within the context of Agile software development. The TIM provides a more granular and detailed representation of the artifacts and their relationships. However, the TIM has a limitation: it considers source code as a single high-level artifact. This can lead to a loss of forward traceability, potentially creating difficulties in managing risks and change impact due to requirement changes, especially in large projects.

Application in Agile Methodologies

Both SAFe and TIM aim to address the challenges of requirements traceability in Agile environments, albeit with different approaches:

- SAFe focuses on organizing requirements at multiple levels and emphasizes traceability between these levels.

- TIM provides a detailed conceptualization of traceable artifacts and trace links within Agile software development.

In our proposed model, we aim to incorporate elements from both SAFe and the TIM, while addressing the limitations of the TIM regarding source code traceability. By combining the hierarchical structure of SAFe with the detailed artifact representation of the TIM and extending it to include more granular source code traceability, we strive to create a comprehensive and effective requirements traceability model for Agile environments. This approach allows us to leverage the strengths of both models while mitigating their individual limitations, providing a more robust solution for requirements management in Agile methodologies.

3.2 Proposed model

Building on our review of existing models, we present our proposed model for software requirements traceability. We explain the model's components, relationships, and advantages in detail.

Key features of our proposed model

Considering the practices described above, we have updated the BABOK [13] model to achieve our goals. The changes include:

- Providing the possibility to define custom levels of requirements. Elements of other types will also reside on their semantic levels (code level, test level, design level).
- Allowing users to add items at sublevels within each level.
- Adding the capability to create historical versions of the graph, enhancing pre-RS traceability. This feature allows us to track changes and maintain a comprehensive history of requirement states over time. By capturing historical versions, we ensure that the context and evolution of requirements are preserved and easily accessible.

These modifications to the BABOK model allow us to utilize various frameworks for mapping requirements in different environments and provide traceability for every item throughout the entire development lifecycle. Our proposed model incorporates elements from both SAFe [15] and the TIM [17], while addressing the limitations of the TIM regarding source code traceability. We combine the hierarchical structure of SAFe with the detailed artifact representation of the TIM and extend it to include more granular source code traceability. This results in a comprehensive and effective requirements traceability model that is adaptable to various development methodologies. By generalizing the requirements hierarchy, our proposed model covers all the mentioned existing models. This generalization allows our model to be compatible with various frameworks for mapping requirements in different environments, providing a unified approach to requirements traceability across development methodologies. The key advantages of our proposed model include:

- Flexibility in defining custom levels of requirements and adding items at sublevels, allowing the model to adapt to different project structures and methodologies.
- Historical versioning of the requirements graph, enabling better pre-RS traceability, change management, and impact analysis.
- Compatibility with various frameworks for mapping requirements in different environments, providing a unified approach to requirements traceability across development methodologies.

- The ability to demonstrate requirements traceability for different software development models to students in an educational application. This is a crucial advantage that stems from the model's generalization, allowing for comprehensive teaching of traceability concepts across various software development approaches.

The resulting diagram describing relationships between items in our traceability hierarchy is provided below as fig. 5.1. We have created the diagram using the UML [18] class diagram format, which is one of the most commonly used methods for representing such hierarchies and relationships. Our proposed model offers a flexible and comprehensive framework for managing requirements traceability, addressing the limitations of existing models and incorporating best practices from various development methodologies. It serves as an effective educational tool, enabling students to understand and apply traceability concepts across different software development paradigms.

Chapter 4

Development of the application

In this chapter, we dive into the technical aspects of implementing our proposed tool. We discuss key design decisions, development challenges, and the overall architecture.

4.1 Failed attempt to extend StrictDoc

Initially, we planned to leverage the StrictDoc for requirements documentation. StrictDoc is based on the Hotwire architecture, where the server delivers ready-made HTML templates to the client in fillable form. This approach relies on server-side HTML generation, which we initially thought would provide a solid foundation for our requirements management tool.

4.2 Challenges with StrictDoc

However, StrictDoc posed several challenges that ultimately led us to abandon this approach:

1. Lack of customization options: StrictDoc's rigid structure limited our ability to tailor the tool to our specific needs.

2. Integration difficulties: We found it impossible to integrate StrictDoc with our desired frontend framework, Angular. The architectural differences between StrictDoc's server-side rendering approach and Angular's client-side rendering made them incompatible.

3. Limited support for visualizing traceability: StrictDoc's capabilities for creating and manipulating interactive requirement graphs were insufficient for our project's needs.

These issues prompted us to change our strategy and seek an alternative solution. After careful consideration, we decided to develop a web application from scratch using another toolset.

4.3 Chosen toolset

Angular [19] was chosen for the frontend and StrictDoc for the backend. Initially, it was envisioned that the client (in Angular) would interact with the server side (StrictDoc), providing users with a flexible and dynamic interface to display data.

However, integrating Angular with StrictDoc proved impossible due to architectural differences. StrictDoc is based on Hotwire architecture [20], where the server delivers ready-made HTML templates to the client in fillable form. This approach relies on server-side HTML generation, which did not meet the requirements of Angular. Angular requires a more flexible and dynamic system for displaying data, where interface updates happen on the client side using JavaScript and DOM manipulation.

Thus, the difference in approaches to content generation and display between StrictDoc and Angular made it impossible to integrate them effectively, which led to the need to revise the architecture and choose an alternative backend.

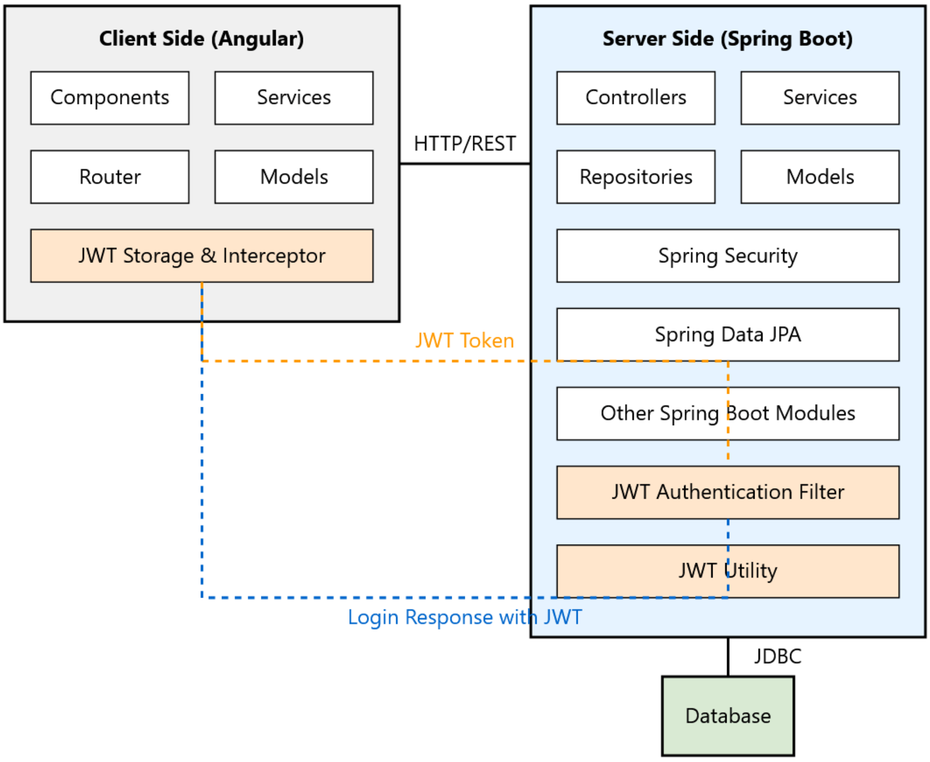
After an unsuccessful attempt to integrate Angular with StrictDoc, it was decided to create a new server side based on Spring Boot [21]. The framework was chosen for its ease of customization and rich set of features.

Compared to StrictDoc, Spring Boot supports the creation of a RESTful API, which is ideal for interacting with the client in Angular. This allowed the server side to provide data in JSON format, which can be easily processed on the client side.

4.4 Data layer

For data storage, it was decided to use Postgres relational database, which provided reliable data storage and management. Postgres supports the JSON data type, which proved convenient for storing dynamic data. This allowed us to create a universal entity of an item in a graph, thereby simplifying the application. JPA was used to manage the data and interact with the database.

Basic architecture diagram is attached below:

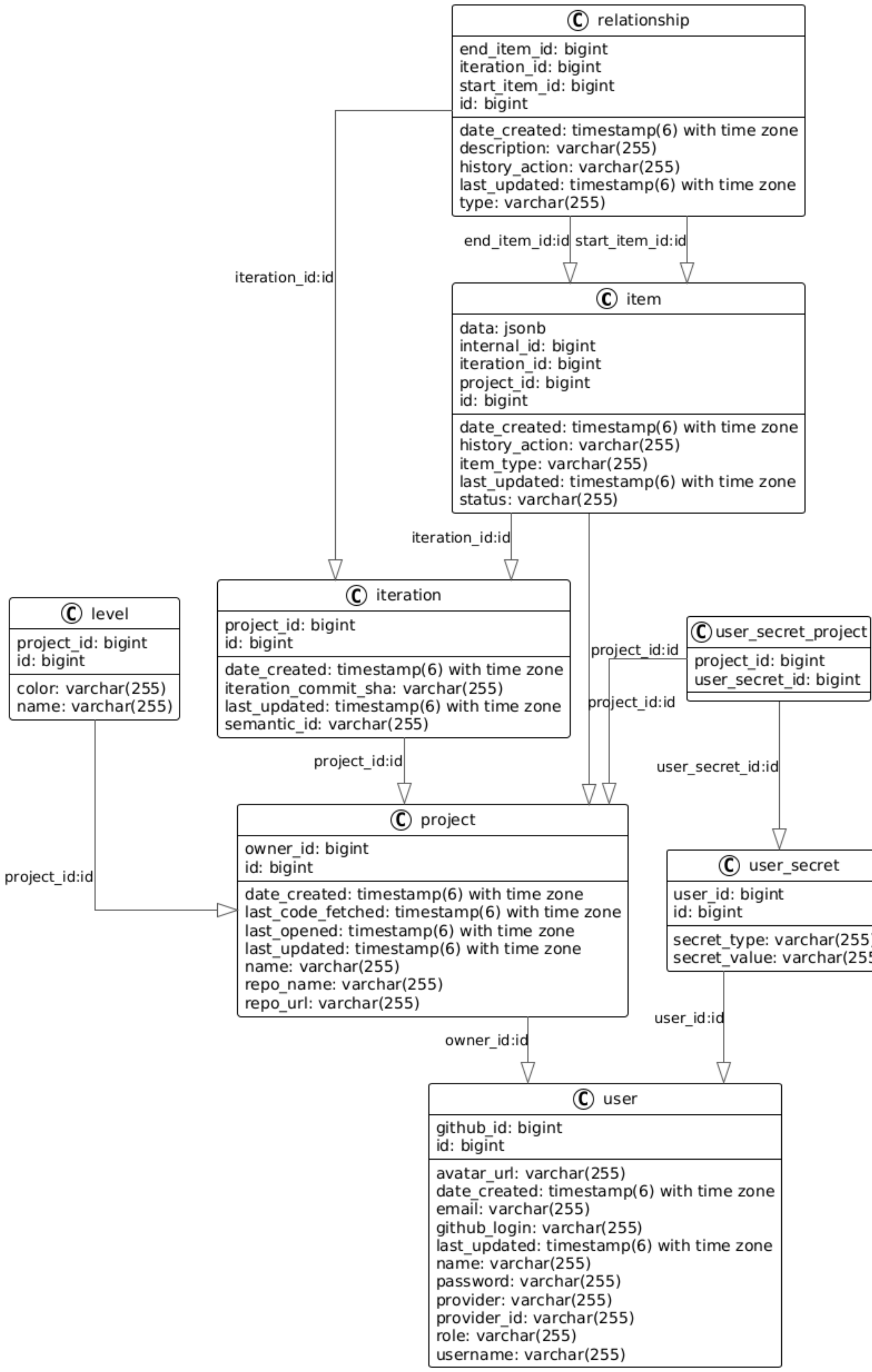


We will briefly discuss important points regarding models for entities within our app:

- JSON support in Postgres: Thanks to the capabilities of Postgres, a data schema has been proposed that allows you to create a graph whose vertices can be data of any type. Each vertex is represented by an Item entity, which has a JSON attribute data to store dynamic information.
- Relationships between entities: Relationships in the graph are represented by the Relationship entity. This entity carries a payload in the form of a type attribute that defines the nature of the relationship, which adds a semantic layer to the data structure.
- Project Management: The main task of the application is to manage software requirements within projects represented by the Project entity. Projects interact with requirements and source code, and the key element is the traceability graph. This graph includes several types of nodes: requirements, source code, design elements, and tests.
- Iteration Management: To ensure traceability of changes in the graph, the development of the graph is captured in the form of Iteration entities. The Iteration entity captures the state of the project at a particular

point in time, including casts of the Item and Relationship entities with attributes indicating historical changes.

The full entity-relation diagram is attached below.



4.5 Integration capabilities

The application provides rich possibilities for integrating third-party tools. Integration of our application with version control systems is realized through the use of the GitHub API. When creating a project, users set a personal access token that gives the application access to their GitHub repository. This allows our application to interact with the code storage infrastructure, providing robust change management and monitoring.

The modularity of the application is achieved by clearly separating functional components, each performing a different task. This approach simplifies the application architecture, making it more flexible and easy to maintain and develop. Using the GitHub API provides many possibilities for working with code, such as cloning repositories, creating commits, managing branches, etc. In the future, thanks to its modular architecture, we can easily extend the application's functionality by adding support for other version control systems such as GitLab or Bitbucket. It is also possible to connect to any other servers storing code, which will require additional effort but will not break the overall architecture of the system.

4.6 Securing the user data

The security of our application is ensured through several key mechanisms. One of them is the use of JSON Web Tokens (JWT) for user authentication and authorization. After successful authentication, the server generates a token that contains information about the user and his access rights. This token is passed to the client and is used to authenticate all subsequent requests. Every request from the authorized user to the server includes this token, thus avoiding the need for multiple authentication and ensuring secure communication between the client and the server.

The second aspect of security is related to storing personal access tokens for third-party services such as the GitHub API. We store them in our database in encrypted form using the Jasypt library. A more reliable way to do this can involve the usage of some secure cloud storage for user secrets, like Azure Key Vault.

4.7 User testing possibilities

We also provided a database script that runs when the application is set up. This script creates a test user and a project with the requirements for our application. By doing this, we documented our own requirements using the tool we built. This makes sure the requirements are always up-to-date and easy to find in the application. Having the requirements already in the database also makes testing a lot easier. Including our own requirements in the database also shows how the tool can be used to document and manage requirements effectively. It gives users a real example of application usage.

4.8 Enhancing user experience

The user in our application will be working on data that has a hierarchical structure and generates a large number of internal relationships. In addition, our application should implement a number of multi-step user scenarios. In such circumstances, the system can easily get into a situation where the user is working with invalid (non-consistent) data. For this purpose, a number of validations have been introduced, affecting work with graph elements and links and work with creating historical versions of the data. A full description of validations is provided in the requirements specification.

4.9 Enhancing developer Experience

Our application leverages the OpenAPI specification for the entire API, providing a comprehensive and machine-readable description of the RESTful API. We utilize the Orval library, which generates TypeScript DTOs (Data Transfer Objects) and services based on the OpenAPI specification. This automation saves development time and reduces errors that can occur when manually creating DTOs and services. Changes to the data model can be quickly propagated throughout the application by running the Orval generation process, ensuring the client stays in sync with the server-side API. The generated DTOs provide type safety and autocompletion in the IDE, catching potential type-related errors early in the development process. By leveraging OpenAPI and Orval, our application achieves a more efficient and error-resistant development workflow, allowing developers to focus on implementing business logic and user interface.

4.10 Deployment strategy

Our application utilizes a modern deployment strategy to ensure scalability, ease of maintenance, and rapid deployment:

- Backend deployment: The backend, consisting of the server and database, is containerized using Docker. This approach ensures consistency across different environments, simplifies the deployment process, and allows for easy scaling. Docker containers encapsulate the entire runtime environment, including the application, its dependencies, libraries, and configuration files.

- Frontend deployment: The frontend is deployed using Vercel, a cloud platform for static and serverless deployment. Vercel provides a seamless deployment experience for Angular applications, offering features like automatic HTTPS, custom domains, and instant global deployment. This approach allows for rapid updates and ensures high performance and availability of the frontend.

This deployment strategy combines the robustness of containerization for the backend with the simplicity and performance optimizations of a specialized frontend hosting platform, resulting in a flexible and efficient deployment pipeline.

4.11 Outcome and Reflection

The development process faced several challenges, including the initial attempt to integrate with StrictDoc, which proved unsuccessful due to architectural differences. However, the decision to develop a new web application from scratch using modern technologies like Spring Boot, Angular, Docker, and rete-js allowed for a more flexible and customizable solution.

While the development process encountered obstacles, the insights gained and the resulting application lay a solid foundation for future improvements and extensions. The modular architecture and the chosen tech stack provide the flexibility needed to adapt to evolving requirements and integrate with additional tools and systems.

Chapter 5

Application Features and Capabilities

Accomplished goals

This project aimed to enhance the understanding of requirements traceability among FMFI UK students. Significant time was spent implementing the client side due to the complexity of the visualization library and its poorly documented features, which presented several challenges. Despite these difficulties, the following objectives were successfully met:

- **Basic user authentication system:** We developed a fundamental system for user authentication that ensures security and controls access.
- **Visualization of hierarchies:** The application provides a graphical representation of hierarchical relationships, which are presented in a format that is accessible and easy to understand.
- **Interactive manipulation of hierarchies:** Users have the ability to dynamically interact with the hierarchical structure, enabling them to add, remove, and modify elements as needed.
- **Export functionality:** The application includes an export feature that allows users to extract data from the hierarchical elements for further use or analysis.
- **Self-documentation of application:** The application documents its own requirements using the implemented model, although documentation of code and tests was not completed due to time constraints.

- **Achievement of backward and forward traceability:** The system successfully implemented both backward and forward traceability, allowing for comprehensive tracking of data provenance and dependencies within the project framework.

Tasks not completed

Not all functionalities planned for the initial release were implemented:

- **Comprehensive documentation:** The application did not achieve complete documentation of code hierarchies and testing procedures, which was a key goal for demonstrating the application's documentation capabilities.

- **Historical version viewing:** The application failed to develop capabilities for viewing historical versions of the project iterations, which would have allowed for better tracking and understanding of the project's evolution.

- **Partial validation operations:** Not all validation operations over the hierarchy elements were developed. We managed to implement only the cycle creation check when adding relationships.

Infrastructure readiness

While not all parts of the application are fully operational, the server infrastructure for the unfinished sections is ready and only needs to be integrated with the client-side components.

Limiting factors

The project faced several challenges that impacted the completion of all planned tasks:

- **Time restrictions:** The limited time available adversely affected the implementation of all intended functionalities.

- **Underestimation of complexity:** The complexity of integrating various development methodologies was initially underestimated, leading to project delays and task reprioritization.

- **Infrastructure dependencies:** Significant time was spent trying to utilize the infrastructure of another application, which was ultimately not as effective or economical as anticipated.

Future work recommendations

For future phases of the project, the following is recommended:

- **Incorporating remaining features:** Focus on integrating the functionalities that were not completed in this phase, particularly those related to documentation.
- **Visual display of iteration histories:** Further develop the functionality to visually depict the development history of projects in the form of iterations, which was not implemented in the initial phase but is critical for tracking the evolution of project requirements.

Conclusion

In this work, we endeavored to create a new visualization tool that organizes software development artifacts—such as requirements, code artifacts, tests, and other items—into a clear and user-friendly hierarchical format. This tool not only illustrates the application of some requirements traceability techniques but also demonstrates its utility in teaching software engineering principles. Despite facing challenges and not meeting all planned objectives, the project has laid a solid foundation for future development. The completed work underscores the tool's potential to significantly enhance the educational process by providing effective means for understanding and managing software development processes. The effort and progress made thus far suggest a promising trajectory towards achieving the project's ultimate educational goals.

Bibliography

[1] June Verner, Jennifer Sampson, and Narciso Cerpa. What factors lead to software project failure? In *2008 Second International Conference on Research Challenges in Information Science*, pages 71–80, 2008.

[2] ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, 2017.

[3] Jana Katreniaková. *Presentation of the Content Structure for E-learning*. PhD thesis, Comenius University in Bratislava, 2006.

[4] O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, 1994.

[5] Olusola Olufemi Oduko. Comparison of requirement management software. 2021.

[6] `https://www.ibm.com/docs/en/rsm/7.5.0?topic=modeler-introduction-rational-products`. Accessed: 02.11.2023.

[7] codeBeamer ALM. `https://codebeamer.com/cb/login.spr`. Accessed: 02.11.2023.

[8] Visure Solutions, Inc. *Source Code Parser User Manual*, 2022. `https://visuresolutions.com`.

[9] rmtoo. `https://rmtoo.florath.net`. Accessed: 13.11.2023.

[10] Doorstop. `https://doorstop.readthedocs.io/en/latest/`. Accessed: 04.11.2023.

[11] J. Browning and R. Adams. Doorstop: Text-based requirements management using version control. *Journal of Software Engineering and Applications*, 7:187–194, 2014.

[12] StrictDoc. https://strictdoc.readthedocs.io/en/stable/strictdoc_01_user_guide.html. Accessed: 11.11.2023.

[13] *BABOK: A Guide to the Business Analysis Body of Knowledge*. Number v. 3. International Institute of Business Analysis, 2015.

[14] I. Sommerville. *Software Engineering*. Always learning. Pearson, 2016.

[15] Scaled Agile, Inc. SAFE 6.0. https://scaledagileframework.com/, 2024. Accessed: 14.05.2024.

[16] Dean Leffingwell. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley Professional, 1st edition, 2011.

[17] Patrick Rempel and Patrick Mäder. Estimating the implementation risk of requirements in agile software development projects with traceability metrics. In Samuel A. Fricker and Kurt Schneider, editors, *Requirements Engineering: Foundation for Software Quality*, pages 81–97, Cham, 2015. Springer International Publishing.

[18] Information technology — open distributed processing — unified modeling language (UML) version 1.4.2. Standard, International Organization for Standardization, Geneva, Switzerland, 2005. Available at: https://www.iso.org/standard/32620.html.

[19] Nilesh Jain, Ashok Bhansali, and Deepak Mehta. AngularJS: A modern MVC framework in JavaScript. *Journal of Global Research in Computer Science*, 5(12):17–23, 2014.

[20] Hotwire: HTML Over The Wire. https://hotwired.dev/. Accessed: 11.04.2024.

[21] Spring Boot. https://spring.io/projects/spring-boot/. Accessed: 27.04.2024.

List of Figures

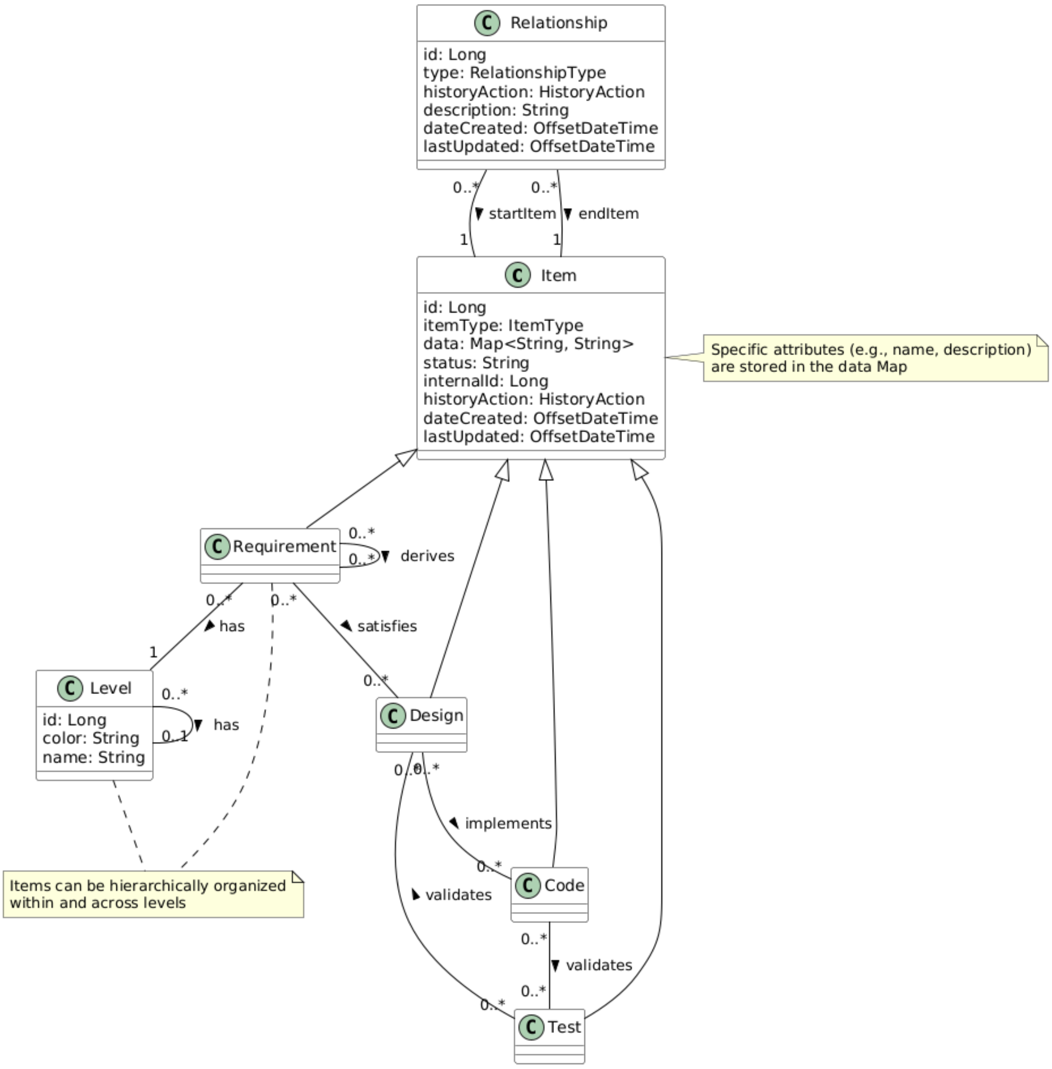


Figure 5.1: Hierarchy Diagram

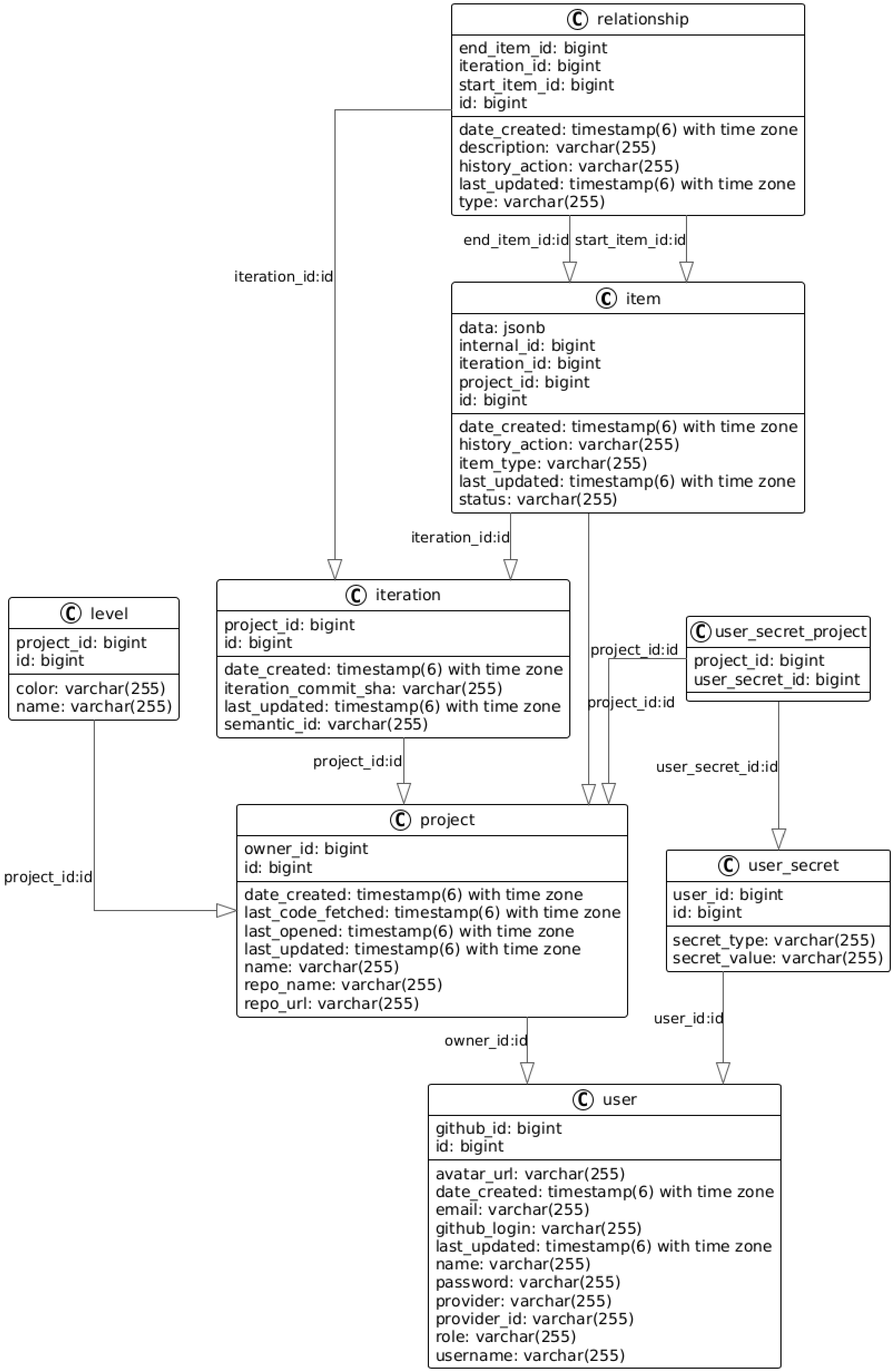


Figure 5.2: Entity-Relationship Diagram

Appendix A

App source code

Source code of our app is fully hosted on GitHub: `https://github.com/havr-p/traceability-tutor`

Appendix B

Application requirements specification

B.1 Requirements

- **REQ-001**
  Level: Business
  Name: Quality Education on Software Design Principles
  Statement: The system shall support the educational objectives by providing examples of software design principles.
  Status: OPEN
  Relationships: REQ-006 (DERIVES), REQ-002 (DERIVES)

- **REQ-002**
  Level: Business
  Name: Quality Education on Software Analysis
  Statement: The system shall include features that support teaching software analysis.
  Status: OPEN
  Relationships: REQ-001 (DERIVES), REQ-003 (DERIVES)

- **REQ-003**
  Level: Business
  Name: Quality Education on Requirements Traceability
  Statement: The system shall include features that support teaching requirements traceability.
  Status: OPEN
  Relationships: REQ-002 (DERIVES), REQ-004 (DERIVES)

- **REQ-004**

Level: Business
Name: Visual Examples of Requirements Traceability
Statement: The system shall include visual representations of real-world requirements traceability examples.
Status: OPEN
Relationships: REQ-003 (DERIVES), REQ-005 (DERIVES)

- **REQ-005**
  Level: Business
  Name: User-Friendly Tool for Requirements Traceability
  Statement: The system shall be user-friendly and visually demonstrate the principles of requirements traceability.
  Status: OPEN
  Relationships: REQ-004 (DERIVES), REQ-011 (DERIVES)

- **REQ-006**
  Level: Stakeholder
  Name: Support Common SW Development Models
  Statement: The system shall support various software development methodologies, including waterfall, iterative-incremental, and agile.
  Status: OPEN
  Relationships: REQ-001 (DERIVES), REQ-035 (DERIVES)

- **REQ-007**
  Level: Stakeholder
  Name: Custom Item Types
  Statement: The system shall enable users to define custom item types.
  Status: OPEN
  Relationships: REQ-008 (DERIVES), REQ-016 (DERIVES), REQ-035 (DERIVES)

- **REQ-008**
  Level: Stakeholder
  Name: Custom User Levels
  Statement: The system shall enable users to define custom user levels.
  Status: OPEN
  Relationships: REQ-007 (DERIVES), REQ-009 (DERIVES)

- **REQ-009**
  Level: Stakeholder
  Name: Custom Levels for Requirements

Statement: The system shall allow users to define levels for requirements, each characterized by a color.
Status: OPEN
Relationships: REQ-008 (DERIVES), REQ-018 (DERIVES)

- **REQ-010**
  Level: Solution
  Name: Display Different Versions of Traceability Hierarchy
  Statement: The system shall manage and display different versions of the traceability hierarchy.
  Status: OPEN
  Relationships: REQ-011 (DERIVES), REQ-012 (DERIVES), REQ-019 (DERIVES), REQ-017 (DERIVES)

- **REQ-011**
  Level: Solution
  Name: Inform User About Traceability Issues
  Statement: The system shall provide warnings about potential issues in the traceability hierarchy.
  Status: OPEN
  Relationships: REQ-005 (DERIVES), REQ-010 (DERIVES)

- **REQ-012**
  Level: Solution
  Name: Visual Management of Traceability Hierarchy
  Statement: The system shall support visual adding, editing, and removing of hierarchy items and their relationships.
  Status: OPEN
  Relationships: REQ-010 (DERIVES), REQ-013 (DERIVES), REQ-019 (DERIVES), REQ-017 (DERIVES)

- **REQ-013**
  Level: Solution
  Name: Display Traceability Hierarchy
  Statement: The system shall visually display the traceability hierarchy.
  Status: OPEN
  Relationships: REQ-012 (DERIVES)

- **REQ-014**
  Level: Solution
  Name: Display Lineage of Hierarchy Item

Statement: The system shall enable the user to see the backward and forward lineage of any hierarchy item.
Status: OPEN

- **REQ-015**
  Level: Solution
  Name: Automatic Derivation of Relationships from Git Commits
  Statement: The system shall automatically derive relationships from git commits mentioning requirements IDs.
  Status: OPEN

- **REQ-016**
  Level: Solution
  Name: Enforce Item Type Selection
  Statement: The system shall enforce the selection of an item type for each new hierarchy item.
  Status: OPEN
  Relationships: REQ-007 (DERIVES)

- **REQ-017**
  Level: Solution
  Name: Enforce User Level Selection
  Statement: The system shall enforce the selection of a user level for each new requirement.
  Status: OPEN
  Relationships: REQ-012 (DERIVES)

- **REQ-018**
  Level: Solution
  Name: Support Internal Levels Within User Level Hierarchy
  Statement: The system shall support defining internal levels within a user level hierarchy.
  Status: OPEN
  Relationships: REQ-009 (DERIVES)

- **REQ-019**
  Level: Solution
  Name: Arrange Nodes Within Traceability Hierarchy
  Statement: The system shall enable rearranging nodes in the traceability graph for better visualization.

Status: OPEN
Relationships: REQ-012 (DERIVES)

- **REQ-020**
  Level: Solution
  Name: Create a Project
  Statement: The system shall enable users to create a project by specifying the project name, repository URL, and access token.
  Status: OPEN
  Relationships: REQ-028 (DERIVES)

- **REQ-021**
  Level: Solution
  Name: Delete a Project
  Statement: The system shall enable users to delete an existing project.
  Status: OPEN

- **REQ-022**
  Level: Solution
  Name: Add a New Release to a Project
  Statement: The system shall enable users to add a new release to an existing project.
  Status: OPEN

- **REQ-023**
  Level: Solution
  Name: Delete a Release from a Project
  Statement: The system shall enable users to delete an existing release from a project.
  Status: OPEN

- **REQ-024**
  Level: Solution
  Name: View Archived Releases
  Statement: The system shall enable users to view, but not edit, archived releases.
  Status: OPEN

- **REQ-025**
  Level: Solution
  Name: Add an Item to the Current Project
  Statement: The system shall enable users to add new items to the

current project.
Status: OPEN

- **REQ-026**
  Level: Solution
  Name: Delete an Item from the Current Project
  Statement: The system shall enable users to delete items from the current project.
  Status: OPEN

- **REQ-027**
  Level: Solution
  Name: Edit an Item in the Current Project
  Statement: The system shall enable users to edit items in the current project.
  Status: OPEN

- **REQ-028**
  Level: Solution
  Name: Validate Repository During Project Creation
  Statement: The system shall validate that the repository URL is valid and accessible with the provided access token.
  Status: OPEN
  Relationships: REQ-020 (DERIVES)

- **REQ-029**
  Level: Solution
  Name: Validate Relationships to Avoid Cycles
  Statement: The system shall prevent the creation of relationships that would introduce cycles in the traceability graph.
  Status: OPEN

- **REQ-030**
  Level: Solution
  Name: Validate Status Changes of Items
  Statement: The system shall enforce specific conditions when changing the status of items.
  Status: OPEN

- **REQ-031**
  Level: Solution
  Name: Validate Item Edits and Provide Warnings

Statement: The system shall notify users of potential issues when editing items.
Status: OPEN
Relationships: REQ-032 (DERIVES), REQ-033 (DERIVES), REQ-034 (DERIVES)

- **REQ-032**
  Level: Solution
  Name: Warn on Editing Requirement with Child Items
  Statement: The system shall warn users when editing a requirement with child items without updating the child items.
  Status: OPEN
  Relationships: REQ-031 (DERIVES)

- **REQ-033**
  Level: Solution
  Name: Warn on Changing Status of Requirement with SourceArtifact Children
  Statement: The system shall warn users when changing the status of a requirement with SourceArtifact children without updating the status of the SourceArtifact items.
  Status: OPEN
  Relationships: REQ-031 (DERIVES)

- **REQ-034**
  Level: Solution
  Name: Suppress Warnings Option
  Statement: The system shall provide an option to suppress warnings when editing items or changing their status.
  Status: OPEN
  Relationships: REQ-031 (DERIVES)

- **REQ-035**
  Level: Solution
  Name: Handle Iterations/Increments
  Statement: The system shall specify how to handle iterations and increments.
  Status: OPEN
  Relationships: REQ-006 (DERIVES), REQ-007 (DERIVES)

- **REQ-036**

Level: Business
Name: Provide Real-World Examples
Statement: The system shall provide real-world examples of requirements traceability in a visual way.
Status: OPEN
Relationships: REQ-005 (DERIVES)

- **REQ-037**
  Level: Business
  Name: Employ User-Friendly SW Tool
  Statement: The system shall employ an easy-to-use software tool that demonstrates basic principles of requirements traceability.
  Status: OPEN
  Relationships: REQ-005 (DERIVES)