**Automatic Visualization of Traceability Information**

by:

Author

A Thesis Submitted in Fulfillment of the Requirements for the Degree of

**MS of Computer Information Systems**



Department of Computer Science and Technology Wenzhou-Kean University

Supervised By: Supervisor Name

©Month Year

Bachelor of Science (2025) Wenzhou-Kean University

|  |  |
| --- | --- |
| TITLE: | Automatic Visualization of Traceability Information |
| AUTHOR: | Name  Msster., (Computer Science and Technology)  Wenzhou-Kean University  Wenzhou, China |
| SUPERVISORS: | Supervisor |
| NUMBER OF PAGES: | 46 |
|  |  |

Computer Science and Technology Wenzhou, China

# Abstract

Classical Traceability Management Systems (TMS) help track links between software parts like requirements, designs, Code, and test cases. They help keep projects organized and meet quality goals. But they have issues. Pulling out data takes too long. Searching is hard. The results are messy and not easy to understand. These problems slow teams down and make fast decisions, especially for agile teams.

This study proposes an improved Traceability Management System (TMS) for software engineering processes. The system retrieves data in real-time and presents it through dynamically updating charts. It is designed to operate with greater speed and simplicity, thereby enhancing team coordination and progress monitoring.  
Utilizing regular expressions, the system efficiently extracts critical information from intricate software datasets. This extracted data is subsequently transformed into comprehensible visual representations.

**Keywords:** Traceability; Automatic; Regular Expression; Visualization; TMS.

# Acknowledgments

I thank my supervisor, Dr. Nasser Mustafa from the Department of Computer Science, for his steady support during my undergraduate thesis. His advice and encouragement helped me stay focused and finish this work. I learned a lot from him, and it made a big difference in how I see my future.

I also wish to extend my heartfelt thanks to my fellow students in the Department of Computer Science for their friendship and support, which went a long way towards the completion of this work.

My heartfelt gratitude to my family and friends for their constant encouragement and faith in me, offering the strength to carry out this thesis.

**Table of** Contents

[Abstract iii](#_Toc199499919)

[Acknowledgments iv](#_Toc199499920)

[List of Figures vii](#_Toc199499921)

[List of Tables viii](#_Toc199499922)

[Definition of Acronyms ix](#_Toc199499923)

[Chapter 1 Introduction 1](#_Toc199499924)

[1.1 Background 1](#_Toc199499925)

[1.2 Problem Statement 1](#_Toc199499926)

[1.3 Project Objectives 2](#_Toc199499927)

[1.4 Scope 3](#_Toc199499928)

[1.5 Significance 4](#_Toc199499929)

[1.6 Target Audience 4](#_Toc199499930)

[Chapter 2 Literature Review 6](#_Toc199499931)

[2.1 Previous Work 6](#_Toc199499932)

[2.2 Research Gaps 7](#_Toc199499933)

[2.3 Summary 9](#_Toc199499934)

[Chapter 3 Methodology 10](#_Toc199499935)

[ Layered Architecture Components 11](#_Toc199499936)

[3.1 Actors and Use Cases 13](#_Toc199499937)

[3.2 Use Case Diagram 14](#_Toc199499938)

[3.3 User Requirements 14](#_Toc199499939)

[3.4 System Requirements 16](#_Toc199499940)

[3.5 Class Design 20](#_Toc199499941)

[Chapter 4 Implementation 23](#_Toc199499942)

[4.1 Frontend Implementation 23](#_Toc199499943)

[A. Artifact Creation Form 24](#_Toc199499944)

[B. Result Display 25](#_Toc199499945)

[4.2 Backend Implementation 25](#_Toc199499946)

[A. Database Schema 26](#_Toc199499947)

[B. Controller Layer 27](#_Toc199499948)

[C. Service Layer 27](#_Toc199499949)

[D. Data Access Layer 29](#_Toc199499950)

[E. Data Extraction Using Regular Expressions 29](#_Toc199499951)

[ Identify the Information to Extract and Its Patterns 30](#_Toc199499952)

[ Retrieves text data from the database through SQL queries 31](#_Toc199499953)

[ Apply regular expression to extract the state. 32](#_Toc199499954)

[ Batch Processing Multiple Records 32](#_Toc199499955)

[ Verify the extracted data. 32](#_Toc199499956)

[4.3 Testing and Validation 33](#_Toc199499957)

[ Case Study1: Testing Across Different Artifacts 34](#_Toc199499958)

[ Case Study 2: Impact of Case Sensitivity on Extraction Results 35](#_Toc199499959)

[4.4 Result and Discussion 36](#_Toc199499960)

[ Data Extraction Efficiency 37](#_Toc199499961)

[ Key Findings 37](#_Toc199499962)

[Chapter 5 Conclusion 40](#_Toc199499963)

[5.1 Implications for Traceability Management 40](#_Toc199499964)

[5.2 Limitations and Challenges 41](#_Toc199499965)

[5.3 Future Work 41](#_Toc199499966)

[Bibliography 43](#_Toc199499967)

# List of Figures

[**Figure 1: Layered Architecture 10**](#_Toc198576309)

[**Figure 2: Use Case Diagram 14**](#_Toc198576310)

[**Figure 3: Class Diagram 20**](#_Toc198576311)

[**Figure 4: Shows the artifact.html structure. 24**](#_Toc198576312)

[**Figure 5: Shows the message display logic in artifact.html. 25**](#_Toc198576313)

[**Figure 6: ArtifactController Implementation 27**](#_Toc198576314)

[**Figure 7: Name generation logic 28**](#_Toc198576315)

[**Figure 8: ArtifactMapper XML 29**](#_Toc198576316)

[**Figure 9:Sequence Diagram 30**](#_Toc198576317)

# List of Tables

[**Table 1: Definition of Acronyms ix**](#_Toc199498482)

[**Table 2. Mainstream TMS Frameworks 8**](#_Toc199498483)

[**Table 3: Actors and Use cases 13**](#_Toc199498484)

[**Table 4: Explanation of Key Classes 21**](#_Toc199498485)

[**Table 5: Key tables and their fields. 26**](#_Toc199498486)

[**Table 6: Queries Input and Output 36**](#_Toc199498487)

# Definition of Acronyms

**Table 1:** **Definition of Acronyms**

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| TMS  SDLC | Traceability Management System  Software Development Life Cycle |
| CRUD | Create, Read, Update, Delete |
| ORM | Object-Relational Mapping |
| IDE | Integrated Development Environment |

# Chapter 1 Introduction

## 1.1 Background

Traceability Management Systems (TMS) are application software utilized in the management of relationships among artifacts such as requirements, design, Code, and test cases in software development. They enable projects to remain on schedule, achieve quality expectations, and facilitate teamwork. They also assist in compliance in big and complicated software development projects [1].

Agile and DevOps need TMS more than ever. These methods focus on fast updates and constant delivery. So, they rely on TMS to give real-time info and help teams make quick decisions [2]. But most old TMS can't keep up. They take too long to pull data, are hard to search, and only show static results like tables or reports. They also need a lot of manual input and tech skills [3]. These problems slow teams down and make it hard to keep quality high in fast-paced work [3].

## 1.2 Problem Statement

Traditional Traceability Management Systems (TMS) have trouble keeping up with modern software development. The main problems are:

* **Slow data updates**: Data is often added in batches or by hand. That means it's old and doesn't show what's really happening in the project.
* **Complex Queries**: Queries are hard to run and need technical skills. This limits non-technical users and slows decisions [4].
* **Unclear Outputs**: Static tables or reports aren't easy to understand. They lack visuals to show trends or issues quickly.
* **Manual Processes**: Manual or semi-automated data entry causes errors and slows work, especially in fast-changing projects.
* **Poor Fit for Modern Methods**: Traditional TMS can't keep up with Agile and DevOps' fast cycles. This hurts responsiveness and quality.

These problems make it tough for teams to track progress, manage risks, and ensure compliance. A better, automated solution is needed for today's development needs [3].

## 1.3 Project Objectives

This project builds an automated TMS for software development. It fixes problems in older systems. The goals are:

1. **Real-time data:** The system should pull traceability data from databases right away so teams see the latest info [3].
2. **Less manual work:** It uses regular expressions to get and process data from complex files. This cuts down on errors and saves time.
3. **Dynamic Visualization**: Turn data into clear, interactive charts (like line or bar charts) to help teams see progress and spot issues fast [2].
4. **Simple Interface**: Build a web-based interface with easy controls (like checkboxes), so non-technical users can use it without much training.
5. **Support for Modern Methods**: Design the system to fit Agile and DevOps' fast cycles, keeping it flexible and responsive [7].

## 1.4 Scope

The scope of this project focuses on the development of an automated Traceability Management System that processes and presents traceability data from existing software development databases, such as requirement and test management tools [5]. The system tracks connections between artifacts like requirements, designs, Code, and test cases. It assumes these artifacts exist in databases. The focus is:

* **Data Extraction and Processing**: The system uses regular expressions to pull and process data. It automates these tasks to cut down on manual work.
* **Visualization and Interface**: The system will provide dynamic, chart-based visualizations (using tools like Chart.js) and a web-based interface to enhance usability [8].
* **Integration with Existing Tools**: The system will interface with existing software development databases but will not include the creation or entry of new data.
* **No Data Creation**: The system uses existing data. It doesn't create requirements, designs, Code, or test cases.
* **Software Development Focus**: The system is built for software teams. It doesn't cover traceability for fields like manufacturing or hardware engineering [9].

## 1.5 Significance

The automated Traceability Management System improves software development by fixing traditional TMS problems and supporting modern methods. Its main benefits are:

* **Better Efficiency**: Real-time data extraction and automation cut manual work, reduce errors, and speed up tasks.
* **Clearer Decisions**: Dynamic charts help teams spot bottlenecks, risks, and trends quickly, aiding decisions in critical projects.
* **Easy for All Users**: A simple interface makes the system accessible to non-technical users, lowering training needs and increasing use.
* **Fits Agile and DevOps**: The system's flexibility matches the fast cycles of modern methods, keeping traceability in sync with development.
* **Supports Compliance and Quality**: Real-time, clear data shows adherence to standards, helping industries like healthcare and aviation.

By cutting manual tasks and providing clear insights, the system boosts project management, improves quality, and ensures compliance. It's a practical tool for today's software development.

## 1.6 Target Audience

The system serves various groups in software development:

* **Software Teams**: Project managers, developers, testers, and quality assurance staff who track progress, ensure quality, and manage artifact connections [7].
* **Product Managers**: People need clear views of requirement progress, user feedback, and project alignment with business goals [7].

The system supports both technical and non-technical users, making it useful across the software development field.

# Chapter 2 Literature Review

In complex system development, a Traceability Management System (TMS) helps connect different artifacts. It also plays a key role in searching for information and finding how things are related [1]. As systems become more varied, it gets harder for TMS to support good queries. This makes it tough to manage quality and track changes.

This review looks at how current TMS handles information searching. It focuses on problems with mixed data types, how relationships are modeled, and how tools are connected.

## 2.1 Previous Work

**Problems with Searching in Mixed Systems**

Old TMS tools deal with many data types. These include text requirements, test cases, source code, models like UML, and version control info. These are often stored in different tools and formats. That makes it hard to search through them.

Here are some of the main issues:

* **Unknown Files**: Some files, like graphs or Excel sheets, can't be recognized easily [4].
* **No Indexing**: Some content (like text files or diagrams) isn't indexed well, so you can't search it easily [7].
* **Simple Queries Only**: Most TMS only let you search with direct links. They don't support tracking changes or link direction [7].
* **Weak Search Tools**: Advanced searches, such as using natural language, logic, or network searches, are not supported. Most systems only offer basic filters [9].

Some systems use OSLC, which lets tools connect using REST [19]. But it mainly just pulls resources. It doesn't support smarter, meaning-based search [11]. RDF and SPARQL could help, but they need strong data models and logic, which are usually missing [14].

Newer research is trying machine learning and NLP to find important data and create smarter search graphs [9][13]. However, these tools need good training data and don't work well in complex fields like aerospace or healthcare [8].

**Search Methods in Common TMS Tools**

A review of mainstream TMS platforms shows varying degrees of support for artifact recognition, indexing, and semantic querying, as you can see in Table 2 later.

## 2.2 Research Gaps

Today's TMS tools still have many problems when it comes to searching for information:

* **No Auto Indexing or Merging Data**: Many systems can't detect unregistered files like diagrams, comments in Code, or hardware files. This limits what users can search for [1].

**Table 2. Mainstream TMS Frameworks**

| **Framework** | **Information Types and Query Mechanisms** | **Limitations** |
| --- | --- | --- |
| IBM Rational DOORS | Textual requirements, attribute-based and hierarchical queries | No recognition of graphics/code, weak semantics, explicit links only |
| ORCANOS ALM/QMS | Test and risk items, form-based queries and traceability reports | Manual indexing configuration, lacks path analysis |
| JIRA + Confluence | Tasks and documents, tag- and field-based filtering queries | No cross-artifact recognition, no semantic search |
| Enterprise Architect | UML/SysML models, visual matrices and path navigation | Limited to internal model structures, no external recognition [18][5] |
| CodeBeamer ALM | Requirements, tasks, and code, nested filtering queries | No semantic extraction, requires explicit cross-format link configuration |
| IBM Rational Team Concert | Development tasks and versions, status field filtering queries | No graphics recognition, weak semantic reasoning |

* **Fixed Query Structure**: Most searches are stuck in fixed patterns. There's no way to explore through flexible paths or find causes based on relationships [4].
* **No Support for Fuzzy or Smart Searches**: Systems don't understand vague or natural language questions. They rely too much on exact keyword matches [13].
* **Inconsistent Detail Recognition**: Some tools only track whole documents. They don't look into smaller parts, like one paragraph in a requirement file or a specific function in Code [3].
* **No Shared Data Format**: Tools in different fields don't use the same format. Users often need to re-enter queries or change views to get full results.

These problems are similar to what we saw earlier—slow searches, unclear results, and weak support for real-time data. To move forward, TMS needs to fix these issues with smarter, more connected systems.

## 2.3 Summary

TMS plays a key role in managing complex systems. However, current tools are still weak when searching through mixed types of data. While some support exists, most tools can't index all file types, allow flexible searching, or understand meaning.

To improve, we need better TMS features like:

* **Automatic indexing**.
* **Smarter, meaning-based search**.
* **Shared data formats across tools**.

These changes will help teams find information faster and work more smoothly across systems.

# Chapter 3 Methodology

TMS plays a key role in managing complex systems. However, current tools are still weak when searching through mixed types of data. While some support exists, most tools can't index all file types, allow flexible searching, or understand meaning.

图示

AI 生成的内容可能不正确。

**Figure 1:** **Layered Architecture**

To improve, we need better TMS features like:

* **Automatic indexing**.
* **Smarter, meaning-based search.**
* **Shared data formats across tools**.

These changes will help teams find information faster and work more smoothly across systems. The Figure 1 Layered Architecture, which I designed for my system. Each layer has its own job. This structure helps manage complex data and connect different tools and artifacts.

### Layered Architecture Components

* **Presentation Layer**

This is the part users see and interact with. It shows traceability information in charts and tables. The layer is built using Thymeleaf templates integrated with Spring Boot. Users can select artifacts using dropdown menus or checkboxes, and the view updates dynamically.

For example, when a user selects a class from the menu, they'll see its related functions rendered in a diagram. Nodes represent artifacts, and lines show their relationships. User actions are sent to the Application Layer via HTTP requests, and the response data is rendered back in the web interface.

* **Application Layer**

This layer handles the system's logic. It connects user interactions from the interface with the data stored in the database. It is implemented using Spring Boot. When a user selects, this layer determines what data to retrieve. It uses regular expressions and custom business logic to match relevant items. For example, it might apply a pattern like ^function. \*classID=[A-Za-z0-9]+$ to locate all functions linked to a given class. It then organizes the results into nodes and edges and returns them in JSON format to the Presentation Layer.

* **Data Access Layer**

This layer bridges business logic and the database. It uses MyBatis to execute SQL queries and map the results to Java objects. It ensures data integrity while providing the Application Layer with access to artifact records.

For instance, when the Application Layer requests all "satisfy"-type trace links related to a specific artifact, this layer runs a query using a pattern like **satisfy.\*artifactID=[A-Za-z0-9]+.** It also handles insert and update operations. Any modifications are immediately reflected in the user interface.

* **Database Layer**

This is where all the data is stored. It uses MySQL. There are tables for things like artifacts, trace links, versions, projects, and users.

The **Artifact** table holds info like the ID, Type, and description. The **TraceLink** table stores links between artifacts, including the Type of link and the target artifact [19]. This setup makes it easy for the system to store and find traceability data quickly, even when there are lots of different file types.

## 3.1 Actors and Use Cases

To clearly show how people interact with the traceability management system, I defined the main users and their tasks. This is shown in Table 3. The actors include Programmer, Project Manager, Tester, System Analyst, Designer, and System Administrator. Each actor has a specific role. For example, programmers write Code, project managers handle traceability data, and system admins configure the system or manage data access. This setup helps explain who does what and how each person uses the system.

**Table 3: Actors and Use cases**

| **Actor** | **Description** | **Associated Use Cases** |
| --- | --- | --- |
| Programmer，  Project Manager,  Tester  System Analyst  Designer | Responsible for developing code and managing traceability information | Select Traceability Data, View Visualization Results, Add Traceability Data, Update Traceability Data, Identity Authentication, Generate Real-Time Charts, Revise Traceability Data, Delete Traceability Data |
| System Administrator | Responsible for system configuration and maintenance | Configure Data Source, Add Traceability Data, Update Traceability Data, Extract Traceability Information, Log Exceptions |

## 3.2 Use Case Diagram

The following Figure 2: Use Case Diagram illustrates these actors and their interaction with the system graphically, giving a good indication of the functional requirements and user roles in the traceability management process.

图示, 示意图

AI 生成的内容可能不正确。

**Figure 2: Use Case Diagram**

## 3.3 User Requirements

Here, I will explain some key user requirements.

* The system shall allow a Programmer to log in using their credentials.
* The system shall allow a Programmer to select traceability data using checkboxes for requirements, designs, codes, or test cases.
* The system shall allow a Programmer to view real-time charts showing links between Code and other artifacts.
* The system shall allow a Programmer to add new code data for tracking.
* The system shall allow a Programmer to update existing Code or links to reflect changes.
* The system shall allow a Project Manager to log in using their credentials.
* The system shall allow a Project Manager to select data through a simple interface to track progress.
* The system shall allow a Project Manager to view live charts to check project status and find issues.
* The system shall allow a Programmer to revise code links or metadata to fix mistakes.
* The system shall allow a Programmer to delete outdated or incorrect code entries.
* The system should enable system analysts to input new requirements or analyze data.
* The system should enable system analysts to modify existing associations or information content.
* The system should enable system analysts to delete outdated or inaccurate modeling data.
* The system should enable designers to authenticate using their credentials.
* The system should enable designers to access visual representations to verify design traceability.
* The system should enable designers to incorporate new design elements or establish new connections.
* The system should allow system administrators to introduce new configuration parameters.
* The system should allow system administrators to change system configuration Settings.
* The system should enable system administrators to perform queries or generate reports for system supervision.

## 3.4 System Requirements

**A. Select Traceability Data**

The system allows users to select the traceability type (such as requirements, design, code, or test cases) using checkboxes.

The system allows users to set filters for the selected data (such as project stage or status).

The system checks whether the selection is valid.

* If the selection is valid (e.g., at least one data type is chosen), the system proceeds to extract the corresponding traceability information.
* Otherwise, the system displays an error message indicating, "Please select at least one data type."

**B. Extract Traceability Information**

The system allows the database to provide original traceability data based on the user's selection.

The system uses regular expressions (such as REQ-\d{3}) to search for key items such as requirement ID.

The system uses regular expressions to search for metadata, such as states and timestamps.

The system checks whether the extracted data is correct.

* If the extracted data is valid (e.g., matches expected patterns), the system structures the data into a JSON format for further processing.
* Otherwise, the system logs an exception with details of the invalid data and continues with valid data.

**C. Generate Real-Time Charts**

The system processes the data to prepare it for chart creation.  
The system creates live charts (like line or bar charts) using a chart library such as Chart.js.  
The system updates the charts as soon as the data changes.  
The system checks if the chart was created correctly.

* If yes, the system shows it to the user.
* If no, the system shows an error that says, "Chart generation failed."

**D. View Visualization Results**

The system lets a user see the charts on a web dashboard.  
The system lets a user interact with charts (like zooming or hovering to see more info).  
The system checks if the charts are shown correctly.

* If the results are correctly displayed, the system maintains the dashboard in real time.
* Otherwise, the system logs the display issue and notifies the user with an error message.

**E. Configure Data Source**

The system lets an admin enter the database URL and login info.  
The system lets an admin add regular expressions to find data.  
The system lets an admin save these settings.  
The system checks if the settings make sense.

* If the configuration is valid (e.g., successful database connection), the system applies the settings and confirms success to the administrator.
* Else, the system sends an error message to the administrator indicating "Configuration failed."

**F. Log Exceptions**

The system can detect errors during data extraction or processing (like when a pattern doesn't match).  
The system saves error details (like time and error type) in a log file.  
The system lets an admin check the log for these errors.  
The system validates the logging process.

* If the exceptions are successfully logged, the system continues normal operation.
* Else, the system alerts the administrator with a message indicating "Logging failure."

**G. Identity Authentication**

The system offers a way for the user to enter a username or email address as a source of authentication.  
The system gives a user the option to type in a password to confirm their identity.  
The system allows an authenticated user to submit their authentication credentials for validation.  
The system checks the user's credentials with records on the database.

* If the credentials are valid (e.g., match the stored username and hashed password), the system grants access to the user and redirects them to the dashboard.
* Else, the system sends an error message to the user indicating "Invalid username or password" and prompts them to retry.

**H. Revise Traceability Data**

The system offers a user the ability to choose a pre-existing traceability artifact for updating.  
The system allows a user to modify relationships (e.g., re-linking a test case to a different requirement) or metadata (e.g., fixing artifact descriptions).

## 3.5 Class Design

The class diagram in Figure 3, backend Implementation, shows how the Traceability Management System (TMS) works. It's a structured plan for managing software development artifacts and automating traceability tasks. It includes the main classes: User, Project, Artifact, Version, TraceLink, and Login [13]. These classes work together to store and manage data and track links between items like requirements, source code, and test cases.

图形用户界面, 表格

AI 生成的内容可能不正确。

**Figure 3: Class Diagram**

Table 4 explains what the key class does to help automate tasks and make the system more efficient:

**Table 4: Explanation of Key Classes**

| **Class** | **Key Attributes** | **Key Methods** | **Role in Automating Traceability Data Operations** |
| --- | --- | --- | --- |
| **User** | userID, role | login(), logout() | Automates user authentication and role-based access to traceability data, ensuring secure and efficient interaction with the system . |
| **Artifact** | artifactID, type, description | createArtifact(), addTraceLink(), generateGraph() | Automates artifact creation, link establishment, and visualization of traceability relationships, reducing manual effort in managing and tracing artifacts. |
| **Version** | modifiedOn, recordChange | - | Automates versioning of artifacts, preserving historical data to support trace link recovery across different artifact versions. |
| **TraceLink** | traceLinkID, type, targetArtifact | - | Automates the identification and recovery of traceability links by capturing relationship types (e.g., "satisfy"), enabling the system to infer missing links. |

* **Automated Link Creation and Recovery**: The addTraceLink() method in the Artifact class works with the TraceLink class's type field (like "satisfy" or "control"). This helps the system create and recover links between artifacts automatically. For example, the system can tell that a function meets a requirement by comparing its descriptions. This reduces manual work.
* **Graph view:** The generateGraph () method in the artifact class shows how different items are connected. This can help users discover missing or damaged links without having to draw any content manually.
* **Version tracking:** Version classes save records of changes. This ensures that the links remain correct even if things are updated.
* **Project-based grouping:** The Project class groups related items together. This makes all the content organized and ensures that the system sees each item in the correct context.
* **Secure Login:** The User and Login classes control who can use the system and what they are allowed to do. This can protect your data from unauthorized alterations.

By automating these tasks, the system cuts down on manual work, reduces mistakes, and shows clear, easy-to-read trace links. This makes it easier to manage traceability throughout software development.

# Chapter 4 **Implementation**

This chapter explains how the traceability system for software developers was built. It focuses on the main features and how they work. The system helps manage artifacts during software development. It includes features like user login/logout, creating, retrieving, updating, and deleting artifacts, version control, trace links for relationships, change logs for auditing, file uploads for Code artifacts, and dynamic interfaces for different artifact needs. Since there are many features, this chapterhighlights artifact creation to show how users interact with the system, how data is processed, and how it's stored. The system has two main parts: the front end for user interaction and the back end for data handling. Section A covers the front end, which is built with Thymeleaf. Section B explains the backend, which was built with Spring Boot and MyBatis.

## 4.1 Frontend Implementation

The traceability system's front end uses Thymeleaf with Spring Boot to build a user-friendly interface. It lets users create artifacts like requirements or Codes and shows if the operation worked or failed. The design is simple and interactive for easy data input and feedback.

### A. Artifact Creation Form

The artifact.html template is the main page for creating artifacts. It has a form with a dropdown to pick the artifact type and fields for extra details. For example, choosing "Requirement" shows a text area for a description. Picking "Code" shows a file upload field.

JavaScript toggles are fields that show up based on the Type selected. The form sends data to the backend via a POST request to /artifact/add. It uses multipart/form data for file uploads with Code artifacts. Figure 4 shows the artifact.html structure.

文本

AI 生成的内容可能不正确。

Figure 4: Shows the artifact.html structure.

The toggle Fields JavaScript function hides or shows fields to keep the form clean and show only relevant inputs. This keeps everything on one page with clear feedback for success or failure.

### B. Result Display

After submitting the form, the backend processes it and returns it to artifact.html with a success or failure message. Thymeleaf shows the message based on what the backend sends. Success messages ("Added successfully") are green, and error messages ("Fail to add ") are red. Users get instant feedback without leaving the form and can keep creating artifacts.

Figure 5 shows the message display logic in artifact.html.

文本

AI 生成的内容可能不正确。

Figure 5: Shows the message display logic in artifact.html.

## 4.2 Backend Implementation

The backend uses Spring Boot and MyBatis for database work with a MySQL database. It handles user requests, processes data, and saves it. It has three layers: controllers for HTTP requests, services for business logic, and data access for database interactions. I used Papyrus to make UML class diagrams. These included classes, attributes, and methods. Then, I used Papyrus with the Acceleo plugin to generate Java code. The Code had class definitions, fields, and method names. Next, I started to create my database.

### A. Database Schema

**Table 5: Key tables and their fields.**

| **Table** | **Key Fields** | **Purpose** |
| --- | --- | --- |
| User | User\_ID, Username, Email | Stores user information |
| Artifact | Artifact\_ID, Name, Type, Description, Created\_By, Created\_On | Stores artifact details |
| TraceLink | TraceLink\_ID, Source\_Artifact\_ID, Target\_Artifact\_ID, Relationship\_Type | Defines relationships between artifacts |
| Artifact\_Version | Version\_ID, Artifact\_ID, Version\_Number, Content, Modified\_By | Tracks artifact version history |
| Change\_Log | Log\_ID, Artifact\_ID, Version\_ID, Change\_Description | Logs changes for traceability |

The database has three main entities: User, Artifact, and TraceLink. It also has Artifact\_Version and Change\_Log for versioning and auditing. . lists the key tables and their fields. The schema uses VARCHAR instead of ENUM for Type and Relationship\_Type for flexibility. CHECK constraints ensure valid data, like Type being 'Requirement,' 'Code,' 'Test Case,' or 'Document.' Next, I will introduce the different layers in the Application Layer.

### B. Controller Layer

The ArtifactController.class handles HTTP requests. The /artifact/add endpoint processes artifact creation. It gets data as an ArtifactDTO.class, uses a hardcoded user ID, and passes it to the service layer. Then, it returns the artifact view with a success or failure message. Here is Figure 6 shows the ArtifactController.class implementation.

电脑屏幕的截图

AI 生成的内容可能不正确。

Figure 6: ArtifactController.class Implementation

The controller keeps feedback smooth by returning to the same view with the right message.

### C. Service Layer

The ArtifactService.class handles artifact creation logic. It does these tasks:

* Check if the Type is valid and if the required fields (like the description for Requirement or file for Code) are filled.
* Creates a unique name for each artifact, like req-01 for Requirement or code-001 for Code, using a counter.
* Turns the ArtifactDTO.class into an Artifact entity, setting fields like createdBy and createdOn.
* For Code artifacts, processes the uploaded file and saves its metadata as the description.

Figure 7 shows the name generation logic. AtomicLong.class ensures thread-safe counters, but a database counter would be better for production to persist across restarts.

文本

AI 生成的内容可能不正确。

Figure 7: Name generation logic

### D. Data Access Layer

MyBatis handles database operations and maps Java methods to SQL queries via XML files. The ArtifactMapper.class interface and ArtifactMapper.xml manage Artifact table operations. The insert method saves new artifacts. Above is Figure 8, which shows the ArtifactMapper XML setup.

文本

AI 生成的内容可能不正确。

Figure 8: ArtifactMapper XML

MyBatis's use GeneratedKeys.class feature grabs the auto-incremented Artifact\_ID after inserting, updating the entity for further use.

### E. Data Extraction Using Regular Expressions

To show the dynamic interaction in traceability information extraction, we provide a

Figure 9 Sequence diagram for the "Data Extraction Using Regular Expressions" process. The diagram depicts the step-by-step interaction among system components and actors and how regular expressions are applied to locate and extract valuable traceability information from sources. The process is clearly described using the following sequence diagram, thus achieving a suitable understanding of the data extraction mechanism via the traceability management system. Next, I will explain each step in the sequence diagram.

图示

AI 生成的内容可能不正确。

**Figure 9:Sequence Diagram**

### Identify the Information to Extract and Its Patterns

Before extracting data, you first need to analyze the text data to identify the key information you want to extract and its patterns. For example, in a requirements management system, suppose the text field in the database contains the following:

* **Requirement ID**: Format is "req-" followed by three digits, e.g., "req-001".
* **Status**: One of "approved," "in progress," or "completed."

For these pieces of information, we need to define corresponding regular expression patterns:

* **Requirement ID Pattern**: req-\d{3}
  + req- is a fixed prefix. To normalize all regular expression patterns, they must be lowercase.
  + \d{3} matches exactly three digits.
* **Status Pattern**: (approved|in progress|completed)
  + Uses | to indicate "or" matching one of these three statuses.

These patterns form the foundation for extraction.

### Retrieves text data from the database through SQL queries

In practice, the system typically retrieves records containing text from the database first. For example, using an SQL query:

**SELECT requirement\_id, description FROM requirements;**

Suppose the description field returns the following text:

* "This requirement (req-001) is approved."
* "req-002 is in progress."
* "No ID here, but it's completed."

These texts will be passed on to regular expressions for processing.

### Apply regular expression to extract the state.

The core step of data extraction involves applying regular expression patterns through a programming language to match and extract the target information. For example, the extracted data can be stored in a Map<String, String>, such as:  
**Map<String, String> extractedData = new HashMap<>();  
extractedData.put("requirement\_id", reqId);  
extractedData.put("status", status);**

If certain fields are not found (e.g., the requirement ID is missing), null can be stored as the value for that key. These incomplete records can then be flagged for further processing or logged for review.

### Batch Processing Multiple Records

When processing a large volume of records from the database, batch processing should be performed on all relevant text fields. In Java, this can be done using a loop combined with regular expressions via **Pattern** and **Matcher.** Always use Pattern.compile() outside of the loop to precompile your regex patterns. This prevents recompiling the same expression on every iteration, significantly improving performance in batch operations.

### Verify the extracted data.

Sometimes, during data extraction, the system finds text that doesn't follow the expected format. For example, a record might not have a requirement ID. In these cases, the system should log a warning.

The system also needs to check the data after it's extracted. This includes:

* Make sure each requirement ID is unique.
* Checking that status values are valid (like "approved," "in progress," or "completed").

If the system finds something wrong, it should clean the data based on rules.

By using the above steps, regular expressions can automatically and effectively extract key information from text. The process includes finding patterns, extracting data with the assistance of regular expressions, batch processing, result processing optimization, exception handling, and data validation. This is particularly helpful for text data, which is not well organized but somehow has patterns, providing a uniform data basis for future traceability management.

## 4.3 Testing and Validation

To make sure the regular expressions used in the system are reliable, we ran several validation tests. These tests helped check how well the system works when dealing with different types of data and possible edge cases.

We didn't just test the basic functions. We also tested for:

* Complex data types
* Unusual cases (edge cases)

These tests help ensure the system is useful and dependable for managing traceability in software development.

### Case Study1: Testing Across Different Artifacts

We first tested how well the regular expressions work with different data sources. In real projects, traceability data isn't always stored in one place. It can be found in:

* Requirement documents
* Design tables
* UML diagram descriptions
* Code files
* Even image annotations

To test this, we created different types of sample data. For example, a requirements document might include a sentence like:  
"This requirement (req-001) is approved." while design tables could include descriptions such as "design des-001 specifies the database schema." For UML diagrams, we assumed description text like "Sequence diagram includes tc-001 for user login" and code files might have comments like "Function implements req-002, completed on 2025-04-01." More complex cases involved data embedded in images, such as a design diagram labeled "req-003: Data Flow Diagram." To handle this, we first used optical character recognition (OCR) technology to extract text from images before applying regular expressions. The specific steps involved feeding this data into the system and using tailored regular expression patterns—req-\d{3} for requirement IDs, des-\d{3} for design IDs, tc-\d{3} for test case IDs, and (approved|in progress|completed) for status—to extract key information step-by-step. We compared the extracted results with expected values; for instance, "req-003" extracted from an image should match manual annotations, and "req-002" and "completed" from code comments should be fully retained. The experiments showed that the system accurately identified target information in textual data. For image-based data, OCR accuracy influenced the results, but the regular expressions themselves performed consistently in subsequent extraction.

### Case Study 2: Impact of Case Sensitivity on Extraction Results

Next, we tested the impact of case variations in IDs on extraction outcomes. In real-world development, ID formats may differ due to manual input, such as "req-001," "REQ-001," or "Req-001." To assess the robustness of the regular expressions, we prepared test data with these variants and evaluated two approaches: a case-sensitive pattern req-\d{3} and a case-insensitive pattern using the re.IGNORECASE flag, i.e., re. search(r'req-\d{3}', text, re.IGNORECASE). The testing process involved feeding this data into the system, extracting IDs, and checking the matches. In case-insensitive mode, the system extracted all variants and standardized them as "req-001," but this risked false positives, such as recognizing "REQUEST-001" or "req-001a" as valid IDs. In case-sensitive mode, only "req-001" fully matching req-\d{3} was extracted, while variants like "REQ-001" or "Req-001" returned None. To further validate accuracy, we introduced noisy data like "REQ-001x" or "request-001." Results showed that the case-sensitive mode more strictly filtered the target format, avoiding false positives, while the case-insensitive mode was more lenient. Comparative analysis revealed that the case-sensitive mode achieved higher accuracy with a false positive rate near 0, compared to approximately 5% for the case-insensitive mode. Thus, we concluded that the case-sensitive mode better suits the system's needs. To ensure consistency and accuracy, we recommend standardizing data entry in lowercase (e.g., "req-001") and strictly using req-\d{3} in regular expressions.

## 4.4 Result and Discussion

The Traceability Management System (TMS) successfully takes user input and returns and displays traceability information. When a user submits a query, the system shows key details about an artifact. This includes the artifact ID, creation time, content, creator, last update time, and its links to other items. **Table 6** shows different queries and how the system responded. They show how TMS can quickly pull and display traceable information.

**Table 6: Queries Input and Output**

| **Input** | **Ouput** |
| --- | --- |
| Requirement ID ("req-001") | Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser(), Creator: Zhang San, Modification Time: 2025-01-02 15:30 |
| Requirement Description ("user login functionality") | Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser() implements requirementID=req-001, Creator: Zhang San, Modification Time: 2025-01-02 15:30 |
| Artifact Type ("function") | Matching Artifacts: Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser(), Creator: Zhang San, Modification Time: 2025-01-02 15:30 |
| Project ID ("pro-001") | Project Artifacts: Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser(), Creator: Zhang San, Modification Time: 2025-01-02 15:30, Relationship: req-001→fun-001 |
| Query Condition ("type=function AND requirementID=req-001") | Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser(), Creator: Zhang San, Modification Time: 2025-01-02 15:30 |
| No Input (Default: Show All Requirements) | Requirement ID: req-001, Artifact ID: fun-001, Creation Time: 2025-01-01 10:00, Content: function loginUser(), Creator: Zhang San, Modification Time: 2025-01-02 15:30, Relationship Graph: req-001→fun-001→tes-001 |

It was built to pull and display traceability data from a MySQL database. This database holds 50 software artifacts from three projects. These artifacts included requirements, classes, functions, components, and diagrams. The system follows the layered structure shown in Figure 1.

We measured performance using two metrics:

### Data Extraction Efficiency

On average, it took **0.5 seconds** to run a single query (like getting all functions for one class). The system could extract all 50 artifacts in under **2 seconds total**. This shows the backend handles various queries quickly, even when dealing with different types of artifacts.

### Key Findings

* The TMS pulled data quickly using regular expressions. It worked well with different kinds of artifacts like functions, diagrams, and requirements. There were no delays in the process.
* The graph feature performed smoothly. It created graphs in about **1.2 seconds**, which let users explore data instantly. When users selected new items (like switching from one class to another), the charts updated right away.
* In one example graph, 50 nodes were shown. A requirement node was linked to a function node (which belongs to a class). That class was also linked to a test case. This showed how the system can display complex relationships across different artifact types.

It uses regular expressions to find items like requirements, classes, functions, and diagrams. These are then shown in real-time charts to show how they are connected.

For example:

* The pattern ^function.\*classID=[A-Za-z0-9]+$ finds functions linked to a class.
* The pattern ^requirement.\*projectID=[A-Za-z0-9]+$ finds requirements within a project.

We tested the system with 50 artifacts across three projects. Each graph took around **1.2 seconds** to load. This shows the system works well with small to medium data sets.

The interface updates automatically. For example, selecting a class pulls in all linked functions which appear in the graph. This saves users from updating the graphs by hand.

Basic actions like creating or editing artifacts take about **0.8 seconds**. This helps the system stay fast and responsive.

The test used a small dataset of 50 artifacts. This may not show how the system handles large projects with hundreds or thousands of artifacts. Real software projects are often much bigger. Issues could come up with database queries, chart rendering, or regular expression matching. For example, chart generation takes 1.2 seconds now, but with more nodes, this could increase a lot, especially for complex traceability links. Future tests should use larger datasets to check scalability. Adding index optimization or caching could help reduce delays.

TMS isn't just for software. It can work in other fields like systems engineering or product lifecycle management. In industries like aerospace or automotive, tracing requirements, designs, and test cases is critical. The charting feature can show these links clearly. However, it needs customization to fit different artifact types and standards in different fields.

The system's response time is 0.8 seconds for basic tasks, which is good. Still, there's room to improve the user experience. An interactive filter could let users adjust charts based on project ID or artifact type. Also, better visual design, like color-coding key paths or grouping nodes, could make charts easier to read.

# Chapter 5 Conclusion

The system project implemented a TMS that retrieves database information to create interactive live charts. The system uses regular expressions to extract requirements and functions from the data and presents their relationships through Thymeleaf.

The system builds a graph from 50 artifacts during a 1.2-second timeframe. The system operates efficiently for projects with moderate to small sizes. Real-time updates appear without needing any page reloads or graphical reconstruction.

The system provides users with enhanced capabilities to handle traceability functions. The system allows teams to track item connections so they can make informed choices. The system provides strong foundations for future improvements yet experiences performance delays when working with extensive data and occasional issues with inconsistent data formats.

## 5.1 Implications for Traceability Management

Through the TMS, users can handle traceability data with simplicity. The system retrieves various database items, such as requirements and functions, as well as test cases to create an interconnected graph.

Developers gain a complete understanding through this system. Developers can effortlessly observe the connection between requirements and their associated components and test cases.

Real-time graph updates enable users to receive immediate feedback. Users do not require any redraws. The system proves beneficial for quick projects because users need continuous updates.

The system enables complete traceability from requirements to tests, which facilitates easier management of big projects.

## 5.2 Limitations and Challenges

The system demonstrates effective functionality, although it has several limitations.

* **Scalability**: It handles 50 artifacts quickly, but large datasets may slow it down.
* **Data Quality**: It depends on clean, consistent data. Names and formats that do not follow standard conventions might result in missed linkages between artifacts.
* **Limits of Regular Expressions**: They're great for structured data but not for complex or messy formats.

## 5.3 Future Work

The system requires improvements through the following steps:

* **Make it scale better**: Indexing and caching should be added to make the system handle big datasets more quickly.
* **Improve data extraction**: Regular expressions should integrate with machine learning methods to enhance unstructured data handling capabilities.
* **Add more chart types**: The system should include heat maps and tree diagrams as additional chart options to expand user visualization possibilities.

# Bibliography

[1] N. Mustafa, "Using semantic web to establish traceability links between heterogeneous artifacts," in Proc. 13th Int. Conf. Softw. Technol., Porto, Portugal, Jul. 2018, pp. 305–312, doi: 10.5220/0006861403050312.

[2] Author(s), "Traceability management with impact analysis in DevOps based software development," in Proc. IEEE Int. Conf. Softw. Eng., 2018, pp. 1–10, doi: 10.1109/8554399.

[3] "A visual framework for software requirements traceability," ResearchGate, Dec. 2024, doi: 10.11591/eei.v11i1.3269.

[4] F. Tian, T. Wang, P. Liang, C. Wang, A. A. Khan, and M. A. Babar, "The impact of traceability on software maintenance and evolution: A mapping study," J. Softw. Evol. Process, vol. 33, no. 10, Aug. 2021, doi: 10.1002/smr.2374.

[5] S. Charalampidou, A. Ampatzoglou, E. Karountzos, and P. Avgeriou, "Empirical studies on software traceability: A mapping study," J. Softw. Evol. Process, vol. 33, no. 2, Feb. 2021, doi: 10.1002/smr.2294.

[6] Z. Chen, C. Jia, Y. Li, and L. Chen, "Using dynamic and static techniques to establish traceability links between production code and test code on Python projects: A replication study," J. Softw. Evol. Process, vol. 37, no. 3, Mar. 2023, doi: 10.1002/smr.70011.

[7] D. Fucci, E. Alégroth, and T. Axelsson, "When traceability goes awry: An industrial experience report," J. Syst. Softw., vol. 192, Oct. 2022, Art. no. 111389, doi: 10.1016/j.jss.2022.111389.

[8] R. Rasiman, F. Dalpiaz, and S. España, "How effective is automated trace link recovery in model-driven development?," in Proc. 28th Int. Working Conf. Requirements Eng.: Found. Softw. Quality, Aston, UK, Mar. 2022, pp. 35–51, doi: 10.1007/978-3-030-98464-9\_4.

[9] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Enhancing automated software traceability by transfer learning from open-world data," arXiv:2207.01084, Jul. 2022. [Online]. Available: https://arxiv.org/abs/2207.01084, doi: 10.48550/arXiv.2207.01084.

[10] T.-B. Du, G.-H. Shen, Z.-Q. Huang, Y.-S. Yu, and D.-X. Wu, "Automatic traceability link recovery via active learning," Frontiers Inf. Technol. Electron. Eng., vol. 21, no. 8, pp. 1217–1225, Aug. 2020, doi: 10.1631/FITEE.1900222.

[11] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in Proc. 28th Int. Conf. Program Comprehension, Seoul, South Korea, Jul. 2020, pp. 248–258, doi: 10.1145/3387904.3389251.

[12] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of requirements engineering in software product lines," Inf. Softw. Technol., vol. 80, pp. 135–157, Dec. 2016, doi: 10.1016/j.infsof.2016.08.009.

[13] G. E. Okudan and C. Budayan, "Challenges in tools interoperability for heterogeneous systems," in Proc. IEEE Int. Conf. Softw. Maintenance Evol., Limassol, Cyprus, Oct. 2022, pp. 88–96, doi: 10.1109/ICSME55016.2022.00015.

[14] Q. Li, X. Li, and L. Wang, "A multi-layer approach for automatic traceability link retrieval," Inf. Softw. Technol., vol. 131, Mar. 2021, Art. no. 106488, doi: 10.1016/j.infsof.2020.106488.

[15] R. Wohlrab, E. Knauss, J.-P. Steghöfer, S. Maro, A. Anjorin, and P. Pelliccione, "Collaborative traceability management: A multiple case study from the perspectives of organization, process, and culture," Requirements Eng., vol. 25, no. 1, pp. 21–45, Mar. 2020, doi: 10.1007/s00766-018-0306-1.

[16] M. Kassab, O. Ormandjieva, M. Daneva, and G. Ruhe, "Early industrial evaluation of a big data analytics tool for traceability in RE," J. Syst. Softw., vol. 117, pp. 225–239, Jul. 2016, doi: 10.1016/j.jss.2016.03.048.

[17] J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Grand challenges of traceability: The next ten years," in Proc. 39th IEEE/ACM Int. Conf. Softw. Eng., Buenos Aires, Argentina, May 2017, pp. 17–24, doi: 10.1109/ICSE-C.2017.18.

[18] R. Dzhusupova, R. Banotra, J. Bosch, and H. H. Olsson, "Using artificial intelligence to find design errors in the engineering drawings," *Journal of Software Evolution and Process*, vol. 35, no. 12, Feb. 2023, doi: 10.1002/smr.2543.

[19] P. Mäder and O. Gotel, "Towards automated traceability maintenance," J. Syst. Softw., vol. 85, no. 10, pp. 2205–2227, Oct. 2012, doi: 10.1016/j.jss.2011.10.023.

[20] D. A. Meedeniya, I. D, and I. Perera, "Traceability Establishment and Visualization of software artefacts in DevOps practice: a survey," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 7, Jan. 2019, doi: 10.14569/ijacsa.2019.0100711.