JOIV : Int. J. Inform. Visualization, 8(3-2): IT for Global Goals: Building a Sustainable Tomorrow - November 2024 1849-1860

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Problem-Frame-Oriented Requirements Traceability to Enhance Requirements Management

Xiao ShengWen [a,*], Sa'adah Hassan [a], Noraini Che Pa [a]

[a] *Department of Software Engineering and Information Systems, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia*

Corresponding author: [*]gs64797@student.upm.edu.my

***Abstract*—Managing software requirements is a challenge in software development and maintenance. Requirements changes are inevitable, particularly in a rapid iterative development approach that leads to occasional changes in software requirements. Unable to manage this properly will impact the overall quality of the software. Thus, requirements traceability is essential because it ensures that all requirements are adequately addressed, changes are managed effectively, and that there's a clear linkage between business requirements and the system's functionality. Inadequate traceability mechanisms can make changing the requirements and detecting their impact difficult. Thus, it is crucial to establish precise requirements traceability and maintain clear links to manage the requirement changes effectively. Our research explores using a problem frames modeling approach to address this issue. It starts by representing requirements as problems, creating a requirements relationship diagram, and generating a corresponding relationship matrix. The values in the traceability matrix help identify which elements are most affected by requirement changes, allowing developers to prioritize changes that minimize overall system impact. Furthermore, using problem frame modeling, complex problems can be broken down into manageable sub-problems, providing a clear structure for understanding the requirements. Additionally, a tool has been created to streamline the process, and a case study is used to demonstrate the functionalities. An evaluation has been conducted to assess the usability of the proposed work. The requirements relationship diagrams and relationship matrices visually and quantitatively map the links between requirements, enabling traceability and identifying the impact of changes in requirements.**

***Keywords*— Requirements traceability; traceability matrix; problem-frame; requirements engineering.**

Manuscript received 15 Mar. 2024; revised 7 Jun. 2024; accepted 12 Sep. 2024. Date of publication 30 Nov. 2024.

I. INTRODUCTION

In software development, requirements must often be adjusted due to changing user needs, technological advancements, and market trends. To manage these changes, traceability links connect different artifacts and ensure that updates in one are reflected in others. However, maintaining requirements traceability in software development can be challenging, impacting the overall quality of the software and the development process. Ineffective traceability can lead to inconsistencies, misunderstood requirements, increased development costs, and uncontrolled project scaling.

Various methods and tools have been suggested to tackle these issues, but they persist, particularly in large-scale software projects and more complex software environments. Manual tracing requirements are both labor-intensive and error-prone. In addition, some of the proposed tools could be more complex and costly. For instance, tools like IBM DOORS and Jama Connect need help to keep up with the frequent changes in requirements in agile development. While traceability methods, such as Natural Language Processing (NLP), aim to overcome these limitations. A study by Laliberte *et al*. [1] suggests that NLP is likely not a practical approach to requirements traceability.

This paper presents a framework for enhancing the effectiveness of requirements traceability in software development projects. It utilizes problem-framing and advanced modeling techniques to automate traceability and provide a structured framework for identifying and resolving requirements conflicts. The proposed framework aims to improve the software product's requirements management and overall quality. Initially, the requirements are structured using the problem frames, and corresponding tracing relationship diagrams are created to show the correlations clearly. Matrix operations are then used to trace the requirements.

This paper is organized as follows: Section II explores the existing related work in the field and thoroughly explains the

proposed framework. Section III presents a developed tool as proof-of-concept and uses a case study to demonstrate its functionalities. The usability of the tool is also evaluated and discussed. Finally, the paper is concluded in Section IV.

II. Materials and Method

This section aims to provide a comprehensive review of key areas of research in this research: evolving practices and challenges in requirements traceability, requirements management in software development, requirements modeling and application based on problem framing, and previous attempts and approaches in requirements management and traceability tools. This review synthesizes these interrelated areas, providing insights into current trends and future research directions while also defining essential terms and concepts related to these areas.

A. Requirements Traceability

Requirements traceability is critical to the successful development and management of complex projects, and current practice is dominated by manual traceability methods supplemented by tools such as IBM DOORS and Jama Connect, which provide user-friendly interfaces for managing requirements. Although these manual methods are widely used, they can be labor-intensive and error-prone, especially in large projects. According to Jayatilleke et al. [3], most of the automation techniques for the traceability process use information retrieval (IR) techniques, which are mainly based on natural language processing (NLP) such as term frequency-inverse document frequency (TF-IDF) and latent semantic indexing (LSI). However, these automated NLP methods have limitations in capturing the deeper meaning behind the requirements, resulting in high errors.

In web application development, tight schedules and complex designs often accompany requirements. Due to the dynamic nature of web applications, which change rapidly, traceability becomes even more complicated, according to Lyu *et al.* [4]. Comprehensive development guidelines and tools are often needed to manage requirements traceability effectively in web development. This usually leads to difficulties for developers in understanding the impact of changed requirements and the backward and forward relationships between different requirements. Forward traceability involves tracing from requirements to system design and beyond. Backward traceability focuses on tracing back elements, such as system design, to the original requirements.

The study highlights that in a global software development (GSD) environment [5], managing and keeping requirements traceability effectively becomes more challenging due to geographic dispersion and diverse stakeholder backgrounds. These factors may lead to ambiguous and incomplete requirements documentation, affecting the overall quality of the software. Traditional approaches often rely on manual methods or semi-automated systems, which, while beneficial for smaller projects, become increasingly impractical in more extensive and complex GSD environments. Managing and maintaining various software artifacts such as requirements specifications, design specifications, source code, and test cases takes a lot of work. These artifacts evolve independently at different rates, leading to inconsistency and disjointedness. Kamalabalan et al. [6] state that the current practice utilizes traceability links to connect these different artifacts, thus ensuring that changes in one artifact are reflected in the others. However, creating these links manually is both labor-intensive and error-prone. Mezghani et al. [7] state that traceability is transferred to tool support to address these shortcomings. These include Application Lifecycle Management (ALM) tools, which provide a holistic management approach that covers the entire system development lifecycle, facilitating the management and analysis of artifacts. Requirements management tools are then used, which integrate management system requirements for more effective traceability.

For the automated requirement traceability aspect, work by Kchaou *et al.* [8] pointed out that semantic models can be utilized with advanced natural language processing techniques. The model transcends language barriers and can analyze the semantics in a sentence, automate the comparison of similarities between words that serve the same purpose as the requirement changes, and provide a more efficient approach.

Automated traceability of requirements can be achieved through a requirements traceability matrix. However, traditional requirement traceability matrices are usually static and inflexible and need to be more suitable for the dynamic and iterative nature of requirements in agile development. The Agile Requirement Traceability Matrix (ARTM) proposed by Jeong *et al.* [2] solves this problem. Traceability mapping between artifacts is managed through a spreadsheet format, which automatically generates an up-to-date Requirement Traceability Matrix to automate the tracing of requirements.

Automatic requirement traceability through matrix operations. Yinghui et al. [9] proposed using the reachability matrix in the evolution of software architecture, creating a software architecture model through semantic relationships and a relationship matrix and reachability matrix based on the SA model. Yinghui *et al.* [10] used the relationship matrix multiplication operation for software requirement traceability.

Requirements traceability is diverse, and standardization and diversification of traceability practices become particularly important, as some organizations may rely on manual methods using spreadsheets. In contrast, others may use more automated systems. This difference in approach hinders the uniformity of requirements traceability in software development and the migration of requirements across organizations. Marques *et al.* [11] propose a standardized approach to RT processes, particularly in defining critical aspects of these processes and establishing clear roles and responsibilities in traceability development.

Several studies have explored the application of different requirement-tracing techniques in real-world environments. Rajbhoj *et al.* [12] applied a demand management approach and traceability tool to the context of railway operations in formal system modeling. The proposed VisualisierbaR tool integrates interactive visualization and requirements traceability. The tool helps to bridge the gap between formal modeling and its practical application. Requirements elicitation and scoping are particularly important in this process, where the implicit knowledge and assumptions of the domain of interest are transformed into explicit representations, and the model's scope is clearly defined.

Requirements traceability links parts of the formal system model to the target system. This process involves forward tracing (linking requirements to model components) and backward tracing (determining the origin of model components in requirements). This bi-directional traceability ensures the accuracy of every model aspect and helps validate the model's structure and behavior against the expected requirements. Moreover, VisualisierbaR's interactive visualization capabilities allow dynamic exploration and validation of the model through user interaction. This integrated approach to requirements management and traceability significantly reduces the cognitive burden on the user and makes the validation process more efficient.

B. Requirements Modeling based on Problem Frames

Problem frames focus on splitting complex problems encountered in real life into several suitable sub-problems. It focuses on real-world problems from which the user's needs are understood, revealing the real-world issues that software can solve. Lavazza et al. [13] highlight conceptual clarity between User Needs, User Requirements, and Business Objectives; there needs to be more clarity between these concepts. User needs are distinct from user requirements, although the two are interrelated. Kannan [14] proposes a methodology for establishing hierarchical relationships between these elements, using problem framing to model and analyze them effectively. He also shows how requirements can be represented at different levels of abstraction and how these levels can be aligned with the problem framework model.

The efficiency of requirements engineering depends heavily on the analyst's experience and educational background. The DRAP-PF methodology [14] combines the problem framing (PF) concept with an analysis model to simplify eliciting, analyzing, and specifying software requirements. By decomposing the environment into different problem domains, stakeholders can identify requirements more clearly. The traditional PF approach focuses on viewing software as a means of solving real-world problems. This approach emphasizes understanding the constraints and context of the issue at hand.

However, the ability to dynamically adapt to changing requirements still exists. In the work by Liu and Jin [15], the goal-oriented approach, represented by the I* framework, starts from an understanding of the high-level goals of stakeholders and captures the social and organizational aspects critical to software design. The problem framework lacks the granularity to express detailed data descriptions, essential to understanding complex system requirements. Xie, Xiao, and Li [33] propose the integration of ChatGPT with the PF. This integration enhances the details of the data by allowing the description of shared phenomena in problem diagrams. The approach consists of creating a table of causes through interaction with ChatGPT to identify causal relationships in these phenomena. Subsequently, these causes are ranked and used to extend the fuzzy phenomena in the problem diagram. Tekutov and Smirnova [29] proposed work using a problem domain model for enhancing and assessing system requirements in a computing education context. Their work emphasizes stakeholder involvement in problem modeling.

C. Requirements Change and Conflict Management in Software Development

Influenced by innovative systems such as cloud computing and microservices architectures, software application development has high scalability and flexibility [17]. Changing requirements are inherent to software development and are usually driven by changing user needs, technological advances, or shifts in market trends. These changes are dynamic and require developers to adopt a flexible and agile approach to adapt to these changes without severely impacting project schedules or increasing costs. Conflict management stems from stakeholders' differing perspectives, priorities, and understanding of project goals. This requires effective communication strategies and collaborative decision-making processes to resolve conflicts.

Dynamic changes in requirements in agile software development environments often stem from changing user needs, technological advances, and regulatory updates [18]. Conflicts regarding requirements arise when integrating various resources and may affect other non-functional requirements. It is also mentioned that Agile methodologies and continuous integration practices are used to respond to changes in requirements and ensure that software applications remain relevant and functional in a rapidly evolving digital environment [19]. In addition, a comprehensive testing regime ensures that changes do not adversely affect the application's performance or the user experience. Work by Mustafa et al. [20] highlights that distributed teams using a hybrid Scrum-XP methodology can better deal with changes in requirements and conflicting requirements. The iterative nature of Scrum, combined with XP's emphasis on continual feedback and adaptation The iterative nature of Scrum, combined with XP's emphasis on continuous feedback and adaptability, helps to effectively manage changing requirements and ensure that the final product meets stakeholder expectations and market needs.

Bukhari *et al*. [21] propose a structured approach to requirements prioritization and conflict management in Web development. The framework uses value-oriented prioritization (VOP) principles to manage and prioritize requirements effectively. These requirements are then assigned weights, and a prioritization matrix is constructed. Finally, the matrix is used as a decision-making tool to help resolve conflicts. Changes in requirements in Web development can lead to disputes in project scope, schedule, and resource allocation, which can affect the management and delivery of the overall project. The Web Complexity Factor (WCF) model proposed by Saif & Wahid [22] addresses this challenge by providing a more nuanced and comprehensive scale measure. Focusing on GSD environments, changes in requirements, and conflict management in web application development have become more complex. Alsanad *et al*. [23] proposed a system domain ontology for requirements change management (RCM), which employs a hybrid approach combining methodology and 101 methodologies and is represented using web ontology language (OWL). While larger organizations with more resources tend to develop more structured processes to manage requirement changes and conflicts using formal methods and tools [24]. Conversely, smaller companies tend to rely on more agile and flexible approaches. This dichotomy reflects that

organizations of different sizes have different needs and capabilities in dealing with the fluidity of web application development.

D. Requirements Management and Traceability Tools

A study conducted by Tian et al. [25] highlights 13 approaches to requirements management and traceability tools. These include information retrieval (IR)-based approaches, feature model-based approaches, scenario-based approaches, tactic and decision-based approaches, and constraint-based approaches. These approaches address different aspects of requirements traceability, ranging from automatic generation of traceability links to improved understanding and management of software system changes. However, there are significant gaps in the integration and effectiveness of these tools in industrial environments, and further validation in real-world software development scenarios is required. In addition, a study by Tufail *et al.* [26] identified 33 relevant studies, thus identifying seven requirements traceability models, ten challenges, and 14 tools in the field. Among the tools mentioned are ECOLABOR, TOOR, RESAT, POIROT, CREWS-EVE, ProR, Trace Analyzer, TRIC, ADAMS, SCOTCH+, Trace Maintainer, DOORS, RequisitePro, and RETRO. These tools use various methods to maintain traceability links between software development artifacts, with DOORS being particularly effective.

Applications in complex systems development environments. To address the difficulties posed by manual traceability, by integrating model-based systems engineering (MBSE) tools to automate traceability by creating digital threads, thus providing dynamic requirements management for stakeholders to manage requirements more comprehensively and integrated. For example, Escalona *et al.* [27] provide integration of requirements management methodologies and traceability tools within a Model Driven Engineering (MDE) framework. This approach utilizes automated processes to facilitate requirements traceability, maintain quality, and manage change. Wang *et al.* [28] To reduce the gap between Requirements Analysis and Design Language (AADL) models, an intermediate model, RAInterM (RM-RNL to AADL Intermediate Model), was introduced. This strategy simplifies the conversion process, ensures compatibility, and reduces complexity.

In addition, SAT-Analyzer, a semi-automatic tool, enhances the creation and visualization of traceability links of artifacts. The tool utilizes advanced technologies such as Neo4j for graphical database management, allowing efficient handling of large datasets and relationships. Besides, it combines NLP and machine learning (ML) techniques to extract and analyze information in artifacts to facilitate the semi-automatic identification of traceability links. DizSpec [12] is an automated methodology that transforms requirements specification documents into a modeled form to facilitate traceability and impact analysis in the IT services industry. The methodology integrates techniques such as meta-modeling, model extraction, and dependency extraction and is based on the application of artificial intelligence in the software development life cycle (SDLC) and NLP. The tool transforms documents into machine-processable models to more effectively manage and analyze product functions, processes, activities, rules, parameters, and interdependencies.

Zhang *et al.* [34] introduced the Information Retrieval Based Requirements Traceability (IRRT) tool. The tool generates requirements traceability links using the Vector Space Model (VSM), a recognized model in the field of information retrieval for converting software artifacts, such as requirements documents and source code, into a vector form to compute the similarity of the documents. In addition, the tool refines these links using the Traceability Recommendation Code Class Structure (TRCCS) to improve the accuracy of traceability links. The Capra tool [30] addresses the variability and specificity of traceability requirements in different projects and organizations. It can create traceability links for any artifact. In addition, Capra allows project managers to customize traceability link types to meet the unique needs of each project.

This section outlines our proposed framework based on the problem-framing approach. Fig. 1 illustrates the steps involved in the problem-frame-oriented requirement traceability framework. In brief, problem-frame modeling breaks down a complex problem into simpler sub-problems, then modeled individually. Once the modeling is complete, the results are matched to the requirement correlation matrix. Matrix operations are then performed, and the traceability of the requirement correlation matrix is carried out. Finally, the impact of any changes can be assessed based on the matrix.

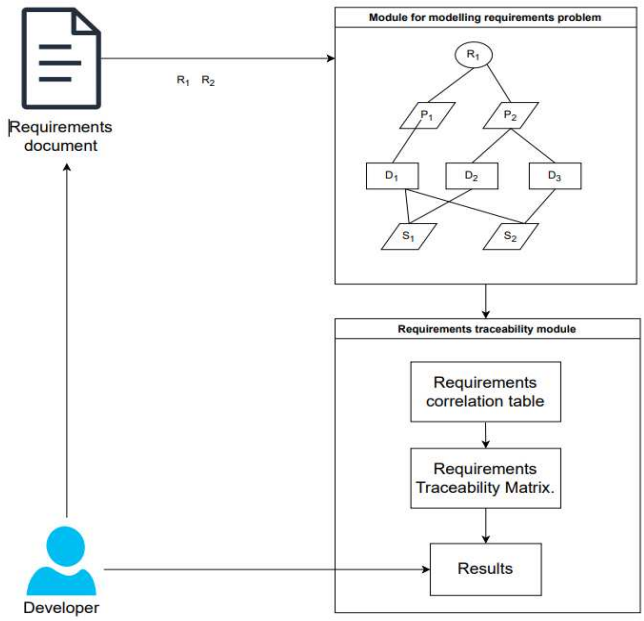

Fig. 1 The proposed framework

E. Module for Modeling Requirements

This study bases requirements modeling on problem frames. Problem framing mainly involves splitting complex problems encountered in real life into several suitable sub-problems [31]. The work primarily starts from the perspective of real-world problems, from which the user's needs are understood, and the real-world problems that the software can solve are revealed. Unlike traditional object-oriented modeling ideas, the problem frame is problem-oriented. Therefore, we use problem diagrams to model requirements,

as shown in Fig. 2. A problem diagram consists of domains, machines, and problems connected by cause-and-effect relationships, which allow for a more comprehensive description of the problem.

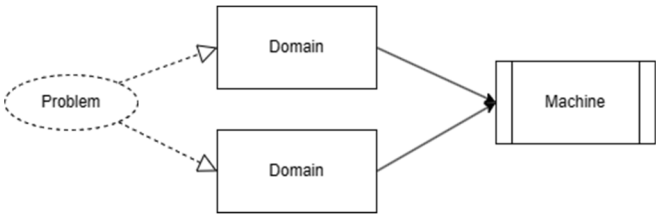

Fig. 2 Structure of the problem diagram

The problem diagram is structured as follows.

- Problem: A problem that needs to be solved in a particular context. Problems and domains are connected by dotted lines and act as constraints on the connected domains in a given situation.
- Domain: The problem framework uses the domain as one of the central representations of the real world. A rectangle in the problem graph represents it. It refers to a set of devices or people associated with a software system.
- Machine: a device that implements software-related functions in the real world. A rectangle with double vertical lines in the problem diagram represents it.

In problem-based requirements modeling, the process begins with identifying a complex real-world problem and then documenting its requirements. Since real-life problems are often not simple, the problem at hand can combine multiple simple problems. Therefore, it is beneficial to create a problem diagram by breaking down the complex problem into simpler ones. This approach makes it easier for the analyst to analyze the problem. Once relevant sub-problem diagrams are created, corresponding problem diagram modeling is conducted. The key step involves abstracting real-world entities into domains and establishing connections between the domains and the software to be developed. Finally, a requirements traceability route diagram is generated, as depicted in Fig. 3.

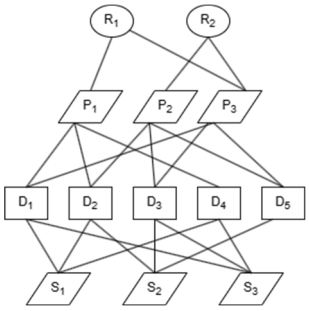

Fig. 3 Sample Requirements Traceability Route Diagram

As shown in Fig. 3, Requirement R1 corresponds to two Problem Diagrams, P1 and P3, and Requirement R2 corresponds to two Problem Diagrams, P2 and P3. Problem Diagram P1 corresponds to three domains, D1, D2, D4, and Problem Diagram P2 corresponds to three domains, D2, D3, and D5, and Problem Diagram P3 corresponds to three domains, D1, D3, and D5, respectively. The final realization of the program set response is the goal to be achieved. Domains D1, D2, and D5 correspond to program set S1, domains D2, D3, and D5 correspond to program set S2, and domains D1, D3, and D4 correspond to program set S3.

The corresponding requirement traceability route can be found in the requirement traceability relationship diagram, such as from $R_1 \rightarrow P_1 \rightarrow D_2 \rightarrow S_2$ or $R_1 \rightarrow P_3 \rightarrow D_1 \rightarrow S_1$, etc. However, if a requirement traceability relationship diagram is large and complex enough, it is hard for the software developer or requirements engineer to understand the relationship clearly. Thus, the relationship in the diagram can be quantized by creating a relationship matrix.

F. Requirements Traceability Module

To ensure that software products and requirements are consistent in software development, it is necessary to conduct two-way requirements traceability, i.e., forward traceability and backward traceability. Forward tracing allows the developer involved in requirements tracing to clearly understand each stage of the requirements and the correlations in the development process. Firstly, the requirements are traced to the problem diagram, then the problem diagram is traced to the machine domain or design domain, and finally, the domain is traced to the program code. According to the requirement trace case diagram, when a requirement R changes, it is possible to know which related assembly S has changed. Reverse tracing is knowing which requirement changes will eventually be affected by a change in the traced software artifact. As can be seen from the traceability schematic, changes in requirements (R) during development affect changes in problem (P), changes in problem P affect changes in domain (D), and changes in domain D affect changes in the program set (S) for that domain. Ultimately, changes in requirements R affect changes in the target program set.

Automated requirements traceability is performed using a requirements traceability matrix. Firstly, the problem framework is used to build a requirements relationship table, a semantic representation of the relationship between elements in each layer, where "1" marks the relationship between elements and "0" marks no relationship between elements. As a result, the traceability relationships of requirement-problem, problem-domain, and domain-assembly are obtained. Finally, a relationship matrix is built from the traceability relationship table, and the matrix multiplication result judges each element's relevance. The elements are marked with "non-zero" to indicate that they are related and "zero" to indicate that they are not related.

Based on the requirement traceability chain presented in Fig. 3, this paper creates, by way of example, a table of associations between a user requirement R and a problem diagram P, a problem diagram P and a domain D, and a domain D and a related procedure S, respectively.

TABLE I
CORRELATIONS BETWEEN REQUIREMENTS AND PROBLEM DIAGRAMS

	P_1	P_2	P_3
R_1	1	0	1
R2	0	1	1

TABLE II
CORRELATIONS BETWEEN THE PROBLEM DIAGRAM AND THE DOMAINS

	D_1	D_2	D_3	D_4	D_5
P_1	1	1	0	1	0
P_2	0	1	1	0	1
P_3	1	0	1	0	1

TABLE III
ASSOCIATIONS BETWEEN DOMAINS AND ASSEMBLIES

	S_1	S_2	S_3
D_1	1	0	1
D_2	1	1	0
D_3	0	1	1
D_4	1	0	1
D_5	0	1	0

Creating a Requirements Traceability Matrix is based on the Requirements Correlation Table, the basis for mapping the relationships between different software system elements. For example, by using Table 1, which lists the relationships between Requirements (R) and Problem Diagrams (P), we can construct a Requirements and Problem Diagram Relationship Matrix (M_{RP}). This matrix visually represents the connection between each requirement and the various problem diagrams:

$$M_{RP}=[1\ 0\ 1\ 0\ 1\ 1]$$

The relationship matrix between need R and problem diagram P shows which problem diagram P the need R correlates with. If the need R changes, the problem diagram P with the correlation will also change. Continuing from Table 2, we can derive the relationship matrix (M_{PD}) between Problem Diagrams (P) and Domains (D). This matrix outlines the connections between different problem diagrams and the specific domains they influence:

$$M_{PD}=[1\ 1\ \ 0\ 1\ 0\ \ 0\ 1\ \ 1\ 0\ \ \ 1\ 0\ 1\ \ 1\ 0\ 1\ \ ]$$

This matrix indicates how changes in a problem diagram can impact the associated domains. For example, if a problem diagram P is altered, the corresponding domain D linked to it will also be affected. Next, we look at the relationship matrix between Domains (D) and Program Sets (S), as outlined in Table 3. This matrix (M_{DS}) provides a clear view of how domains are connected to specific program sets:

$$M_{DS}=[1\ 0\ 1\ \ 1\ 0\ 1\ 0\ \ \ 1\ 1\ 0\ 1\ \ 0\ 1\ 1\ 0\ \ \ ]$$

To achieve comprehensive traceability, we can perform matrix multiplication to combine the insights from the MRP and MPD matrices, resulting in the M_{RD} matrix:

$$M_{RD}=M_{RP}\times M_{PD}=[1\ 0\ 1\ 0\ 1\ 1]\times$$
$$[1\ 1\ \ 0\ 1\ 0\ \ 0\ 1\ \ 1\ 0\ \ \ 1\ 0\ 1\ \ 1\ 0\ 1\ \ ]=$$
$$[2\ 1\ 1\ 1\ 1\ \ \ 1\ 1\ 2\ 0\ 2\ \ ]$$

The resulting MRD matrix allows us to visualize the relationships between requirements and domains, with non-zero values indicating a direct connection. The larger the value in the matrix, the stronger the correlation between the requirement and the domain. Continuing this process, we can multiply the MRD matrix with the MDS matrix to obtain the final Requirements and Program Sets Relationship Matrix (MRS):

$$M_{RS}=M_{RD}\times M_{DS}=[2\ 1\ 1\ 1\ 1\ \ \ 1\ 1\ 2\ 0\ 2\ \ ]\times$$
$$[1\ 0\ 1\ \ 1\ 0\ 1\ 0\ \ \ 1\ 1\ 0\ 1\ \ 0\ 1\ 1\ 0\ \ \ ]=[4\ 3\ 4\ \ 2\ 5\ 3\ ]$$

This matrix provides a comprehensive view of how requirements influence specific program sets, enabling both forward and reverse traceability. For example, from the MRS matrix, we can see that requirement R2 has a greater impact on the program set S2 than R1 does, guiding developers in managing changes with minimal disruption. The correlation matrix MRS represents the relationships between requirements (R) and program sets (S) within a given system. Each element Mij in the matrix denotes the strength of the association between requirement Ri and program set Sj. The matrix is as follows: MRS = [4 3 4 2 5 3].

This matrix quantitatively assesses how requirement changes will impact various program sets. For instance, Requirement R1 has strong relationships with Program Sets S1 (value 4) and S3 (value 4), indicating that changes in R1 will significantly affect these program sets. In comparison, R1's impact on S2 is slightly less but still substantial (value 3). On the other hand, Requirement R2 has a moderate relationship with S1 (value 2), a strong relationship with S2 (value 5), and a moderate relationship with S3 (value 3). This suggests that changes in R2 will primarily affect S2 and will moderately impact S3 and S1. In forward traceability analysis, the value M12 (3) is smaller than M22 (5), meaning the degree of association between R2 and S2 is stronger than between R1 and S2, implying that changes in R2 will have a greater impact on S2 than changes in R1.

Conversely, in reverse traceability, the value M11 (4) being larger than M21 (2) indicates that S1 is more strongly associated with R1 than with R2, suggesting that changes in S1 will significantly affect R1 but less so R2. This matrix analysis supports both forward and reverse traceability of requirement changes, essential for effective management in Agile Requirements Development (ARD). The ability to visualize these relationships and quantify their strengths ensures informed decision-making, highlighting the importance of specific requirements and their impacts on the overall system.

In practical terms, the values in the traceability matrix help identify which elements are most affected by requirement changes, allowing developers to prioritize changes that minimize overall system impact. By choosing paths with lower correlations, developers can implement changes more efficiently, reducing the risk of unintended consequences. Additionally, the traceability matrix supports both forward and reverse traceability, ensuring that any changes in the requirements can be traced through to their effects on program sets and vice versa.

III. RESULTS AND DISCUSSION

This section provides a practical application of the proposed framework using a tool developed as a proof-of-

concept and a case study (e.g., a banking system) to illustrate the tool's functionalities. The Problem-framework-oriented Requirements traceability Tool (PRT Tool) is implemented using Java and SpringBoot. The persistence layer uses MyBatis, and the database uses MySQL, while Node+Vue develops the front end.

A. Case Study

The banking system consists of two main modules: queuing and call service, which ensures efficient customer service. When customers enter a bank, they typically fill out a form stating the type of business they need to conduct. The printer then prints a queue number or voucher for the customer to wait in line for service. The queuing system comprises separate queues for ordinary and VIP customers. Ordinary customers fill out a form and receive a queue number. VIP customers undergo identity verification before receiving their number. VIP customers can make appointments online. Scrolling subtitles on the screen inform customers about their wait time, and when it's their turn, the customer is called. The system provides queuing screen displays and broadcast call services to serve customers, enhancing their bank experience. Based on the above requirements of the bank lobby service system, the functional module diagram was first drawn to reflect the actual requirements accurately. As shown in Fig.4, the bank lobby service system is divided into two sub-modules: the queuing and call service.

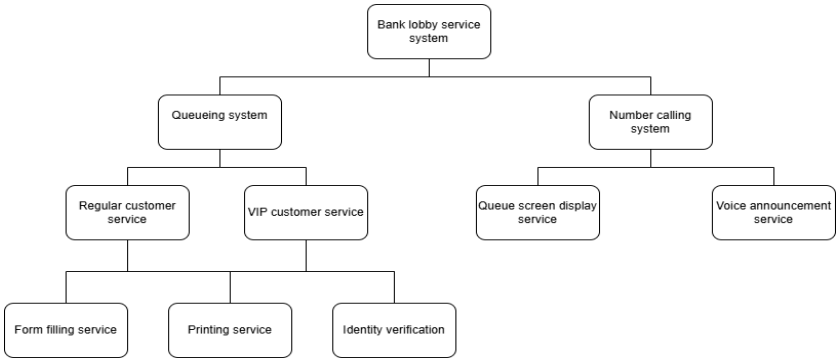

Fig. 4 Functional diagram of case study requirements

Fig. 5 is a problem diagram of the banking lobby service requirements, mainly created from two functional modules: the user queuing function and the customer service call service function.

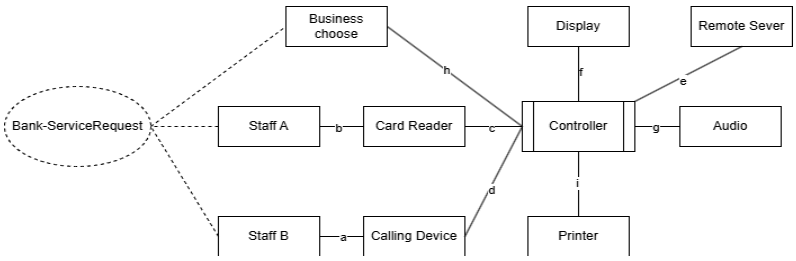

Fig. 5 Problem diagram of banking service requirements

B. Problem Diagram Splitting

1) Requirements for queuing (R1):

Ordinary users enter the banking lobby after the first use of the lobby at the entrance to the business processing machine there to choose their own relevant business and then print the voucher; the user receives the voucher and then goes to the lobby to wait in line for the number called for business. VIP customers first make an appointment online; if the reservation is successful to the bank, there will be staff to use the ID card verifier to verify their identity; if the identity verification is successful, it will be used to print out the voucher, and then wait for the number called to go to the VIP room for business transactions. A sub-problem diagram is created based on the type of uses, as shown in Fig. 6 for ordinary users and Fig. 7 for VIP users.

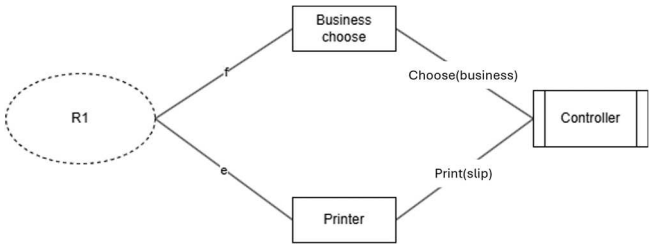

Fig. 6 User queuing

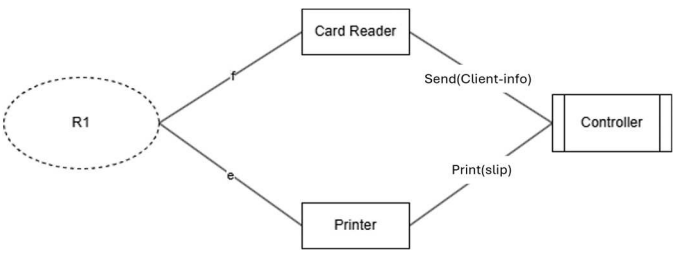

Fig. 7 VIP queuing

TABLE IV
THE SHARED RELATIONSHIPS

Shared Relationship	Description
Check(info)	Describes the information received by the calling device being sent to the remote server for inquiry.
Display(info)	Describes the central controller sending information to the display screen for display.
Choose(business)	Describes the user selecting the business to handle, and the business selector sending the business information to the central controller.
Print(slip)	Describes the printer receiving information and printing the voucher.
Use(queue)	Describes service staff pressing the calling device to call numbers.
Send (calling_info)	Sends the calling information to the central controller.
Display(info)	The central controller sends the information to the screen for display.
Play(info)	The central controller sends the information to the broadcasting system for playback.

2) Requirements for caller service (R2):

When a customer has finished the relevant business, the customer service personnel will press the caller to call the number, and then the screen in the lobby displays "Customer No. XX goes to Service Counter No. XX for service." Customers waiting for service in the lobby go to the service desk according to the voucher printed in their hands. Based on this description, create the problem diagram in Fig. 8.

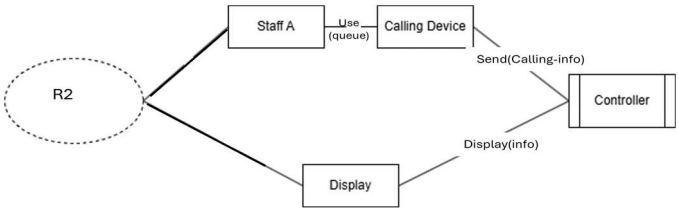

Fig. 8 Caller service

In addition, the user can hear the radio broadcasting, so, a radio broadcast sub-problem diagram is created as shown in Fig. 9.

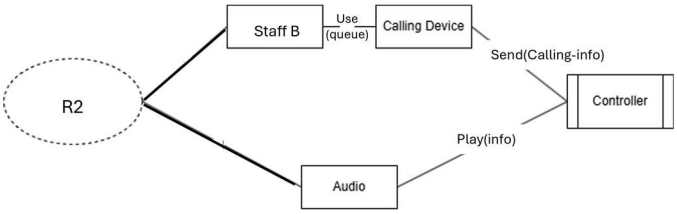

Fig. 9 Radio broadcast sub-problem diagram

The bank lobby service system problem diagram is divided into four simple sub-problem diagrams, from which the correlations from the requirements to the problem diagram and from the problem diagram to the domain can then be found. A problem diagram depicts various domains, machines, and problems, which are connected by cause-and-effect relationships. Table 4 describes the relationship.

The goal of requirements modeling is to solve the requirements, that is, to realize the problem. The domains in the problem diagram and the shared relationship in the diagram are the ways to solve the problem. So, realizing the set of programs is the goal of creating the problem diagram. Through such correlations, a requirement traceability relationship diagram is created.

C. Creating a Traceability Diagram

The user will use the PRT tool to draw a traceability relationship diagram based on the problem diagrams. This diagram outlines the relationships between requirements, problem diagrams, and domains. PRT tool ensures that all relevant elements and their interrelationships are visualized. The user can define and connect these elements in the tool, thus contributing to a detailed and accurate description of the problem domain.

The user specifies the relationships, creating a detailed diagram of how the various elements interact and depend on each other. This phase is critical to establishing the foundation data needed for practical requirements traceability and conflict management. As shown in Fig. 10, the requirements traceability relationship diagram was created from the case of Fig. 6 – 9.

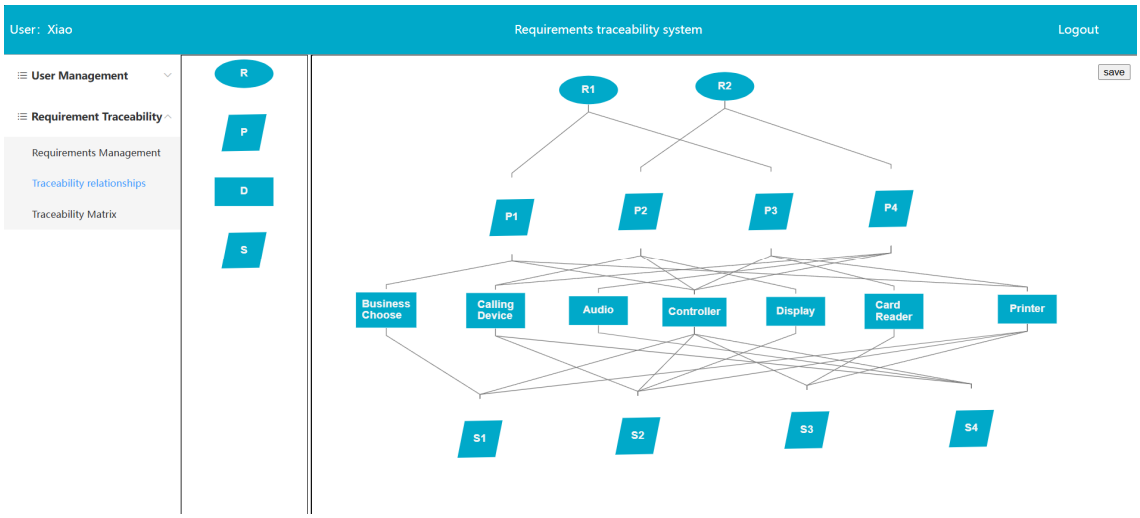

Fig. 10 Requirements Traceability Relationship Diagram

Where R_1 represents the queuing requirement, R_2 represents the calling service requirement, P1 represents the ordinary user queuing sub-problem diagram, P_2 represents the screen display sub-problem diagram, P_3 represents the VIP customer queuing sub-problem diagram, and P_4 represents the radio broadcasting sub-problem diagram. S_1 represents the implementation program set of P_1, S_2 represents the implementation program set of P_2, S_3 represents the implementation program set of P_3, and S_4 represents the implementation program set of P_4. S_1 represents the realization set of P_1, S_2 represents the realization set of P_2, S_3 represents the realization set of P_3, and S_4 represents the realization set of P_4. Table 5 summarizes the representation.

TABLE V
DESCRIPTION OF CASE STUDY DOMAINS

Item	Description
R1	the queuing requirement
R2	the calling service requirement
P1	the ordinary user queuing sub-problem
P2	the screen displays a sub-problem
P3	the VIP customer queuing sub-problem
P4	the radio broadcasting sub-problem
D1	Business choose is used to select the type of business operation
D2	Calling Device
D3	Audio, the speaker used to broadcast calling information
D4	Controller
D5	The display used to show calling information
D6	A card Reader is used to retrieve the ID
D7	The printer used to print vouchers

D. Generating Traceability Matrix

In this phase, the input data (the traceability relationship diagram) is processed to generate the requirements traceability matrix and analyze its relationships. The traceability matrix begins by classifying nodes into specific categories, including requirements (R), problems (P), domains (D), and sets of programs (S). Each node has attributes that help distinguish its type and role in the system.

1) Frontend Processing: When a user builds a requirements traceability diagram, the front end captures all interactions and dynamically updates the node data and linked data arrays. The node data array stores basic information about each node, including its type, category, and connections, while the linked data array captures the relationships between nodes, showing how they are connected in the traceability process. This data is then processed to produce a structured JSON format representing the entire diagram and the relationships it encodes. This JSON data is sent to the back end for further processing, and the actual traceability matrix is calculated. As the matrix is created, the front-end processes the nodes and links to prepare them for conversion into the matrix format. This includes categorizing nodes into specific groups, such as requirements, issues, domains, and assemblies, and determining whether they are linked. For example, the front end identifies whether a requirement node is connected to a problem graph node and assigns a binary value (1 or 0) to represent this relationship in the matrix. This initial processing ensures that the data sent to the back end has been organized into a format that can be used directly for matrix multiplication and analysis. The front end allows the user to interact with the traceability matrix once generated. After the back end has processed the data and returned a computational matrix, the front end presents that matrix in a tabular format, where each cell represents a relationship between a requirement and a program set. Users can click on matrix cells to highlight relationships, tracing connections to the original chart. This interactive visualization is critical to understanding the impact of requirement changes because it allows the user to explore the matrix dynamically.

2) Backend Processing: The backend is implemented using Java, SpringBoot, and MyBatis to process the JSON data received from the front end. Once the front-end nodes (e.g., requirements, problem graphs, domains, and assemblies) and their interconnected JSON data are transferred to the backend, the backend parses these JSON objects. It converts them into structured data arrays that generate traceability matrices. The backend first categorizes the data based on the relationships between different elements, such as the link between a requirement and a problem graph or between a domain and a program set.

The backend initiates the matrix generation process once the data has been properly categorized. This involves creating an initial relationship matrix where each element or node is represented as a row and column. The backend then populates the matrix based on the data received from the front end and whether there is a relationship between the nodes. For example, suppose a requirement node is connected to a problem graph node. In that case, the corresponding cell in the matrix is populated with a "1" to indicate the relationship, and vice versa, with a "0". This binary matrix is a fundamental step in the traceability process, as it provides a clear, quantitative representation of the relationships between different elements. In addition, the backend is responsible for ensuring the persistence of the data. Once the traceability matrix is generated, it is stored in the MySQL database.

Basic steps to create a relationship matrix:

- Iterate through all the elements to get the elements representing the requirements and the elements representing the problem diagram.
- Find out if there is a line between the element representing the requirement and the element representing the problem diagram. 1 is used to indicate a line, and 0 is used to indicate no line.
- Place the resulting 1s and 0s in a temporary array value.
- Place the 1s and 0s from the temp array in the temp array.
- Place the temp array in a two-dimensional array representing the relationship between the requirements and the problem diagram, forming a matrix.
- Empty the temp value array and store the association between the other elements (1 or 0).

Table 6 describes the implemented algorithm. Once the relevant relationship matrix is obtained, it can be multiplied to get the requirements changed to the traceability matrix. The results of the requirements change traceability matrix can be used to determine the degree of relationship between the requirements and the software artifact and the ripple effect of the requirements change. The main output of the PRT tool is a correlation matrix that relates the criteria to the

corresponding program sets. This matrix results from multiplying multiple matrices involving relationships between requirements, problem graphs, domains, and program sets.

TABLE VI
ALGORITHM FOR CREATING A RELATIONSHIP MATRIX

```
Begin://Algorithm begins
    for (str1 in namelist){//Traverse all element names
            if(str1.contains "r")//Find elements that
        represent requirements and are named starting with 'r'
            reqList.add(str1)//Place these
        elements into the requirements array
            else if(str.contains"p")//Find elements that
    represent problem frames and are named starting with 'p'.
            pList.add(str1)}//Place these elements
    into the problem frames array.
    for (i in reqList){//Traverse element names in the requirements
array (denoted as i).
            for (j in pList){//Traverse element names in the
        problem frames array (denoted as j).
            if(hasLineJudge(i,j))//Check if there is a
        connection between i and j.
            valueList add "1";//If there is a
        connection, add the value 1 to the value array.
            else
            valueList add "0"}}//If there is no
        connection, add the value 0 to the value array.
    for (str2 in valueList){//Query all values in the value array.
            tempList add(str2);//Place all values into the
        temp array.
            reqAndpList add templist;//Place the temp
        array into the two-dimensional array that associates
        requirements and problem frames.
            valueList.clear}//Clear the value array to store
        other values.
End//Algorithm ends.
```

The matrix elements indicate the strength and existence of the relationship between the requirement and the program set. Non-zero values in the matrix indicate direct or indirect connections, with larger values indicating stronger correlations. Fig. 11 is a screenshot of the requirement correlation matrix calculated from Fig. 10.

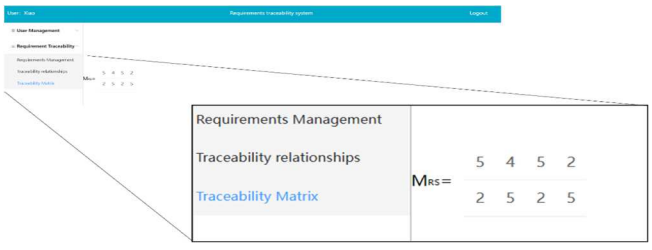

Fig. 11 Requirements correlation matrix

E. Analysis of the Results

The matrix forms the basis for traceability analysis and provides a quantitative basis for understanding the impact of requirements changes on a software system. The correlation matrix and all intermediate data are stored in a structured database to ensure data integrity and accessibility. The PRT tool employs an advanced visualization module. Users can view detailed graphs of the relationships between requirements and assemblies, making it easier to identify key dependencies and areas where changes to requirements may have an impact.

In the final output, if the matrix value M_{ij} is larger than other values, for example, in Fig. 11, the value of M_{11} is larger than that of M_{21} in the relationship matrix between requirements and program sets, it means that the degree of association between requirements R_1 and S_1 is stronger than that between requirements R_2 and S_1 and that the impact on program set S_1 is larger when requirements R_1 are changed than that on program set S_1 when requirements R_2 are changed. When requirement R_1 changes, the impact on program set S_1 is greater than that on program set S_1 when requirement R_2 changes. At the same time, the idea of backtracking can be used. In the relationship matrix, the value of M_{11} is larger than the value of M_{12}; that is, the strength of the association between requirement R_1 and program set S_1 is stronger than the strength of the association between requirement R_1 and program set S_2, so if we want to change program set S_1, the degree of the change in requirement R_1 should be stronger than the change in requirement R_1 by changing program set S_2. The problem diagram is split and modeled using Agile Requirements methodology and tools based on the problem framework. Ultimately, the matrix can be used to react to the impact on the final realization of the program set when the requirements are changed in agile requirements development or to modify a certain piece of program code, which will ultimately affect the change of requirements, which satisfies the rapid development and fast-tracking characteristics of agile requirements.

The evaluation was carried out to assess the usability of PRT using the System Usability Scale (SUS). SUS was developed and designed by Brooke [32] to measure the overall usability of a product's system. The SUS is a 10-item, 5-point Likert scale where users rate their agreement (1 strongly disagree - 5 strongly agree) with each item. The SUS system usability scale is calculated using odd and even-numbered items, and the values of the scale range from 0 to 100, which is used to assess the user's perception of the product's overall usability.

F. The Objectives

The main objective of this experiment is to evaluate the PRT tool's usability.

1) Participants:

There are six participants involved in this evaluation. The participants are all postgraduate students in the Department of Software Engineering. The participants have experience in development. Before starting the collection of results, the participants were briefly trained to conduct this experiment using the PRT tool.

2) SUS Scores

The SUS scores provided by the participants for both tools are detailed in Table 7.

TABLE VII
SUS SCORES

Participant	PRT Tool SUS Score
P1	85
P2	80
P3	90
P4	75
P5	80
P6	85
Average	82.50

For the calculation of the SUS score, the total SUS score = [(Score for odd-numbered items - 1) + (5 - Score for even-numbered items)] × 2.5, and a total SUS score of 70 or more is considered good or acceptable. In contrast, a score of 85 or more indicates that the product's usability is very high and that the overall evaluation is excellent. A score of 50 or less indicates poor or unacceptable usability. The PRT Tool received high SUS scores, indicating good usability.

IV. Conclusion

Inadequate traceability mechanisms can hinder the ability to modify requirements and comprehend their impact. Therefore, it is necessary to establish meticulous requirements traceability and maintain clear links to manage requirement changes effectively. This paper indicates the efficacy of a problem frames modeling approach in addressing issues on requirements traceability. The proposed work aims to ensure that a clear link between business requirements and system functionality exists. The proposed work is a framework that represents requirements as problems, creates a requirements relationship diagram, and generates a corresponding relationship matrix. Problem frame modeling allows complex issues to be deconstructed into manageable sub-problems, thus offering a clear structure for understanding the requirements. The traceability matrix' values aid in pinpointing the elements most affected by requirement changes, enabling developers to prioritize modifications that minimize overall system impact. A tool, namely the PRT tool, is developed as a proof-of-concept and to facilitate the process.

The framework and tool proposed in this paper have room for improvement. The splitting method is based on functional splitting and does not achieve automated splitting. Our future work aims to achieve automated splitting based on Jackson's five types of basic problem frames. The paper mainly focuses on tracing from requirement to program set. While forward tracing of attributes effectively shows the impact of program set changes on the problem domain, there is also a need for reverse tracing for validation. This paper's quantitative assessment system for demand association is based on Java Web. However, it may have limitations when addressing very complex problem graphs. In the future, we intend to design complex problem graphs for unique scenarios and involve professional testers to evaluate them.

ACKNOWLEDGMENT

We thank Universiti Putra Malaysia for all the support given.

References

[1] C. D. Laliberte, R. E. Giachetti and M. Kolsch, "Evaluation of Natural Language Processing for Requirements Traceability," *2022 17th Annual System of Systems Engineering Conference (SOSE)*, Rochester, NY, USA, 2022, pp. 21-26, doi:10.1109/sose55472.2022.9812649.

[2] S. Jeong, H. Cho and S. Lee, "Agile requirement traceability matrix," *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, Gothenburg, Sweden, pp. 187-188, 2018.

[3] S. Jayatilleke, R. J. I. Lai, " A systematic review of requirements change management", 93, 163-185, 2018.

[4] Y. Lyu, H. Cho, P. Jung and S. Lee, "A Systematic Literature Review of Issue-Based Requirement Traceability," in *IEEE Access*, vol. 11, pp. 13334-13348, 2023, doi: 10.1109/access.2023.3242294.

[5] Y. Hafeez, S. Ali, M. Jawad, F. B. Ahmad and M. N. Rafi, "Improving Requirement Prioritization and Traceability using Artificial Intelligence Technique for Global Software Development," *2019 22nd International Multitopic Conference (INMIC)*, Islamabad, Pakistan, 2019, pp. 1-8, doi: 10.1109/INMIC48123.2019.9022775.

[6] K. Kamalabalan *et al*., "Tool support for traceability of software artefacts," *2015 Moratuwa Engineering Research Conference (MERCon)*, Moratuwa, Sri Lanka, 2015, pp. 318-323, doi:10.1109/mercon.2015.7112366.

[7] M. Mezghani, J. Kang, E. -B. Kang and F. Sedes, "Clustering for Traceability Managing in System Specifications," *2019 IEEE 27th International Requirements Engineering Conference (RE)*, Jeju, Korea (South), 2019, pp. 257-264, doi: 10.1109/RE.2019.00035.

[8] D. Kchaou, N. Bouassida, M. Mefteh, and H. Ben-Abdallah, "Recovering semantic traceability between requirements and design for change impact analysis," Innovations in Systems and Software Engineering, vol. 15, no. 2, pp. 101–115, Mar. 2019, doi:10.1007/s11334-019-00330-w.

[9] W. Yinghui, W. Lifu and Z. Shikun, "A Tracing Approach of Software Requirement Change", Journal of Electronics, vol. 8, no. 34, pp. 1428-1432, 2006.

[10] Y. H. Wang, S. K. Zhang, Y. Liu, et al. "Ripple-effect analysis of software architecture evolution based on reachability matrix". Journal of Software, 2004, 15(8): 1107–1115. (in Chinese).

[11] A. Marques, F. Ramalho, and W. L. Andrade, "Towards a requirements traceability process centered on the traceability model," Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1364–1369, Apr. 2015, doi:10.1145/2695664.2695776.

[12] A. Rajbhoj, P. Nistala, V. Kulkarni, S. Soni and A. Pathan, "DizSpec: Digitalization of Requirements Specification Documents to Automate Traceability and Impact Analysis," *2022 IEEE 30th International Requirements Engineering Conference (RE)*, Melbourne, Australia, 2022, pp. 243-254, doi: 10.1109/RE54965.2022.00030.

[13] L. Lavazza, "Business goals, user needs, and requirements: A problem frame-based view," Expert Systems, vol. 30, no. 3, pp. 215–232, Sep. 2012, doi: 10.1111/j.1468-0394.2012.00648.x.

[14] K. Kannan and Saravanaguru RA. K, "An approach for decomposing requirements into analysis pattern using problem frames (DRAP-PF)," *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Kochi, India, 2015, pp. 2392-2396, doi: 10.1109/icacci.2015.7275976.

[15] L. Liu and Z. Jin, "Integrating Goals and Problem Frames in Requirements Analysis," *14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis/St. Paul, MN, USA, 2006, pp. 349-350, doi: 10.1109/RE.2006.34.

[16] A. A. Madaki and W. M. N. Wan Zainon, "A visual framework for software requirements traceability," Bulletin of Electrical Engineering and Informatics, vol. 11, no. 1, pp. 426–434, Feb. 2022, doi:10.11591/eei.v11i1.3269.

[17] M. Qadir, S. Farid, M. H. N. Bin Md Nasir, and A. Akbar, "A Rigorous Approach to Prioritizing Challenges of Web-Based Application Systems," Malaysian Journal of Computer Science, vol. 34, no. 2, pp. 130–150, Apr. 2021, doi: 10.22452/mjcs.vol34no2.1.

[18] D. Miranda, "A Web Accessibility Requirements Framework for Agile Development," *2021 IEEE 29th International Requirements Engineering Conference (RE)*, Notre Dame, IN, USA, 2021, pp. 474-479, doi: 10.1109/RE51729.2021.00071.

[19] A. Ahmad, C. Feng, M. Tao, A. Yousif and S. Ge, "Challenges of mobile applications development: Initial results," *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 2017, pp. 464-469, doi:10.1109/icsess.2017.8342956.

[20] N. Mustafa, S. Saeed, A. Abdulhakeem and M. A. M. Ibrahim, "The Impact of Scrum-XP Hybrid Methodology on Quality in Web Development with Distributed Teamwork," *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, Taiz, Yemen, 2023, pp. 1-8, doi:10.1109/eSmarTA59349.2023.10293401.

[21] S. S. A. Bukhari, M. Humayun, S. A. A. Shah and N. Z. Jhanjhi, "Improving Requirement Engineering Process for Web Application Development," *2018 12th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, Karachi, Pakistan, 2018, pp. 1-5, doi: 10.1109/MACS.2018.8628422.

[22] S. M. Saif and A. Wahid, “Web complexity factors! A novel approach for predicting size measures for web application development,” 2017 International Conference on Inventive Computing and Informatics (ICICI), pp. 897–902, Nov. 2017, doi: 10.1109/icici.2017.8365266.

[23] A. A. Alsanad, A. Chikh and A. Mirza, "A Domain Ontology for Software Requirements Change Management in Global Software Development Environment," in *IEEE Access*, vol. 7, pp. 49352-49361, 2019, doi: 10.1109/access.2019.2909839.

[24] A. Oliveros, F. Napolillo and F. L. Infesta, "Requirements in Web applications development," *IEEE CACIDI 2016 - IEEE Conference on Computer Sciences*, Buenos Aires, Argentina, 2016, pp. 1-5, doi:10.1109/cacidi.2016.7786002.

[25] 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,” Journal of Software: Evolution and Process, vol. 33, no. 10, Aug. 2021, doi: 10.1002/smr.2374.

[26] H. Tufail, M. F. Masood, B. Zeb, F. Azam and M. W. Anwar, "A systematic review of requirement traceability techniques and tools," *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, Milan, Italy, 2017, pp. 450-454, doi:10.1109/ICSRS.2017.8272863.

[27] M.-J. Escalona, N. Koch, and L. Garcia-Borgoñon, “Lean requirements traceability automation enabled by model-driven engineering,” PeerJ Computer Science, vol. 8, p. e817, Jan. 2022, doi:10.7717/peerj-cs.817.

[28] F. Wang *et al*., "An Approach to Generate the Traceability Between Restricted Natural Language Requirements and AADL Models," in *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 154-173, March 2020, doi: 10.1109/TR.2019.2936072.

[29] J. Tekutov and J. Smirnova, “The Requirements Enhancement Based on a Problem Domain Model,” ITM Web of Conferences, vol. 54, p. 01002, 2023, doi: 10.1051/itmconf/20235401002.

[30] S. Maro and J.-P. Steghofer, “Capra: A Configurable and Extendable Traceability Management Tool,” 2016 IEEE 24th International Requirements Engineering Conference (RE), pp. 407–408, Sep. 2016, doi: 10.1109/re.2016.19.

[31] M. Jackson. Problem Frames. Addison-Wesley, 2003.

[32] J. Brooke, "SUS: A “Quick and Dirty", Usability Scale, 1986.

[33] L. Xie, H. Xiao and Z. Li, "Augmenting the Problem Frames Approach with Explicit Data Descriptions Using ChatGPT," *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, Hannover, Germany, 2023, pp. 178-183, doi:10.1109/rew57809.2023.00036.

[34] S. Zhang, H. Wan, Y. Xiao and Z. Li, "IRRT: An Automated Software Requirements Traceability Tool based on Information Retrieval Model," *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, Guangzhou, China, 2022, pp. 525-532, doi: 10.1109/QRS-C57518.2022.00084.