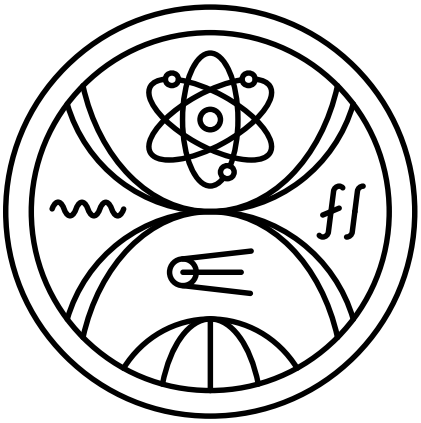
Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

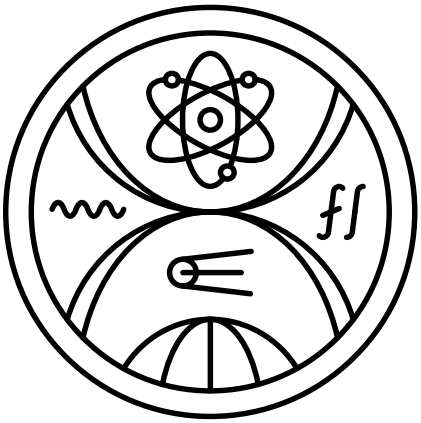


A tool for visualizing software requirements traceability

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

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

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 nástroja na vizualizáciu sledovateľnosti softvérových požiadaviek, ktorý je navrhnutý na podporu vzdelávania v oblasti softvérového inžinierstva na Univerzite Komenského. Nástroj rieši problémy sledovania softvérových požiadaviek počas celého vývojového procesu, kľúčový faktor vo vysokej miere neúspešnosti IT projektov spôsobenej nedostatočne definovanými alebo nesledovateľnými požiadavkami. Vývojom rozhrania založeného na grafe nástroj umožňuje používateľom vizuálne sledovať cesty odvodenia a alokácie požiadaviek, čo zlepšuje pochopenie a manažment softvérových špecifikácií. To prispieva k efektívnejšiemu učeniu a aplikácii princípov vývoja softvéru, čo potenciálne znižuje riziká spojené s nesprávnym manažmentom požiadaviek.

Kľúčové slová: sledovateľnosť softvérových požiadaviek, vizualizačný nástroj, vzdelávanie v oblasti softvérového inžinierstva, grafické rozhranie, manažment IT projektov

Abstract

This bachelor thesis presents the development of a visualization tool for software requirements traceability, designed to support the education of software engineering principles at Comenius University. The tool addresses the challenges of tracing software requirements throughout the development process, a key factor in the high failure rates of IT projects due to poorly defined or untraceable requirements. By developing a graph-based interface, the tool allows users to visually trace the derivation and allocation paths of requirements, enhancing understanding and management of software specifications. This contributes to more effective learning and application of software development principles, potentially reducing the risks associated with requirement mismanagement.

Keywords: software requirements traceability, visualization tool, software engineering education, graph-based interface, IT project management

Contents

List of Figures

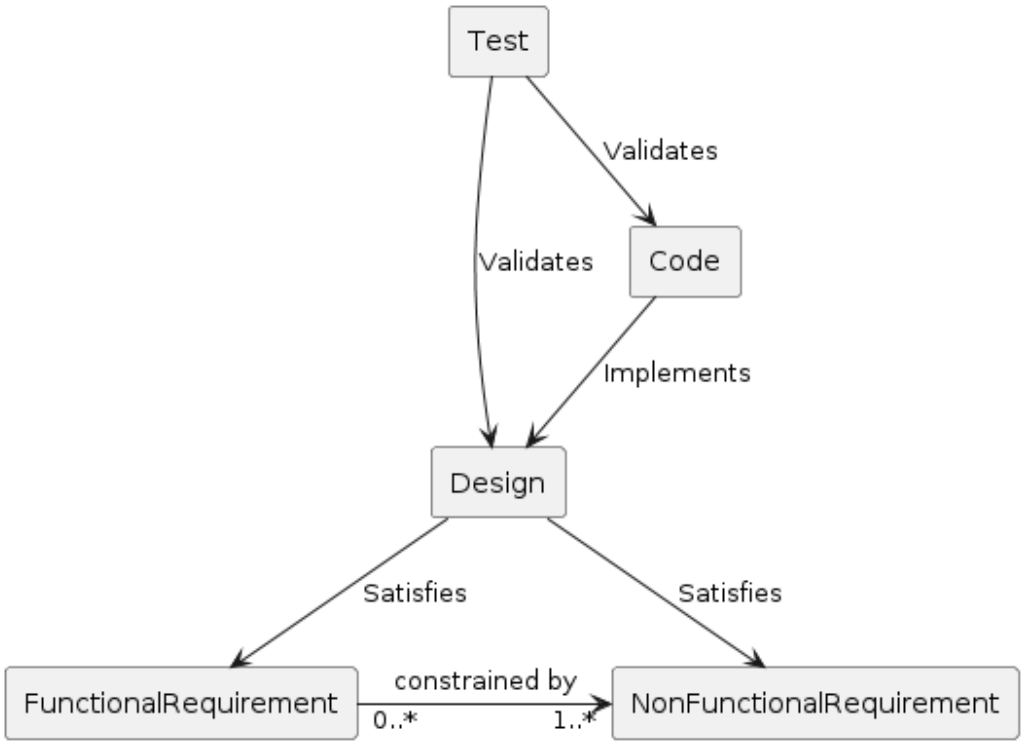


Figure 1: Hierarchy Diagram

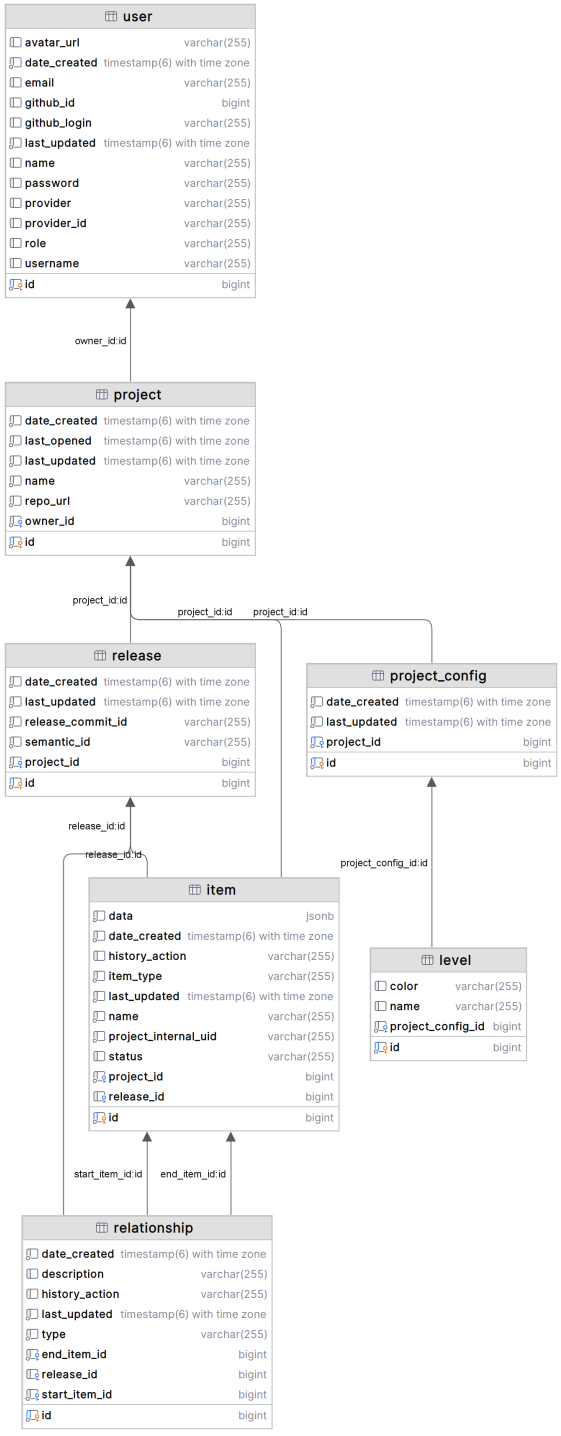


Figure 2: Entity-Relationship Diagram

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

Terminology

- **Requirement:** a feature, constraint, or other property that a system must provide to satisfy the needs of stakeholders [2].
- **Requirement specification (RS):** the assignment of numerical values or limits to a product's design goals [3]
- **Requirement traceability (RT):** the identification and documentation of the derivation path (upward) and allocation/flow-down path (downward) of requirements in the requirements hierarchy [2].
- **Pre-RS traceability:** aspects of a requirement's life prior to inclusion in the requirements specification [4].
- **Post-RS traceability:** aspects of a requirement's life that result from inclusion in the requirements specification [4].

Chapter 2

Specification

2.1 Primary goals

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.

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.

Our application addresses these issues by offering:

- A lightweight, easy-to-use interface that reduces complexity.
- A highly interactive and visually appealing graph representation of requirements.
- High modularity, allowing easy integration with other tools such as Version Control Systems (VCS), eg. git.

Chapter 3

Methodology

This chapter

There are many approaches to requirements traceability. In our opinion, the best way to work with requirements in our case is to represent requirements in the form of a graph. Graph representation is very visually clear [5]. In addition, graph representation of requirements information and other artifacts from other phases of the development process is well established in business analysis and is considered one of the standard approaches to ensure traceability. The graphical representation will facilitate an intuitive understanding of how each requirement evolves from high-level business objectives and is subsequently implemented in source code. This method not only makes the entire lineage of a requirement easily traceable but also includes its validation through testing. By visually depicting the lineage, the application will offer a comprehensive and clear view of each requirement's lifecycle. Below we describe a widely recognized standard for organizing requirements in business analysis from Business Analysis Body of Knowledge (BABOK) [6], most widely globally recognized standard of practice for business analysis, from which we derived our model of requirements hierarchy.

3.1 BABOK requirements hierarchy

Below we will describe briefly how requirements are classified and what are the relations between them in the BABOK.
According to BABOK, we can divide requirements into several categories [6]:

- **Business requirements:** "statements of goals, objectives, and outcomes that describe why a change has been initiated. They can apply to the whole of an enterprise, a business area, or a specific initiative" [6].
- **Stakeholder requirements:** "describe the needs of stakeholders that must be met in order to achieve the business requirements. They may serve as a bridge between business and solution requirements" [6].
- **Solution requirements:** "describe the capabilities and qualities of a solution that meets the stakeholder requirements. They provide the appropriate level of detail to allow for the development and implementation of the solution. Solution requirements can be divided into two sub-categories" [6]:
  - **functional requirements:** "describe the capabilities that a solution must have in terms of the behaviour and information that the solution will manage, and
  - **non-functional requirements or quality of service requirements:** do not relate directly to the behaviour or functionality of the solution, but rather describe conditions under which a solution must remain effective or qualities that a solution must have".
- **Transition requirements:** describe the capabilities that the solution must have and the conditions the solution must meet to facilitate transition from the current state to the future state, but which are not needed once the change is complete. They are differentiated from other requirements types because they are of a temporary nature. Transition requirements address topics such as data conversion, training, and business continuity.

BABOK also defines several types of relation between requirements:

- **Derive:** relationship between two requirements, used when a requirement is derived from another requirement. This type of relationship is appropriate to link the requirements on different levels of abstraction. For example, a solution requirement derived from a business or a stakeholder requirement.
- **Depends:** relationship between two requirements, used when a requirement depends on another requirement. Types of dependency relationships include:
  - **Necessity:** when it only makes sense to implement a particular requirement if a related requirement is also implemented.
  - **Effort:** when a requirement is easier to implement if a related requirement is also implemented.
- **Satisfy:** relationship between an implementation element and the requirements it is satisfying. For example, the relationship between a functional requirement and a solution component that is implementing it.
- **Validate:** relationship between a requirement and a test case or other element that can determine whether a solution fulfills the requirement.

3.2 Our way to make requirements hierarchy

Our model for organizing requirements takes inspiration from the hierarchy from BABOK. We needed to modify our model in several ways to address some limitations of BABOK model discussed in the next section.

In order to properly define the ultimate scope of our application and the nature of future changes, we turn to the article "An Analysis of the Requirements Traceability Problem" [4]. This article provides a detailed comparison of approaches to achieving Requirement Traceability. An exhaustive

study of the effectiveness of traceability techniques involving over hundred practitioners, several focus groups, with in-depth literature and tool reviews conducted.

3.3 Analysis of the requirements traceability problem

The authors [4] broadened the concept of requirements traceability by introducing the definitions of pre-RS traceability and post-RS traceability. The researchers agreed that the vast majority of existing tools emphasized post-RS traceability, yet non-compliance with pre-RS traceability was cited as a major source of problems by the survey participants. The authors proposed recommendations for establishing pre-RS traceability. According to the study, the following factors contribute to ensuring pre-RS traceability:

- Reliable, centralized access to information;
- Historical versions of data would help regain context and reproduce the required details;
- Introducing collaboration capabilities between tools and teams.

The authors also conducted an in-depth comparative analysis of many tools from the domain of requirements engineering that provide in one way or another requirements traceability. It was possible to define four categories based on the purpose and architecture of the tool. Each category has a number of characteristic disadvantages and advantages. Since providing and visualizing the traceability hierarchy is our main goal, we can classify our tool into category C - workbench, or RT (requirements traceability) workbench.

Based on the characteristics of each category of the reviewed tools, we can put our tool in the workbench category, since visualization of requirement traceability is our main goal.

We identified the critical areas of our application that require special attention based on the shortcomings of this type of application described in the article, namely:

- RT workbenches aim to be comprehensive, but none support all activities. They typically enforce a top-down approach, use classification schemes, and assume a relatively static baseline without support for its development. Since RT depends on correct usage, the primary concern often becomes RT itself rather than RS production.

- RT workbenches have poor integration, making it difficult to support RT for early problem definition work or to provide ongoing RT with later changes, often requiring significant manual intervention.

Using these observations, we tried to adapt our application and the hierarchy model in such a way as to

1. get rid of the typical drawbacks of workbench-type tools;

2. ensure the highest possible degree of Pre-RS traceability;

3. make the model more universal so that it can be used in different development models - iterative-incremental, Agile and waterfall.

To begin with, let us describe the changes required to fulfill the requirements described above.

3.4 Handling functional and non-functional requirements in different environments

If the non-functional requirements are stated separately from the functional requirements, the relationships between them may be hard to understand [7].

Also, measuring the impact of requirement change requires an understanding of the specific context in which the project is developing. Moreover, only by understanding this context can the prioritization and feasibility of changes be determined. For example, in the waterfall methodology, the compliance with non-functional requirements (e.g., security) is critical throughout the life of the project. Therefore, requirements specification and implementation in waterfall are two distinct phases. At the same time, in less error-prone planning environments such as iterative-incremental and Agile, changes to requirements are welcomed, even late in development [8].

Therefore, there is a need to separate functional requirements from non-functional requirements in our hierarchy. Below, we review approaches to distinguish functional requirements from non-functional requirements and describe the benefits they bring to traceability in the context of different models. To make our hierarchy fit all development models, we must separately consider the specifics of requirements tracing within each type of environment.

3.5 Tracing non-functional requirements

To analyze existing practices for tracing non-functional requirements, we will use the work of Mahmoud et al. [9]. This work describes several approaches to detecting, classifying and tracing non-functional software requirements. Many of these approaches focus on semantic code analysis and establishing connections with non-functional requirements using statistical methods. In some of the reviewed works, models are proposed for the description of requirements hierarchy. We will disscuss them in the next section and present skeleton of our hierarchy model based on described techniques.

Peraldi-Frati et al. [10] suggested a method for representing and tracing basic non-functional requirements in safety-critical systems. The authors used formal models to demonstrate the satisfaction of a specific NFR in the system through certain predefined relations. Such relations were then used

to establish traceability links between design artifacts and requirements and link requirements to validation and verification elements (test cases) in the system.

3.5.1 Mapping requirements in Agile environments

There is no single way to document requirements in Agile development. The BABOK model distinguishes three types of requirements — business, stakeholder, and solution requirements. Sommerville [7] identifies only two lower levels of requirements — user requirements and system requirements. Another widely used method of organizing requirements is the Big Picture of Agile Requirements, where requirements are distributed at three levels — team level, program level and portfolio level [11].

Rempel and Mäder [12] 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 proposed model is effectively applied in tools for requirements engineering. However, the source code in this model is considered a single artifact high-level object, without visibility associated with the test case class. In large projects, this can be a bottleneck when managing risks and change influence due to requirement changes.

Firdaus et al. [13] proposed the Traceability Process Model (TPM) to support the requirements traceability in Agile development. The proposed model in the article integrates non-functional requirements with user stories and validates them through test cases. This approach ensures that NFRs are explicitly modeled and their relationships with functional requirements (FR) are maintained throughout various development stages. By providing test cases that test NFRs, the model facilitates tracking changes back to both FR and NFR, thus supporting comprehensive traceability in Agile development.

3.6 Updating our model

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

- Distinguishing between functional requirements and non-functional requirements.
- Ensuring traceability of non-functional requirements by introducing relationships between:
  - Functional requirements (FR) and non-functional requirements (NFR)
  - Design elements
  - Code elements
  - Test elements
- 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).
- Adding items at sublevels within each level.
- Introducing an element of rigidity where relationships are distinguished by type and strictly applied to specific types of requirements or elements as follows:
  - A **Derives** relationship is fixed between different levels of requirements to ensure traceability and rationale.
  - The **Satisfy** relationship is strictly applied between the code (or design elements) and the corresponding requirements to demonstrate implementation fidelity.

  - A **Validate** relationship connects each requirement with its respective test case or validation method to confirm the requirement's fulfillment.

- 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.

This allows us to utilize various frameworks for mapping requirements in different environments and provides traceability for every item throughout the entire development lifecycle.

The resulting diagram describing relationships between items in our traceability hierarchy is provided as Figure 1.

Now that the foundation of our model has been built, we can analyze existing tools from our domain that serve similar purposes. This will help us to accurately define the objectives we need to achieve in our implementation to fulfill the goals described above.

Chapter 4

Existing solutions

Now that the foundation of our model has been built, we can analyze existing tools from our domain that serve similar purposes. This will help us to accurately define the objectives we need to achieve in our implementation to fulfill the goals described above.

We looked into a variety of requirement management tools, including both paid proprietary solutions and free open-source options.
We chose three proprietary solutions — IBM Rational, CodeBeamer and Visure — because are notable for their functionalities, particularly in source code integration and parsing, and in designing hierarchies of requirements [14]. Below we focus on how exactly these applications help to achieve better requirement traceability with these approaches.

4.1 IBM Rational

IBM Rational [15] provides a suite for requirements management suitable for complex software and systems engineering projects. Key features include:

- Traceability, linking requirements with various project artifacts.
- Facilitation of collaboration across geographically dispersed teams.

- Capability to handle large-scale projects.
- Integration with various software development tools.

4.2 CodeBeamer

CodeBeamer [16] is an Application Lifecycle Management (ALM) tool with strong integration capabilities with Software Configuration Management (SCM) systems. This tool provides several features for tracking requirements fulfillment in source code, such as:

- Use of fuzzy parsers for languages like C/C++ and Java.
- Ability to extract and relate programming constructs such as classes, methods, and variables.
- Accommodation of preprocessor directives for enhanced C/C++ parsing accuracy.

4.3 Visure

Visure 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 [17].

4.4 Open-Source Requirements Management 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.

4.5 rmtoo

rmtoo [18], designed primarily for Unix-based systems, 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.

4.6 Doorstop

Doorstop [19] builds upon the decentralized model introduced by 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 [20], 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.

4.7 StrictDoc

StrictDoc [21] 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, including those who may not have technical expertise in command-line tools.

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.

4.8 Limitations of Existing Tools

Existing tools, both commercial and open-source, present several limitations 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 and Doorstop also have limitations:

- **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).

In practice, keeping software code in sync with regulatory documentation is a significant challenge. Doorstop offers 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 5

Development of the Application

In the initial stage of application development, key architectural decisions were made that determine the structure and functionality of the future product. One of the first and most significant decisions was the choice of the application type and its technical stack. Considering the users' need to work with a highly interactive requirements graph, a decision in favor of a client-server application was made.

5.1 Chosen toolset

Angular [22] 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 [23], 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 [24]. Framemork 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.

5.2 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 (Java Persistence API) was used to manage the data and interact with the database.

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.

- Release Management: To ensure traceability of changes in the graph, the development of the graph is captured in the form of Release entities. The Release 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.

  Full entity-relation diagram is attached as Figure 2.

5.3 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 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.

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

5.5 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. Full description of validations is provided in the requirements specification.

Chapter 6

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 lineage:** The system successfully implemented both backward and forward lineage, 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] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Washington, DC, USA, 3rd edition, 2014.

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

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

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

[8] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew

Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001.

[9] Anas Mahmoud and Grant Williams. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, 21(3):357–381, Sep 2016.

[10] Marie-Agnès Peraldi-Frati and Arnaud Albinet. Requirement traceability in safety critical systems. pages 11–14, Apr 2010.

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

[12] 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.

[13] Adila Firdaus Binti Arbain, Imran Ghani, and Wan Mohd Nasir Wan Kadir. Agile non functional requiremnents (NFR) traceability metamodel. In *2014 8th. Malaysian Software Engineering Conference (MySEC)*, pages 228–233, 2014.

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

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

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

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

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

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

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

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

[22] 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.

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

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

Appendix A

Myproject 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)