Journal of Computers Vol. 36 No. 3, June 2025, pp. 141-155
https://doi.org/10.63367/199115992025063603010

SCRT: A Smart Contract-Based Requirements Traceability Method

Zan Wang, Zhenglong Li*, Ruiqi Luo, and Bangchao Wang

School of Computer Science and Artificial Intelligence, Wuhan Textile University, Wuhan, China
2215363074@mail.wtu.edu.cn

Received 14 November 2024; Revised 20 November 2024; Accepted 20 November 2024

Abstract. With the globalization of the software industry, requirements traceability has become increasingly critical in the software development process. However, the development of large-scale, complex software systems by cross-organizational research teams often faces challenges due to diverse organizational backgrounds, multi-site environments, conflicting objectives, and organizational boundaries. These factors can lead to trust issues, complicating the implementation of requirements traceability. To address these challenges, this study proposes a Smart Contract-Based Requirements Traceability (SCRT) framework. Smart contracts, which are executable code deployed on a blockchain, exhibit properties such as enforceability, tamper resistance, and verifiability. These characteristics empower the SCRT framework to enhance collaboration, communication, and trust among stakeholders while potentially improving the efficiency and quality of software development. Within the SCRT framework, a novel Requirements Traceability Information Model (RTIM) is introduced, which categorizes the links between new and existing artifacts. This model serves as a guide for the smart contract module, delineating which software artifacts to trace and the relationships to establish.

Keywords: requirements traceability, blockchain, smart contracts

1 Introduction

In the current field of software engineering, the rise of globally distributed development has made software product development increasingly complex, involving close collaboration among multiple development stages and partners. In this environment, effective management of requirement changes and the traceability of software artifacts throughout the entire software lifecycle becomes critically important.

Despite numerous studies on traceability [1], achieving complete traceability remains a complex and challenging task when faced with cross-organizational collaboration, distributed teams, and multi-site environments. Requirement traceability refers to the ability to trace forward and backward in the requirements lifecycle [2]. To meet quality objectives and improve traceability, software process improvement models demand the maintenance of bidirectional traceability for requirements.

Requirements traceability plays a crucial role in achieving high-quality software development, as it supports several key activities throughout the software development lifecycle, including software maintenance, project management [3], change management, and impact analysis [4]. However, in practical applications, many practitioners often view traceability as an additional task, believing that it can interfere with normal workflows and create extra burdens. This perceived overhead can be explained by the "provider-user gap," whereby the developers who create traceability links are often not the end-users of those links [5]. This phenomenon leads to inefficient implementation of traceability practices, as their true value is not fully realized in actual work. Therefore, enhancing the effectiveness and usability of traceability is particularly important for facilitating more efficient collaboration and quality assurance within the software development process.

For example, developers may create links between tasks and submitted source code, while project managers use these links to track project progress. Therefore, developers may lose motivation for this task, setting its priority lower, potentially leading to incorrect or missing links.

With the emergence of distributed teams, the complexity of traceability has further intensified, especially in contexts requiring shared artifacts [6]. In this regard, centralized data storage has been proposed, but it faces challenges of artifact trustworthiness [7] and access restrictions [8]. To address this series of challenges, there is an urgent need for a reliable and shared traceability knowledge repository to track all artifacts and traceability links created by distributed stakeholders throughout the Software Development Life Cycle (SDLC). Such a

* Corresponding Author

repository can effectively integrate information from different sources, ensuring that all participants can access and update the latest traceability data, thereby enhancing collaboration efficiency and transparency. This goal can be achieved by introducing blockchain technology based on smart contracts. The decentralized nature of blockchain not only ensures the security and immutability of data but also simplifies the creation and maintenance of traceability links through automatically executed smart contracts. The application of this technology will greatly enhance the management capabilities of artifacts and traceability links during the software development process, thereby promoting overall improvements in development quality.

The main contributions of this paper are:

(1) A novel Smart Contract Framework (SCRT) has been proposed for the reliable management and traceability of requirements in inter-organizational software projects. This model enables participants in the software development lifecycle to have a comprehensive and reliable view of software artifacts, requirement changes, and traceability links.

(2) Introducing a novel Requirements Traceability Information Model (RTIM) capable of categorizing links between any new or existing artifacts. This model guides the smart contract module on what software artifacts to trace and what relationships to establish.

(3) The introduction of blockchain technology based on smart contracts has established a reliable and shared traceability knowledge repository. This not only enhances collaboration, communication, and trust among stakeholders but also has the potential to improve the efficiency and quality of software development.

2 Related Work

2.1 Smart Contract

Smart contracts are computer programs located at specific addresses on the Ethereum network, comprising code (functionality) and data (state) [9-11]. These programs achieve decentralized applications (DAPPs) by executing transactions according to internal rules and accessing contract state data in the process.

The introduction of smart contracts not only expands the functionality of blockchain but also provides rich features for upper-layer applications [12-15]. Initially written in Turing-complete scripting languages specific to Ethereum, such as Solidity and Serpent, smart contracts exist on the blockchain in Ethereum's proprietary binary format (Ethereum Virtual Machine bytecode) after deployment. Each time called, the Ethereum Virtual Machine (EVM) loads and executes the contract's functionality. Smart contracts internally include a data type representing "account addresses," where external account addresses are typically used for cryptocurrency transactions, and contract account addresses are often used for contract invocation. The gas consumption of smart contracts is calculated based on the CPU and memory resources they occupy during execution. Once gas is depleted, the contract automatically stops execution, effectively preventing contract loops and invalid transactions [16]. Both external accounts and contract accounts can deploy smart contracts, and once successfully deployed, any Ethereum user can invoke the contract for transactions. From a data trust perspective, the call history of smart contracts is recorded in the blockchain's transaction information, providing auditability and traceability characteristics for the calling process and results.

2.2 Blockchain in Software Engineering

Recently, the academic community has called for a deeper exploration of the mutual influences between blockchain technology and software engineering (SE) [17-18]. Demi et al. [19] examined the alignment of blockchain's inherent properties with the contemporary global software engineering landscape, highlighting both the advantages and challenges associated with this technology while proposing several practical use cases. Lenarduzzi et al. [20] introduced a blockchain model that employs smart contracts (SCs) to facilitate the responsibilities of product owners in agile methodologies such as Lean Kanban and Scrum. In this framework, SCs automatically verify whether developers have accurately implemented the functionalities outlined by users by comparing the outputs of acceptance tests with the expected results. Correct implementation of user stories triggers automated payment to developers in the form of cryptocurrency or tokens. Blockchain technology offers a promising solution for supply chain traceability by providing an immutable, decentralized ledger to record transactions and track the movement of goods, enhancing transparency, accountability, and operational efficiency [21].

Fahmideh et al. [22] systematically reviewed the state-of-the-art in Blockchain-Based Software (BBS) engineering, analyzing its unique challenges across the development lifecycle and emphasizing the need for significant improvements in conventional software engineering practices. Yilmaz et al. [23] proposed a blockchain model wherein project managers introduce a new work structure into the blockchain network, allowing developers to select tasks of interest and create code that is validated by testing personnel. These testing personnel share candidate blocks and collaborate to generate subsequent blocks, thereby addressing trust and integrity issues in large-scale agile development. Bose et al. [24] explored the application of blockchain for trustworthy software provenance. They introduced a framework called Blinker, which is supported by blockchain technology. This framework captures and queries source data through the PROV series specifications, verifies data authenticity via a voting mechanism, and achieves layered, interactive visualization of source-related data.

It is important to highlight that, despite the aforementioned research, there is currently a lack of studies focusing specifically on the application of blockchain technology for requirements management and traceability throughout the entire software development lifecycle in inter-organizational software projects.

2.3 Requirements Traceability

Requirements engineering plays a pivotal role in software development projects and is widely regarded as one of the key factors for project success. Although early studies have provided empirical evidence of the positive contributions of effective requirements engineering in enhancing productivity, improving product quality, and strengthening risk management, the inherent complexity of the requirements engineering process makes it challenging to standardize through a single solution. As the complexity of software systems continues to rise, along with the increasing number of stakeholders and their heterogeneity, it becomes especially important to reinforce large-scale requirements engineering processes.

In recent years, software traceability recovery models have primarily relied on machine learning methods. Relevant studies, through experiments on open-source projects or publicly available datasets in the traceability domain, have achieved promising results. Mills et al. [25] proposed the traceability classification model TRAIL, which integrates six feature selection algorithms, four data resampling methods, and six classification algorithms to determine the optimal configuration for predicting trace links between artifacts.

Rath et al. [26] developed a model for defect-commit trace link recovery and recommendation that employs a random forest approach. This model utilizes various process information, including personnel relationships, temporal relationships, and pre-and post-trace linkages. They identified a total of 16 process features and 2 text similarity features to enhance the model’s effectiveness.

Guo et al. [27] proposed a deep learning-based traceability model using a recurrent neural network (RNN), validated on a small dataset collected from a traffic control system, aimed at recovering trace links between subsystem requirements and design definitions.

Moran et al. [28] developed a Bayesian hierarchical probabilistic traceability model called Comet by combining multi-dimensional measures of textual similarity, developer feedback, and propagation link mechanisms. They validated the model in a small project at Cisco and also developed a Jenkins-based traceability link recommendation plugin.

3 SCRT Framework

The SCRT framework consists of modules such as feasibility analysis, data collection, Requirements Traceability Information Model (RTIM), smart contracts, off-chain user control, incentive mechanisms, and blockchain.

Fig. 1 illustrates the introduction of smart contracts and blockchain-supported requirements traceability framework SCRT. This framework is built on the following assumptions: user members are distributed in an environment where mutual trust is lacking but collaboration is required in large-scale software development projects. In such a scenario, blockchain technology based on smart contracts is considered a secure repository, meeting collaboration requirements by ensuring the reliability, transparency, trust, traceability, and auditability of software artifacts and their changes.

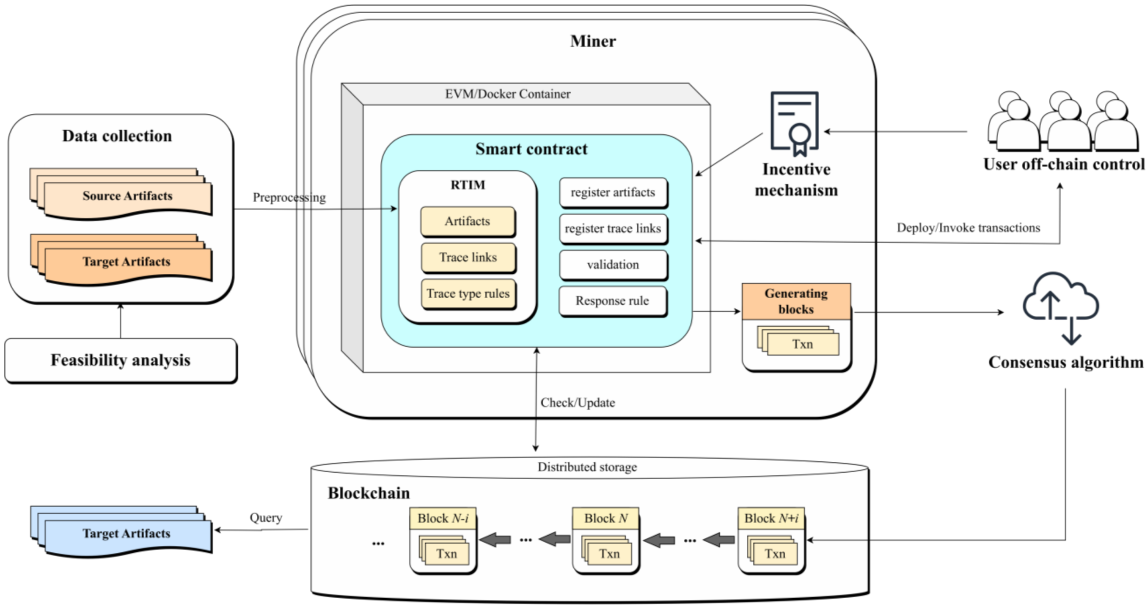

Fig. 1. Smart Contract-based Requirements Traceability Framework (SCRT)

3.1 Feasibility Analysis

Feasibility analysis aims to investigate whether blockchain is needed and can be applied to achieve requirements traceability in a particular environment. The analysis should consider a combination of blockchain features, requirements traceability strategies, alternative tools, and techniques for representing and storing traceability links. These techniques may include traceability matrices, cross-reference tables, relational databases, and graphical traceability repositories [4]. In the analysis, the possible benefits of introducing blockchain to achieve requirements traceability, such as enhanced security, decentralized transparency, tamper resistance, and verifiability, need to be evaluated. At the same time, we need to carefully consider implementation costs, technical requirements, regulatory compliance, and governance challenges. Further, we need to ensure seamless integration of blockchain technology with existing requirements traceability methods and a comprehensive assessment of all aspects, including technical teams, organizational culture, and business impact.

3.2 Data Collection

Throughout the software development lifecycle, a variety of tools, such as Rational DOORS and Git, are used to generate artifacts that conform to the traceability information model. To achieve automatic capture, data ingestion tools or plug-ins can be used [24], and parsing artifacts generated from different tools is necessary before recording data to the blockchain.

To improve the credibility of the system, the data to be recorded is collected and processed in a standardized and centralized manner. Flexible, efficient, fast, and consistent processing of data helps build traceability links for subsequent requirements. In addition, stakeholders can manually create traceable links and invoke the appropriate smart contracts to register those links on the distributed ledger. This hybrid approach to data collection ensures that critical information obtained from various tools and manual inputs can be integrated into traceability systems, enhancing data integrity and reliability.

3.3 Requirements Traceability Information Model

The Requirements Traceability Information Model (RTIM) facilitates the systematic classification of traceability links between new or existing pairs of artifacts, offering clear guidance on which software artifacts can be traced and the relationships governed by the smart contract module. This approach helps mitigate inconsistencies in large-scale projects involving numerous stakeholders [29]. As a best practice, it is recommended to predefine the traceable artifacts and their relationships when setting up a traceability environment [30]. A more in-depth exploration of RTIM will be provided in the following section. In addition to ensuring consistency, RTIM supports the structured management of complex dependencies, improving visibility and accountability across various project stages. Predefining relationships and traceability targets not only enhances clarity but also streamlines project coordination, especially when multiple teams and systems are involved.

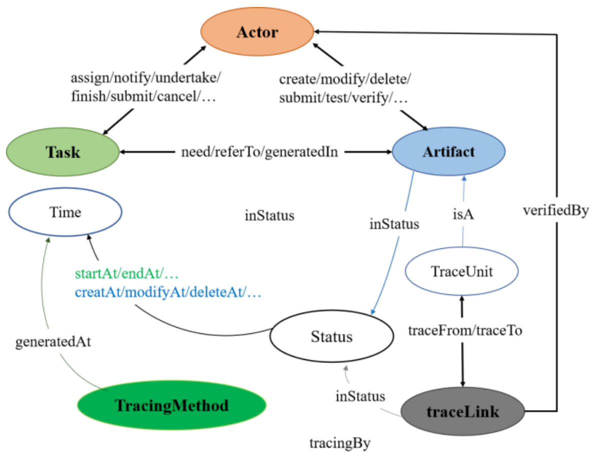

Fig. 2. Part of the high-level core pattern of requirements traceability information (Part)

Requirements Traceability Information Model (RTIM), which consists of the following three main elements:

Artifacts: Define which artifacts should be traceable and their properties.

Traceability link: Defines a link that is based on the relationship between the source artifact ID and the target artifact ID.

Traceability Type rules: Naming rules are used to define traceability links; for example, the name of the relationship can be "valid" if the source code is related to a requirement, or "invalid" if not.

These elements can be encoded into smart contracts to enforce only the artifacts and traceability links needed in the registered project and automatically identify the semantics of the traceability links. Fig. 2 illustrates several high-level core patterns relevant to requirements traceability information. These patterns encompass key elements essential to traceability activities, such as traceability artifacts, involved personnel, associated tasks, traceability links, and the methods employed for establishing traceability. Each of these components play a critical role in ensuring a comprehensive and structured approach to managing traceability, enabling better coordination and accountability throughout the development process.

3.4 Smart Contract

Smart contracts are developed to facilitate the following functions: 1) The registration of artifacts, encompassing requirements such as ID, type, name, description, priority, and parent ID, among others. 2) The establishment of traceability links, which include the source artifact ID, target artifact ID, traceability type, and additional details. 3) The verification of the quality of traceable links, along with the appropriate reward of their creators based on the assessed quality.

Fig. 3 shows the internal implementation framework of smart contracts. According to the operation mechanism, the implementation framework of smart contracts can be summarized into six stages: requirement collection and planning, contract creation, formal verification, contract triggering, contract operation, and contract cancellation. In the requirements collection and planning stage, data elements to be collected for this project need to be collected and analyzed from multiple sources to ensure the traceability of requirements.

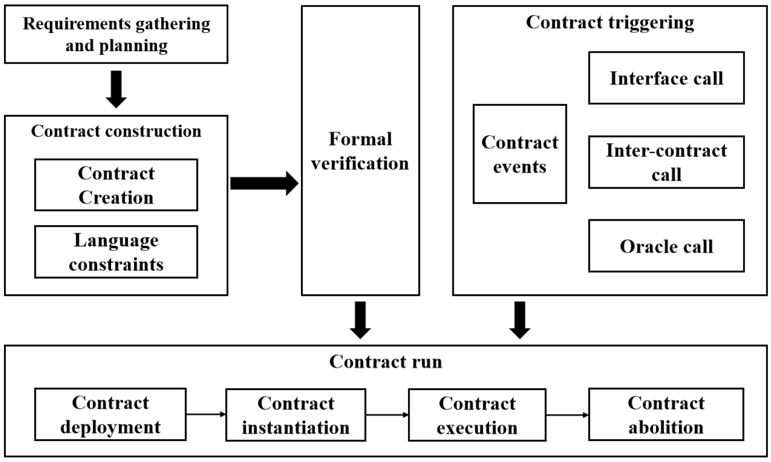

Fig. 3. Smart contract internal implementation framework

After the smart contract is jointly agreed upon and signed by multiple parties, it is submitted together with the Transaction (TXN) initiated by the user. After being disseminated by a P2P network and verified by miners, the smart contract is stored in a specific block of the blockchain. After obtaining the returned contract address and contract interface information, the user can invoke the contract by initiating a transaction. After receiving a contract creation or calling transaction, a miner creates a contract or executes contract code in a local sandbox execution environment, such as an Ethereum virtual machine. Based on trusted external data sources and checking information about the state of the world, the contract code automatically determines whether the current scenario meets the contract trigger conditions to strictly enforce the response rules and update the state of the world. Once the transaction is validated, it is packaged into a new block of data, which is authenticated by the consensus algorithm and linked to the blockchain backbone, making all updates effective.

3.5 User Off-chain Control

The quality of traceable links is verified by an approver or verifier whose decision process is influenced by a specific voting strategy. The approver provides their agreement or rejection and records this decision on the distributed ledger. The smart contract is responsible for calculating and summarizing the votes of all approvers and deciding whether to accept or reject the traceability link based on the result of the accumulated voting score.

It should be noted that assessing traceability link quality is a challenging task as this involves a process of manual inspection. Factors such as the subjective opinions of the approvers, differences in criteria between different teams, and the complexity of changes to requirements may affect the accuracy of the assessment. Therefore, when implementing smart contracts, it is necessary to consider establishing clear criteria and voting strategies to ensure consistency and reliability of the evaluation process.

3.6 Incentive Mechanism

Incentives are an important part of the SCRT framework and are designed to encourage stakeholders to actively participate in the creation and maintenance of traceability to ensure that high-quality traceability links are generated. The module supports the presentation of artifacts and traceability links in a holistic and trusted view, presented in a layered and graphical manner, facilitating the widespread application of traceability in software development life cycle (SDLC) activities.

Incentive design covers key aspects such as identifying who is qualified to create traceable links, defining criteria to verify the quality of traceable links (such as correctness, timeliness, accuracy, completeness, consistency, and usefulness), and setting the level of incentive to create traceable links based on priority.

Incentives are designed to motivate stakeholders to create traceable links in a high-quality manner, ensuring that their contributions to the system have real value. The framework considers the possibility of creating incentives to encourage the creation of high-priority traceability links, ensuring that resources and attention are focused on areas of greater value to the system.

Ultimately, incentive policies are embedded in smart contracts to distribute digital tokens to stakeholders who create high-quality traceability links, further motivating them to actively participate in traceability activities.

3.7 Blockchain-based Distributed Storage

To address network synchronization issues caused by high storage costs, as well as alleviate storage limitations, the SCRT framework allows the content of artifacts to be stored in platforms such as Ethereum or immutable storage such as IPFS. The SCRT framework treats requirements as digital assets and represents them using the concept of tokens. Each token can generate its blockchain ledger to audit the lifetime of any demand token throughout the software development lifecycle.

Fig. 4 shows the data structure of blocks, and each block consists of two parts: block head and block body. The block header contains metadata about the block, which is used to authenticate the block and establish an association with its parent and child blocks. Typically, block headers contain the Parent Block Hash, Timestamp, Nonce, Difficulty, Merkle Root, and other information. The block body is a sequence of transactions that have occurred since the previous block.

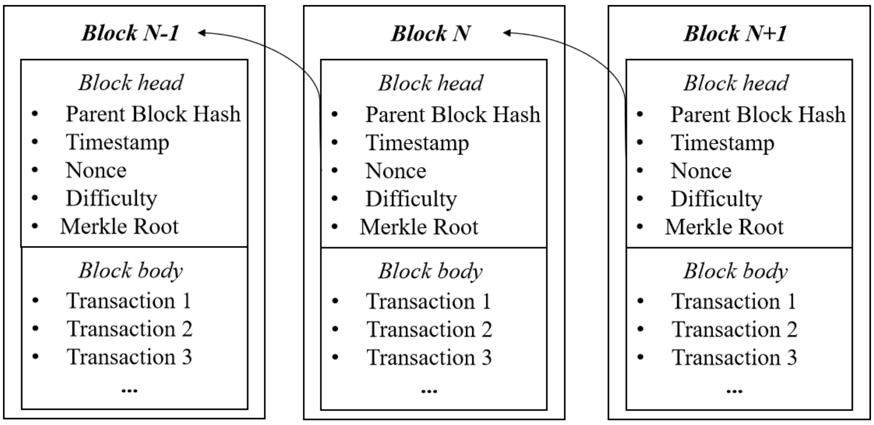

Fig. 4. Block data structure

4 RTIM Study Design

In complex enterprise projects, the challenges related to the quality of traceability between software artifacts become significantly pronounced, underscoring the pressing need for automated traceability recovery practices. These projects often experience dynamic shifts in software methodologies, programming languages, and team structures, which in turn lead to rapid alterations in software artifacts. The potential trace links among various artifacts—such as requirements documents, defect reports, test cases, and source code—are extensive, making it exceedingly difficult to maintain high-quality trace links through manual efforts.

Moreover, the exceedingly low proportion of linked artifacts in historical data results in a scarcity of labeled samples, complicating the development of effective trace link prediction models aimed at enhancing software traceability quality. Another significant challenge is that software traceability requires the alignment of different artifacts to ascertain the existence of links between them. When the ratio of “true” links to “false” links is heavily skewed, it further aggravates the imbalance within the sample data. For example, a requirement or defect (the source artifact) may correspond to one or more code commits (the target artifacts). Consequently, all potential code commits generated during the lifecycle of the source artifact are selected as candidate target artifacts. In industrial settings, the frequent code commits necessitated by tight delivery schedules lead to a considerable number of unrelated commits being included as candidate target artifacts, thereby significantly contributing to the data imbalance issue. Previous research has seldom proposed targeted solutions to address this problem within the context of software traceability.

In summary, the globalization of software development paradigms enhances the complexity of implementing requirements traceability, as communication, coordination, and trust among stakeholders become increasingly challenging in distributed environments. This paper highlights the critical necessity of ensuring the availability and reliability of traceability in inter-organizational software projects. To tackle these challenges, our research introduces a smart contract-based requirements traceability model (RTIM), specifically designed to fulfill these needs. Importantly, this paper does not focus on particular algorithms, features, or configurations; instead, it conceptualizes RTIM as a flexible framework in which individual components can be substituted with others performing analogous functions.

4.1 Features Representing Links

This section further introduces the Requirements Traceability Information Model (RTIM), which considers all possible links between two given sets of software artifacts and predicts the validity of each link (i.e., the relevance between the artifacts). Specifically, for two artifact sets, D1 and D2, where D1 is the source artifact set and D2 is the target artifact set, the method predicts the validity of each element in the Cartesian product D1 × D2. In other words, for each artifact d1 ∈ D1 from the source artifact set, RTIM predicts the validity of potential links between d1 and any artifact d2 ∈ D2 from the target artifact set.

In the implementation of RTIM, a set of internal vector representations for the classifier is required to perform this classification. We utilize IR (information retrieval)-based features, which can capture the semantic similarity between software artifacts. Given the possible links between artifacts d1 ∈ D1 from the source artifact set and d2 ∈ D2 from the target artifact set, RTIM uses two IR-based metrics to capture the features of the link.

This method employs IR techniques twice to obtain two sets of features. First, using d1 as the query and the artifacts in the target set D2 as the corpus, an IR engine is run, and the rank of d2 in the result list is captured as the first type of feature. Then, using d2 as the query and the artifacts in the source set D1 as the corpus, the same process is repeated, and the rank of d1 in the result list is captured as the second type of feature. Thus, RTIM represents each potential link by considering trace recovery in either direction (i.e., using either artifact in the link as the query). This is a key distinction from traditional IR methods, which typically consider only a single direction. Previous research has shown that the choice of retrieval direction can impact retrieval performance, particularly in the context of traceability.

4.2 IR-based Feature Extraction

In this study, we use 4 different IR methods to calculate the above two clock metrics, and in this next section, we briefly describe each method.

Vector Space Model (VSM). The Vector Space Model (VSM) is a widely used model in the fields of text processing and information retrieval. Feature extraction based on the VSM model transforms the problem of artifact association into a vector operation. Specifically, VSM treats the source artifact set D1 as a query q, and by calculating the similarity score with each document in the target artifact set D2, it quantifies the degree of association between the source and target artifacts. In VSM, each text is represented as a vector composed of terms $T=\{t_1, t_2, ..., t_n\}$, where each term t_i is assigned a weight w_i.

In this model, the weights $W=\{w_1, w_2, ..., w_n\}$ of the terms are considered the values of the corresponding dimensions in an n-dimensional space, and the term set T is regarded as a vector in this n-dimensional space. The weights w_i of the terms are typically computed based on the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm, which assigns higher weights to terms that appear frequently in a few documents but are relatively rare in the overall corpus. This better reflects the importance of the terms within the documents. This approach allows VSM to effectively capture semantic similarities between documents, making it widely used in information retrieval and text analysis. The similarity between documents is calculated by the following formula:

$$sim_{cosine}\left(d_1,d_2\right)=\frac{\sum_{i=1}^{t}w_{d_{1,i}}w_{d_{2,i}}}{\sqrt{\sum_{i=1}^{t}w_{d_{1,i}}^2\times\sum_{i=1}^{t}w_{d_{2,i}}^2}}. \tag{1}$$

$$w_i=tf_d\left(t_i\right)\times idf_i. \tag{2}$$

$$idf_i=\log\left(\frac{c}{df_i}\right). \tag{3}$$

In the above equation, $tf_d(t_i)$ represents term frequency, idf_i represents inverse document frequency, c denotes the total number of documents in the corpus, and df_i represents the number of documents containing the specific term t_i. The weight of the i-th term in vectors d_1 and d_2 is represented by $w_{d_{1,i}}$ and $w_{d_{2,i}}$, respectively.

Latent Semantic Indexing (LSI). LSI is an enhanced information retrieval (IR) model based on the Vector Space Model (VSM), which delves deeper into the latent semantic structure within a corpus. By effectively capturing the "semantic relevance" between queries and documents, LSI excels at addressing issues related to synonyms and polysemy, thereby significantly improving the computational accuracy of the traditional VSM model.

LSI employs Singular Value Decomposition (SVD) to process the term-document matrix. Through SVD, LSI maps the original term-document matrix into a lower-dimensional latent semantic space, uncovering deeper semantic relationships between documents. In practical applications, LSI first computes document vectors, which are typically generated using the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm. This algorithm calculates the weight of each term within a document to construct the document vector. However, when dealing with large document collections, processing such a massive matrix becomes challenging. To tackle this, LSI leverages SVD to reduce the dimensionality of the term-document matrix X. The SVD algorithm is as follows:

$$X=Q_uS_uD_u^T. \tag{4}$$

Here, Q_u is an m × r matrix, where each column is called a left singular vector. D_u is an r × n matrix, where each column is called a right singular vector. S_u is an r × r diagonal matrix, with each value representing a singular value of matrix X. After performing SVD on the term-document matrix X, only the largest k singular values are retained in S_u, along with the corresponding k singular vectors in Q_u and D_u. These k singular values form a new diagonal matrix S_k, and the k left and right singular vectors form the new matrices Q_k and D_k^T, respectively. Thus, the reduced matrix X_k can be expressed as the product of Q_k, S_k, and D_k^T:

$$X_k=Q_kS_kD_k^T. \tag{5}$$

Where X_k is the low-dimensional approximation of the original term-document matrix X. The similarity between requirement documents and target documents can be calculated as follows:

$$sim_{LSI}\left(q,D_i\right)=\sum_{k,j=1}^{m}q_jd_{i,k}\left(\overline{t_j}\cdot\overline{t_k}\right). \tag{6}$$

In the formula, $d_{i,k}$ represents the weight of term t_k in the requirement term document D_i, and q_j represents the weight of term t_j in the target term document q.

Latent Dirichlet Allocation (LDA). LDA is a generative probabilistic model that represents each document as a mixture of latent topics, with each topic characterized by a distribution over words in the corpus. By leveraging the probabilistic distribution of "word-topic" relationships, LDA transforms word similarity into corresponding topic probabilities, making it particularly effective at handling polysemy (the presence of multiple meanings for a single word). Additionally, in the classification process, LDA tends to classify noisy data in a document under secondary or less relevant topics, which helps mitigate the impact of noise in the document.

When using LDA, each document is converted into a probability vector, where each element describes the likelihood of a particular topic appearing in the document. This probabilistic representation allows for a structured comparison between documents. In our approach, we utilize this LDA-based representation in conjunction with the Hellinger distance to compute the similarity between documents. Specifically, the similarity between two LDA document representations, d_1 and d_2, is defined as follows:

$$sim_{LDA}\left(d_1, d_2\right) = 1 - \frac{1}{\sqrt{2}}\left\|\left(\sqrt{d_1} - \sqrt{d_2}\right)\right\|. \tag{7}$$

Jensen-Shannon (JS). The Jensen-Shannon similarity model is primarily used for recovering traceability links. It is driven by probabilistic methods and hypothesis-testing techniques. Like other probabilistic models, the JS model assumes that each document has an underlying probability distribution, where each document is represented by a normalized term-document matrix t×d. The probability distribution of the source and target artifacts is defined as:

$$p = \frac{n\left(w, d\right)}{T_d}. \tag{8}$$

Here, $n(w, d)$ refers to the frequency of a term w in document d, which could be a source or target document, and T_d represents the total number of terms in document d. Then, based on each term's inverse document frequency, the empirical distribution is adjusted to account for the global ranking of words. After applying these global weights, the distribution of each document is normalized. Once all the source and target documents are represented by probability distributions, the JS divergence is used to compute the distance between these distributions, producing a ranked list of potential traceability links.

JS ranks target documents based on their "distance" from the source documents' probability distribution. For two given documents, d_1 and d_2, their similarity, as represented by the JS model, is calculated as follows:

$$sim_{JS}(d_1, d_2) = 1 - [H(\frac{\hat{p}_{d_1} + \hat{p}_{d_2}}{2}) - \frac{H(\hat{p}_{d_1}) + H(\hat{p}_{d_1})}{2}]. \tag{9}$$

$$H\left(p\right) = \sum_{w \in W} - P\left(w\right) \cdot \log P\left(w\right). \tag{10}$$

In the above formula, $H(p)$ represents the entropy of distribution p, while $\hat{p}_{d_1}$ and $\hat{p}_{d_2}$ are the empirical distributions of the query and the document, respectively. w refers to a word in document d, and W is the set of unique words in document d. $P(w)$ represents the probability of the word w appearing in document d.

4.3 Classification Algorithm

The Requirements Traceability Information Model (RTIM) reframes the challenge of traceability maintenance as a binary classification problem, where the task is to classify links between artifacts as either valid or invalid. By leveraging machine learning algorithms, RTIM utilizes the features of existing traceability links to infer statistical patterns that can distinguish between effective and ineffective links between artifacts. Through this approach, RTIM aims to predict the validity of potential links, enabling more efficient traceability management in complex software systems. We have not fixed the algorithms, features, or configurations; instead, RTIM is designed as a framework that allows for the replacement of individual components to perform similar tasks, thus adapting and configuring to specific requirements of other domains or applications. When selecting a machine learning classification algorithm, RTIM considers various algorithms previously used in software engineering applications [31-32], empirically determining the most suitable one for predicting potential link validity in the best possible way.

Table 1. Design space used for the implementations of RTIM

Categories	Notes
KNN	K-nearest neighbors classifier using K=5
NB	Naïve Bayes classifier using estimator classes
RF	An ensemble learning method
LR	Multinomial LR model with a ridge estimator
SVM	A supervised learning algorithm

To determine the optimal configuration of RTIM, given the design space defined in Table 1, we achieve the best performance of RTIM through classification algorithms.

K-Nearest Neighbors (KNN). The KNN is a simple and intuitive supervised learning algorithm. Its basic idea is that for a sample to be classified, it predicts or classifies the sample based on the labels of the k nearest neighbors in the feature space from the training set.

Naïve Bayes (NB). The NB algorithm is a simple and efficient classification algorithm based on Bayes' theorem. It assumes that the features are conditionally independent, meaning each feature contributes independently to the classification outcome. The NB algorithm calculates the posterior probability for each class (i.e., the probability of a class given the features) and selects the class with the highest probability as the prediction result.

Random Forest (RF). The RF algorithm is an ensemble learning method that constructs multiple decision trees and combines their results for classification or regression tasks. The core idea is to generate several training sets by randomly sampling with replacements from the dataset (known as "bagging"), and then build multiple relatively independent decision trees on these training sets. Finally, RF makes its final prediction by majority voting for classification tasks or by averaging the predicted values for regression tasks.

Logistic Regression (LR). The LR algorithm is a supervised learning algorithm commonly used for binary classification tasks. It models the probability that input data belongs to a certain class by fitting a logistic function (i.e., an S-shaped function). The output is a probability score between 0 and 1, and data can be classified into one of two categories by setting a threshold. Logistic regression assumes a linear relationship between the input features and the log-odds of the class probability.

Support Vector Machine(SVM). The SVM is a supervised learning algorithm. Its fundamental idea is to find an optimal hyperplane in the feature space that separates samples of different classes. The goal of SVM is to maximize the margin between this hyperplane and the nearest sample points (known as support vectors), thereby enhancing the model's generalization capability.

5 Experiment and Analysis

5.1 Experimental Data

To identify the optimal configuration for the Requirements Traceability Information Model (RTIM), this study employed a variety of datasets sourced from the Center of Excellence for Software and Systems Traceability (CoEST). The datasets chosen for experimental evaluation comprise Easyclinic, eTour, MODIS, and SMOS. A comprehensive description of these datasets, including the count of valid and invalid traceability links and the types of artifacts associated with each dataset, is presented in Table 2. Through the analysis of these datasets, we are able to thoroughly evaluate the performance and feasibility of RTIM.

Table 2. Data sets used in the evaluation

System	Invalid links	Valid links	Artifact types
EasyClinic	1218	63	UC, CC
EasyClinic	674	46	UC, ID
EasyClinic	1825	69	UC, TC
EasyClinic	1277	85	ID, TC
EasyClinic	875	72	ID, CC
EasyClinic	2857	274	TC, CC
SMOS	5856	1244	UC, CC
MODIS	892	43	High-R, Low-R
eTour	6728	308	UC, CC

(Note: "HighR" indicates "High-level Requirements", "LowR" indicates "Low-level Requirements", "UC" indicates "Use Cases", "CC" indicates "Code Classes", "ID" indicates "Interaction Diagrams", "TC" indicates "Test Cases".)

The EasyClinic dataset is a publicly available dataset commonly used in software traceability research, aimed at evaluating the traceability between requirements and other software artifacts, such as design documents and source code. This dataset originates from a system project managing hospital clinics and includes artifacts related to healthcare services, such as requirement documents, design descriptions, test cases, and source code.

The SMOS dataset is a widely used dataset in software traceability research, originating from the Soil Moisture and Ocean Salinity (SMOS) satellite mission by the European Space Agency (ESA). This mission aims to monitor Earth's soil moisture and ocean salinity levels. The dataset includes a variety of software artifacts, such as system requirements, design documents, and source code, which are used to evaluate traceability techniques.

The MODIS dataset originates from NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) project, which is primarily used in Earth observation. The MODIS dataset includes various software artifacts related to the system's development, such as requirements documents, design descriptions, test cases, and source code, and is typically used for evaluating traceability between software artifacts. This dataset provides a real-world case for software traceability research, particularly suited for verifying traceability techniques in requirements tracking, design, and implementation processes. Known for its complexity and diversity.

The eTour dataset is an electronic tour guide application developed by a team of undergraduate students from the University of Salerno in Italy. This dataset contains several important components related to software development, specifically including 58 Use Cases (UC), 116 source Code Classes (CC), and 308 valid links.

5.2 Experimental Result

Table 3 shows the scores of three evaluation metrics obtained by our design space on different data sets, namely precision rate, recall rate, and F-score.

Hayes proposed a comprehensive metric that considers both precision and recall based on industrial practices, aimed at systematically and comprehensively assessing the quality of requirements traceability link recovery technologies. The specific evaluation criteria are detailed in Table 3, providing a strong reference for measuring the quality of related technologies.

Table 3. Quality standards for requirements traceability technologies

Measure	Acceptable	Good	Excellent
Precision	20~30%	30~50%	50~100%
Recall	60~70%	70~80%	80~100%

In Table 4, the highest precision rate is highlighted in bold and the highest F-score is highlighted in bold with an underline. The results indicate that Random Forest (RF) outperforms other classification algorithms in all other dimensions. This finding also suggests that if we consider RF as the optimal configuration for RTIM, it will provide better guidance for the intelligent contract module in tracing which software artifacts and establishing what relationships.

Table 4. The three evaluation metrics were obtained by implementing RTIM on all datasets

Metrics	Dataset	Classifier				
		KNN	NB	RF	LR	SVM
Precision	UC_CC	0.1759	0.1954	0.2152	0.2500	0.2222
	UC_ID	0.1579	0.1579	0.1765	0.2069	0.1538
	UC_TC	0.1867	0.2121	0.2917	0.2456	0.2642
	ID_TC	0.4921	0.4375	0.4394	0.4462	0.4531
	ID_CC	0.2258	0.3684	0.3784	0.3889	0.3784
	TC_CC	0.3380	0.3902	0.6916	0.3602	0.4375
	SMOS	0.3047	0.3419	0.4182	0.2927	0.3186
	MODIS	0.1395	0.2143	0.1892	0.1500	0.1842
	eTour	0.0957	0.1055	0.1345	0.1172	0.1494
Recall	UC_CC	1.0000	0.8947	0.8947	0.9474	0.9474
	UC_ID	1.0000	1.0000	1.0000	1.0000	1.0000
	UC_TC	1.0000	1.0000	1.0000	1.0000	1.0000
	ID_TC	1.0000	0.9032	0.9355	0.9355	0.9355
	ID_CC	0.8750	0.8750	0.8750	0.8750	0.8750
	TC_CC	0.9730	0.6486	1.0000	0.9054	0.9459
	SMOS	0.7492	0.4862	0.8135	0.6544	0.6850
	MODIS	0.6667	0.6667	0.7778	0.6667	0.7778
	eTour	0.7000	0.5500	0.7500	0.5875	0.5750
F-score	UC_CC	0.2992	0.3208	0.3469	0.3956	0.3600
	UC_ID	0.2727	0.2727	0.3000	0.3429	0.2667
	UC_TC	0.3146	0.3500	0.4516	0.3944	0.4179
	ID_TC	0.6596	0.5895	0.5979	0.6042	0.6105
	ID_CC	0.3590	0.5185	0.5283	0.5385	0.5283
	TC_CC	0.5017	0.4873	**0.8177**	0.5154	0.5983
	SMOS	0.4332	0.4015	0.5524	0.4045	0.4350
	MODIS	0.2308	0.3243	0.3043	0.2449	0.2979
	eTour	0.1684	0.1771	0.2281	0.1954	0.2371

6 Conclusion

This study presents a Smart Contract-Based Requirements Traceability Framework (SCRT) designed to facilitate the traceability of software artifacts produced by distributed stakeholders throughout the entire software development lifecycle. For each requirement recorded on the distributed ledger, the SCRT framework offers a scalable, efficient, and trustworthy mechanism to monitor various aspects, including origin, update history, timestamps of creation or modification, implementation or testing status, and associated software artifacts such as source code and test cases. The strength of SCRT lies in its ability to provide comprehensive visibility not only for requirements managers, developers, and testers but also for other stakeholders, including project managers and quality assurance teams. By leveraging blockchain technology, SCRT guarantees complete transparency across the software development lifecycle, enabling the tracking of real-time progress and project evolution.

This enhanced visibility allows practitioners to more effectively analyze the impact of changes on system costs, timelines, and quality. In a distributed collaborative environment, such analysis can become cumbersome; however, SCRT reliably tracks all changes transparently, thereby offering robust support for project management and informed decision-making. This capability is crucial for addressing the intricate challenges of traceability in software engineering, establishing a solid foundation for achieving high-quality and efficient software development. Ultimately, SCRT not only streamlines processes but also enhances collaboration and accountability among stakeholders, fostering an environment conducive to successful project outcomes.

References

[1] S. Demi, M. Sanchez-Gordon, R. Colomo-Palacios, What have we learnt from the challenges of (semi-) automated requirements traceability? A Discussion on Blockchain Applicability, The Institution of Engineering and Technology Software 15(6)(2021) 391-411. https://doi.org/10.1049/sfw2.12035

[2] O.C.Z. Gotel, C.W. Finkelstein, An analysis of the requirements traceability problem, in: Proc. IEEE International Conference on Requirements Engineering, 1994. https://doi.org/10.1109/ICRE.1994.292398

[3] S. Akman, M. Özmut, B. Aydın, S. Göktürk, Experience report: implementing requirement traceability throughout the software development life cycle, Journal of Software: Evolution and Process 28(11)(2016) 950-954. https://doi.org/10.1002/smr.1824

[4] R. Elamin, R. Osman, Implementing Traceability Repositories as Graph Databases for Software Quality Improvement, in: Proc. 2018 IEEE International Conference on Software Quality, Reliability and Security, 2018. https://doi.org/10.1109/QRS.2018.00040

[5] R. Wohlrab, E. Knauss, J.P. Steghöfer, S. Maro, A. Anjorin, P. Pelliccione, Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture, Requirements Engineering 25(1)(2020) 21-45. https://doi.org/10.1007/s00766-018-0306-1

[6] S. Maro, J.-P. Steghöfer, M. Staron, Software traceability in the automotive domain: Challenges and solutions, Journal of Systems and Software 141(2018) 85-110. https://doi.org/10.1016/j.jss.2018.03.060

[7] S.S. Yau, J.S. Patel, Application of Blockchain for Trusted Coordination in Collaborative Software Development, in: Proc. 2020 IEEE 44th Annual Computers, Software, and Applications Conference, 2020. https://doi.org/10.1109/COMPSAC48688.2020.0-136

[8] P. Rempel, P. Mäder, T. Kuschke, I. Philippow, Requirements traceability across organizational boundaries - A Survey and Taxonomy, in: J. Doerr, A.L. Opdahl (Eds.), Requirements Engineering: Foundation for Software Quality 7830, Springer, Berlin, Heidelberg, 2013 (pp. 125-140). https://doi.org/10.1007/978-3-642-37422-7_10

[9] S. B, A.S. Sh, S.K. E, S.N. K, N. S, Blockchain Industry 5.0: Next Generation Smart Contract and Decentralized Application Platform, in: Proc. 2022 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems, 2022. https://doi.org/10.1109/ICSES55317.2022.9914151

[10] M.J.H. Faruk, B. Saha, J. Basney, A Comparative Analysis Between SciTokens, Verifiable Credentials, and Smart Contracts: Novel Approaches for Authentication and Secure Access to Scientific Data, in: Proc. Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good, 2023. https://doi.org/10.1145/3569951.3597566

[11] D. Li, W.E. Wong, S. Pan, L.S. Koh, N. Chau, Automatic Generation of Solidity Test for Blockchain Smart Contract using Many Objective Search and Dimensionality Reduction, in: Proc. 2023 10th International Conference on Dependable Systems and Their Applications (DSA), 2023. https://doi.org/10.1109/DSA59317.2023.00057

[12] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, M. Thompson, Making Smart Contracts Smarter, in: Proc. 2021 IEEE International Conference on Blockchain and Cryptocurrency, 2021. https://doi.org/10.1109/ICBC51069.2021.9461148

[13] K. Divya, U.P. PS, Defending Delicate Health Information with Corda Blockchain Enabled MAC and UCON-Based Access Controls via IPFS, International Journal of Performability Engineering 20(1)(2024) 48-55. https://doi.org/10.23940/ijpe.24.01.p7.4855

[14] S. Sikarwar, N. Jeyanthi, R. Thandeeswaran, H. Mcheick, SDS-IAM: Secure Data Storage with Identity and Access Management in Blockchain, International Journal of Performability Engineering 20(1)(2024) 32-39. https://doi.org/10.23940/ijpe.24.01.p5.3239

[15] D. Li, W.E. Wong, M. Chau, S. Pan, L.S. Koh, A survey OFNFC mobile payment: Challenges and solutions using blockchain and cryptocurrencies. in: Proc. 2020 7th International conference on dependable systems and their applications (DSA), 2020. https://doi.org/10.1109/DSA51864.2020.00018

[16] Q.-F. Shao, C.-Q. Jin, Z. Zhang, W.-N. Qian, A.-Y. Zhou, Blockchain: architecture and research progress, Chinese Journal of Computers 41(5)(2018) 969-988. http://dx.doi.org/10.11897/SP.J.1016.2018.00969

[17] R. Colomo-Palacios, Cross Fertilization in Software Engineering, in: M. Yilmaz, J. Niemann, P. Clarke, R. Messnarz (Eds.), Systems, Software and Services Process Improvement 1251, Springer, Cham, 2020 (pp. 3-13). https://doi.org/10.1007/978-3-030-56441-4_1

[18] M. Marchesi, Why blockchain is important for software developers, and why software engineering is important for blockchain software (Keynote), in: Proc. 2018 International Workshop on Blockchain Oriented Software Engineering, 2018. https://doi.org/10.1109/IWBOSE.2018.8327564

[19] D. Selina, R. Colomo-Palacios, M. Sánchez-Gordón, Software engineering applications enabled by blockchain technology: a systematic mapping study, Applied Sciences 11(7)(2021) 2960. https://doi.org/10.3390/app11072960

[20] V. Lenarduzzi, M.I. Lunesu, M. Marchesi, R. Tonelli, Blockchain applications for agile methodologies, in: Proc. 19th International Conference on Agile Software Development: Companion, 2018. https://doi.org/10.1145/3234152.3234155

[21] S. Saxena, A. Nagpal, T. Prashar, M. Shravan, A.A. Al-Hilali, M.B. Alazzam, Blockchain for Supply Chain Traceability: Opportunities and Challenges, in: Proc. 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering, 2023. https://doi.org/10.1109/ICACITE57410.2023.10182416

[22] M. Fahmideh, J. Grundy, A. Ahmad, J. Shen, J. Yan, D. Mougouei, P. Wang, A. Ghose, A. Gunawardana, U. Aickelin, B. Abedin, Engineering Blockchain-based Software Systems: Foundations, Survey, and Future Directions, ACM Computing Surveys 55(6)(2023) 1-44. https://doi.org/10.1145/3530813

[23] M. Yilmaz, S. Tasel, E. Tuzun, U. Gulec, R.V. O'Connor, P.M. Clarke, Applying blockchain to improve the integrity of the software development process, in: A. Walker, R. O'Connor, R. Messnarz (Eds.), Systems, Software and Services Process Improvement 1060, Springer, Cham, 2019 (pp. 260-271). https://doi.org/10.1007/978-3-030-28005-5_20

[24] R.P.J.C. Bose, K.K. Phokela, V. Kaulgud, S. Podder, BLINKER: A Blockchain-Enabled Framework for Software Provenance, in: Proc. 2019 26th Asia-Pacific Software Engineering Conference, 2019. https://doi.org/10.1109/APSEC48747.2019.00010

[25] C. Mills, J. Escobar-Avila, S. Haiduc, Automatic Traceability Maintenance via Machine Learning Classification, in: Proc. 2018 IEEE International Conference on Software Maintenance and Evolution, 2018. https://doi.org/10.1109/ICSME.2018.00045

[26] M. Rath, J. Rendall, J.L.C. Guo, J. Cleland-Huang, P. Mäder, Traceability in the Wild: Automatically Augmenting Incomplete Trace Links, in: Proc. 2018 IEEE/ACM 40th International Conference on Software Engineering, 2018. https://doi.org/10.1145/3180155.3180207

[27] J. Guo, J. Cheng, J. Cleland-Huang, Semantically Enhanced Software Traceability Using Deep Learning Techniques, in: Proc. 2017 IEEE/ACM 39th International Conference on Software Engineering, 2017. https://doi.org/10.1109/ICSE.2017.9

[28] K. Moran, D.N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, J. Johnson, Improving the effectiveness of traceability link recovery using hierarchical Bayesian networks, in: Proc. 2020 IEEE/ACM 42nd International Conference on Software Engineering, 2020. https://doi.org/10.1145/3377811.3380418

[29] P. Mader, O. Gotel, I. Philippow, Getting back to basics: Promoting the use of a traceability information model in practice, in: Proc. 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, 2009. https://doi.org/10.1109/TEFSE.2009.5069578

[30] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, E. Romanova, Best Practices for Automated Traceability, Computer 40(6)(2007) 27-35. https://doi.org/10.1109/MC.2007.195

[31] E. Giger, M. D'Ambros, M. Pinzger, H.C. Gall, Method-level bug prediction, in: Proc. 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2012. https://doi.org/10.1145/2372251.2372285

[32] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, A. Folleco, An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data, Information Sciences 259(2014) 571-595. https://doi.org/10.1016/j.ins.2010.12.016