A Neural Blockchain for Requirements Traceability: BC4RT Prototype

Selina Demi [1][0000-0001-5988-4697], Ricardo Colomo-Palacios [1][0000-0002-1555-9726], Mary Sánchez-Gordón [1][0000-0002-5102-1122], Carlos Velasco[2], Ramon Cano[2]

[1] Østfold University College, Halden, Norway
{selina.demi,ricardo.colomo-palacios,mary.sanchez-gordon}@hiof.no
[2] ByEvolution Creative Factory, Málaga, Spain
{carlos.velasco,ramon.cano}@byevolution.com

Abstract. The ever-increasing globalization of the software industry presents challenges related to requirements engineering activities. Managing requirements' changes and tracing software artifacts is not trivial in a multi-site environment composed of a variety of stakeholders that do not trust each other. In this study, we propose a neural blockchain prototype for the traceability of requirements (BC4RT) throughout the software development lifecycle in interorganizational software projects. The prototype is implemented using a neural blockchain platform, namely NDL ArcaNet, due to its inherent properties: performance efficiency, sustainability, and scalability. Besides these features, the proposed prototype provides a holistic and reliable view of software artifacts, requirements' changes, and trace links. The increased visibility enhances collaboration, communication, and trust among stakeholders, and can potentially improve software development efficiency and quality.

Keywords: blockchain technology, requirements traceability, interorganizational software projects, neural distributed ledger

1 Introduction

Software engineering (SE) has shifted from conventional co-located development to global distributed development. Today's software products are developed as a result of complex supply chains that entail the collaboration of a variety of distributed partners throughout the software lifecycle, from conceptualization and development, to maintenance [1]. While global software development companies leverage benefits of distributed development: time, cost, and access to skillful resources, they also face a set of challenges: lack of communication and coordination, lack of uniform processes in a multi-site environment, lack of trust, lack of management and transfer, and challenges related to requirements engineering (RE) activities [2] which is the focus of this study. Managing requirements' changes, and tracing software artifacts in both a forward and backward direction is not a trivial activity in interorganizational software projects [3]. Although a plethora of traceability studies exists [4], the traceability community has outlined the open challenge of enabling full traceability in complex and large-scale software development contexts that rely on cross-organizational collaboration of multiple stakeholders [5, 6].

This study proposes a neural blockchain prototype for the trustworthy management and traceability of requirements in interorganizational software projects. This proposal lies in the concept of creating tokens for each requirement, tracking the lifecycle of such tokens, and certifying operations that are performed on tokens, without the need for resource-wasteful consensus algorithms. Therefore, neural blockchains present an opportunity to store artifacts created throughout the software development lifecycle in a scalable, efficient, and transparent manner, while retaining security. In addition, the proposed prototype enables participants of the software development lifecycle with a holistic and reliable view of software artifacts, requirements' changes, and trace links. The increased visibility on the software development process may lead to enhanced communication and coordination, and trust among stakeholders in interorganizational software projects. In turn, this can potentially improve software development efficiency and quality.

The remainder of this study is structured as follows: Section 2 provides an overview of the fundamental blockchain concepts, applications of blockchain technology in software engineering, and requirements engineering and traceability challenges. Section 3 proposes a neural blockchain prototype for the management and traceability of requirements throughout the software development lifecycle, and Section 4 presents implementation details of the prototype. Section 5 concludes the study and presents directions for future research.

2 Background

2.1 Blockchain Basics

Blockchain is a peer-to-peer (P2P) distributed ledger technology that stores digital transactions in a chain of blocks [7]. These digital transactions represent interactions between P2P network peers that entail the exchange of digital assets which can be in the form of information, good, services or rules to trigger another transaction [8]. Network peers group up the transactions into blocks and distribute them throughout the network. It is noteworthy that these peers need to achieve agreement with regards to the correct data state on the system. Ensuring the consistency of data on the ledger for all network peers requires the deployment of consensus algorithms which vary among different blockchain implementations. The main two groups of consensus algorithms are [8]: (i) Proof-of-X algorithms, and (ii) Byzantine Fault Tolerant algorithms. Furthermore, the exchange of assets relies on contractual rights and obligations of nodes that can be digitized and managed by smart contracts (SCs). SCs are computer programs that are stored on the blockchain and enable the modification of the ledger state when certain conditions are met. The modification of the ledger state is triggered by a transaction posted to the distributed ledger [9]. Initially, smart contracts were conceptualized to enable trusted agreements among different parties in a trustless environment [9], but nowadays they are considered similar to general purpose software programs and can, at least theoretically, perform any computational task that can be performed by conventional programs [10].

The first blockchain application was proposed in 2008 and was named Bitcoin [11]. Although distributed ledger technologies existed prior to Bitcoin, the novelty of blockchain lies in the combination of existing technologies, such as P2P networks, cryptography, transactions timestamping and shared computational power [8]. The combination of these technologies enables the sharing and storage of data in a decentralized manner without the need to entrust a central party for the maintenance of the ledger. Belotti et al. [8] categorized blockchains with respect to network accessibility in: (i) permissionless blockchains – anyone can participate in the network and modify the network state, e.g., Bitcoin and Ethereum. (ii) permissioned blockchains – only selected nodes can participate in the network and modify the network state. The latter can be further categorized according to the nature of participants in private blockchains and consortium blockchains. While in private blockchains participants are within the same organization, in consortium blockchains several organizations share a common goal.

2.2 Blockchain in Software Engineering

Recently, academic researchers have encouraged the cross-fertilization of blockchain technology and SE [12, 13]. Our previous systematic mapping study [14] explored the alignment between blockchain inherent properties and the modern (global) SE landscape, benefits and challenges of using this technology, and the proposed use cases. In what follows, a limited number of these use cases is introduced:

Lenarduzzi et al. [15] proposed a blockchain model that uses SCs to relieve some of the duties of the product owner in agile processes such as Lean-Kanban or Scrum. In this model, SCs automatically validate the correctness of user stories implemented by developers by comparing the acceptance tests output with the expected output. The correct implementation of user stories triggers the automatic payment to developers in cryptocurrencies or tokens.

Yilmaz et al. [16] proposed a blockchain model in which the project leader introduces new work structures to the blockchain network, developers choose their preferred tasks and develop code which is validated by testers. Testers share a candidate block and generate consequent blocks collaboratively. This model is aimed to address trust and integrity issues in large-scale agile development.

Bose et al. [17] proposed the application of blockchain for trustworthy software provenance. The authors introduced a framework enabled by blockchain technology named Blinker that captures and queries provenance data by means of PROV family of specifications, verifies the authenticity of the data through voting mechanisms and enables hierarchical and interactive visualization of provenance related data.

Yau and Patel [18] adopted blockchain technology to achieve reliable coordination in collaborative software development. Their blockchain-based approach aims to address limitations of centralized solutions, such as single point of failure, data tampering and auditability, and lack of verification for the data to be stored. Smart contracts are used to verify the compliance of acceptance criteria for software components in an automatic fashion.

None of these studies focus on the application of blockchain technology for the management and traceability of requirements throughout the software development lifecycle in interorganizational software projects.

2.3 Requirements Engineering and Traceability

Requirements engineering (RE) is a critical component of effective software development projects. While previous studies provided empirical evidence to support the contribution of effective RE to improved productivity, product quality, and risk management [19], the RE process has been considered as inherently complex and difficult to standardize via holistic solutions [20]. As software becomes more complex and the number of stakeholders, along with their heterogeneity increases, there is a need to enhance the large-scale RE process [21]. One of the most critical challenges that has been identified in RE, particularly in managing requirements' changes in global software development is the lack of communication, coordination, and control that leads to reduced levels of trust and confidence among distributed team members [22]. In addition, Akbar et al. [22] highlighted the lack of change impact analysis at distributed sites as a significant challenge. Estimating the impact of changes on the system's costs, time and quality is essential, yet difficult to achieve in distributed settings.

According to Jayatilleke and Lai [23], requirements traceability can contribute to keep track of the impact of changes. Traceability has been defined as "*the ability to follow the life of a requirement in both a forward and backward direction…*" [24] or as the ability to create, maintain, and use links between artifacts generated in different phases of the software lifecycle [5]. Traceability is particularly important in safety-critical domains, in light of proving the specification of safety requirements, the consideration of these requirements during the design and development phases, and their validation in test cases [25]. Despite its importance, establishing traceability in practice is not a trivial task [25]. Our recent systematic literature review reported on 21 challenges of implementing traceability in organizational settings [4]. In particular, the findings revealed that in practice, traceability is perceived as an overhead, and its potential benefits are invisible throughout the software development lifecycle. Previous studies [6, 25] pinpointed the provider-user gap as the main factor that shapes this perception, along with the poor visualization of trace links. As a result, practitioners become demotivated to create and maintain trace links and assign a low priority to traceability tasks. In addition, previous studies [6, 25, 26] raised concerns regarding the deterioration of trace links as a consequence of not updating these links when artifacts change. These changes should be propagated and affected stakeholders should be notified in order to update the corresponding trace links.

The global software development paradigm exacerbates these issues, as the communication, coordination, and trust among stakeholders is difficult to achieve in distributed settings [4]. One of the few studies that provides empirical evidence on requirements traceability in interorganizational software projects has been carried out by Rempel et al. [3]. Rempel et al. [3] outlined organizational boundaries as the main problem area, as it leads to restricted access to artifacts created by the other project parties due to lack of trust. Therefore, the authors outlined the need to ensure availability and reliability of traceability in interorganizational software projects. To address these requirements, our study proposes a blockchain-enabled prototype for requirements traceability (BC4RT) which is described in **Section 3**.

3 Blockchain-enabled Requirements Traceability Prototype

Managing and tracing requirements throughout the software development lifecycle in a transparent and reliable fashion is important to ensure trust among different stakeholders. **Fig. 1** depicts a simplified version of the software development lifecycle which consists of 4 logical users – requirements manager, developer, tester, and customer. Other users are omitted for simplicity.

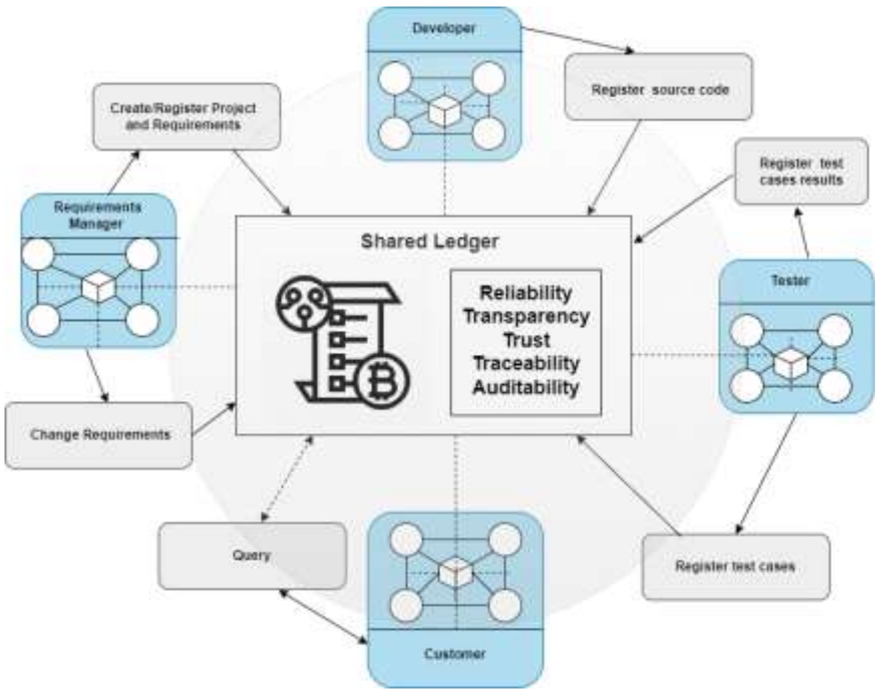


Fig. 1. High-level conceptualization of blockchain-enabled requirements traceability prototype: BC4RT

This prototype relies on the assumptions that users are located in distributed settings, and they do not trust each other, but they need to collaborate for the development of a large-scale software development project. In this context, blockchain technology can serve as a secure repository to store software artifacts and their changes by ensuring reliability, transparency, trust, traceability, and auditability. The logical users can perform different operations which are explained in the following section.

Requirements managers can create or register new projects and new requirements for each project that should be stored on the distributed ledger. The timestamp of when the requirement was created, contributor name, and the current status "created" should also be stored on the ledger. In addition, requirements managers should be able to change existing requirements and their respective attributes, such as version, description, short name. In such a case, the current status of the requirement should be "*changed*" from "*created*" and the timestamp of when the requirement is changed should be stored on the ledger. However, the immutable nature of blockchain technologies does not allow changing stored data.

At first glance, one may argue that the immutable property of blockchain goes against the ever-changing nature of software artifacts. The authors identified two potential solutions to address this challenge: (i) when requirements managers change existing requirements, a new requirement record with a new ID is created. This new requirement should point to the initial requirement that was changed by means of a previous requirement ID field; (ii) perceive requirements as digital assets and using the concept of tokens to represent them. Each token may generate its own blockchain ledger to audit the lifecycle of any requirement token throughout the software development lifecycle. In this study, the authors followed the latter approach, as it is more efficient than the former.

Furthermore, developers can register source code files for each specific requirement and consequently, the current status of the requirement is updated from "*created*" or "*changed*" to "*implemented*". Testers can register test cases for each requirement and the results of these test cases. The registration of test cases changes the status of the requirement automatically from "*implemented*" to "*tested*". Moreover, the customer has permission to view requirements' changes, and track requirements' lifecycle using the audit mode. In addition, the customer can perform more complex queries, for instance retrieve the IDs and number of requirements whose status is "*tested*", but the test result is "*failed*".

Finally, it is important to consider an efficient, scalable and secure platform to store software artifacts, such as source code files, or test cases files. If conventional blockchain platforms were chosen, these files would have been stored in secure off-chain storage, such as IPFS (Interplanetary File System) and the generated hash would have been stored in the blockchain platform to access the file's content [17]. This study adopts a novel blockchain platform that enables the secure storage of files of any size and type, while retaining efficiency and scalability. The blockchain platform adopted by this study is explained in the following section **4.1**.

4 Implementation

4.1 Neural Distributed Ledger

The concept of neural distributed ledger (NDL) was recently proposed by Velasco et al. [27] and inspired by Swan [28]'s idea of developing blockchains as "*personal thinking chains*". A neural blockchain is internally structured into subsets of groups that work in parallel and are interconnected analogously to how neuron groups are aggregated in human brains. The main utility of such blockchains lies in addressing interoperability, performance, and scalability issues that exist in conventional blockchain platforms [27]. In this study, the authors decided to implement an innovative and collaborative P2P network, namely NDL ArcaNet. NDL ArcaNet ensures the protection and secure transfer of digital assets of any type.

In order to understand NDL ArcaNet, it is important to explain the concept of NDL Arca, as a secure token directory. NDL Arca [29] is a distributed repository of tokens that ensures the protection of tokens' content against illegitimate access. Tokens are valuable, unique and certified data that must be accessed only by their legitimate owners and must be stored throughout their lifecycle in a secure repository to prevent unauthorized access and illegitimate modifications. Tokens are grouped into tables which are in turn grouped into databases. This structure lies in the combination of key-value storage and column-based databases. The keys are valuable to enable the identification of contents in any environment and are expressed in the ULID (Universally Unique Lexicographically Sortable Identifier) format. ULID generates identifiers by considering both base32 encoded timestamp (first 10 characters), and randomness (remaining 16 characters). The values are always encrypted and point to dynamic tables (variable array []). The non-static columns of these tables contain token fields' ID and token fields' content. This dynamic nature enables token fields' values to be changed according to users' needs. CRUD (create, read, update, delete) operations can be performed on tokens, along with other operations, such as import and export.

Moreover, according to [29], the security of NDL Arca is ensured by applying a set of techniques, such as 2-key encrypted token, as the data is double encrypted with database password and token password, AES 256 (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman), zero trust and zero knowledge cryptography, and hashing functions. It is worthy to mention that although NDL Arca was designed mainly for a blockchain network due to its inherent capabilities of replication, hashing of contents, and distributing them across the network, it can be installed on any system according to [29], e.g., using Arca to create a centralized dedicated server, or a database system in the cloud. The use of NDL Arca in a multi-domain network is referred to as NDL ArcaNet.

In our case study, requirements are considered tokens because they are valuable, identifiable, and unique digital assets. Requirements tokens are stored in a secure token repository and are created and updated in a collaborative manner among different stakeholders of the software development lifecycle who share a secret key. Each operation applied on tokens is visible and transparent to other parties in the network. All the operations performed on tokens will be validated by trusted certifiers who are incentivized by means of service payments that they receive for each digital signature. Trusted certifiers will validate operations on tokens without knowing the content of tokens, by applying zero-knowledge cryptography.

The authors selected this platform due to three main advantages that are important in the software engineering context: (i) *performance efficiency* – each node (wallet) applies and verifies its own transactions independently, enabling parallel work, thus maximizing the number of transactions per second. Each node can trust the token content by checking signatures, removing the need for the majority of the network nodes to vote and reach a consensus. The lack of consensus leads to each wallet working as a local database, but with slightly higher latency due to the use of signature mechanisms. Should consensus-based distributed ledger technologies be used, storing a large number of requirements or other large software files would not be affordable. However, NDL systems scale better and their limitations regarding real time operations are comparable to the limitations of centralized databases. (ii) *sustainability* – nodes collaborate to validate transactions, therefore costly, resource-wasteful, and competitive-based consensus algorithms (e.g., PoW, PoS) are not used and gas is not required to perform transactions. (iii) *scalability* – the platform can integrate million nodes because each node is independent and can work in real-time. While the Internet transfers packets of data, NDL ArcaNet transfers signed packets of data. Despite this, ArcaNet is able to scale in a similar fashion to the Internet.

4.2 Blocks Structures for BC4RT Prototype

The proposed blockchain-enabled requirements traceability prototype relies on the underlying blocks structures that are depicted in **Fig. 2**.

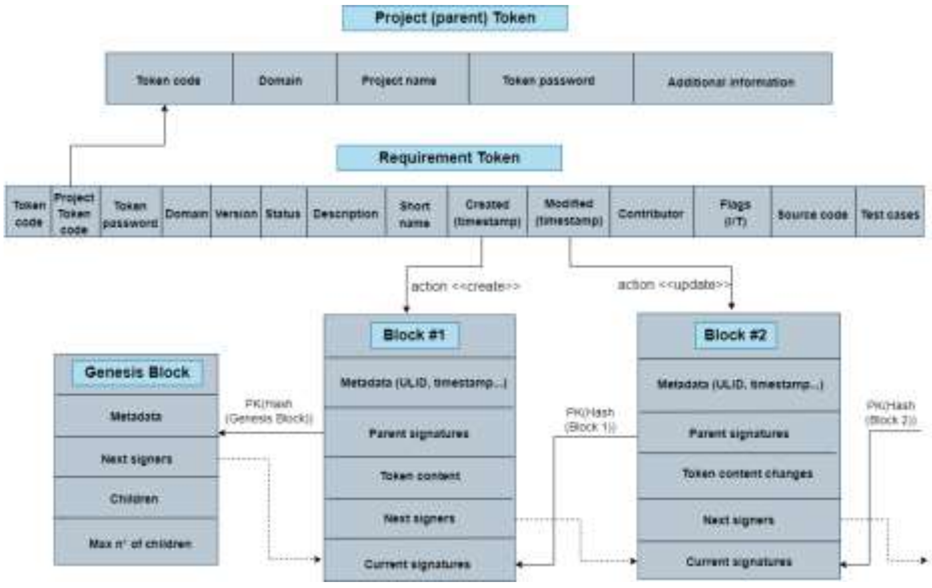


Fig. 2. Blocks structures for BC4RT prototype

Each token generates its own signed blockchain ledger that enables the verification of its provenance, integrity, evolution, and history, by means of the audit mode. The goal of the BC4RT prototype is to trace the lifecycle of requirements throughout the software development lifecycle. Therefore, a token was created for each requirement, as a child of the project token. The project token consists of the following fields: token code (ULID), domain, name of the project, and the password of the token. Other fields can be created to include additional information regarding the project. Furthermore, the requirement token consists of the following fields: token code (ULID), project code that points to the parent token, token password, domain, version of the requirement, the current status of the requirement (created, modified/changed, implemented, tested), requirement's description, short name, timestamp of when the requirement was created, timestamp of when the requirement was modified, the contributor who performed a specific operation on the token, flags (implemented/tested), source code file, and test cases file.

The emission of the requirement token generates the first block (Block #1) which is composed of the following elements: metadata, e.g., ULID, and timestamp, previous block/parent signatures which entail signing with private keys the hash of the previous block, token content which consists of the fields' content of the requirement token, next signers or the signers of the next block which are a set of trusted certifiers and random nodes, and the signatures of the current block. Signers that are defined in the previous block's next signers field should sign with their private keys the hash of the current block fields (metadata, parent signatures, token content, and next signers). While the consequent blocks have the same structure as the first block, they do not store the whole content of token fields, only the changes.

Finally, it is noteworthy that any first block needs a genesis block which is provided by the other parties of the network in a random manner. This structure allows stakeholders of the software development lifecycle to keep track of what/when/how/by whom requirements were created, changed, implemented, and tested in a trustworthy and transparent manner. A shared traceability repository based on blockchain ensures that software artifacts stored by distributed stakeholders have not been altered illegitimately.

4.3 User Interface

In what follows, we present the front end of the BC4RT application using simple scenarios that rely on the iTrust application that can be accessed online [30]. iTrust is an electronic health records application that is developed and maintained as a software engineering project for undergraduate students at North Carolina State University [31]. iTrust was chosen because it deals with safety-critical information and due to the availability of the traceability dataset [31].

The user logs in by specifying the role, i.e., requirements manager, coder, tester, or customer, as depicted in **Fig. 3**. **Fig. 4**, **Fig. 5**, and **Fig.6** show the view of the requirements manager who is allowed to create a new project, new requirement tokens, and update existing requirements, respectively. The attributes of requirements are requirement ID, contributor, requirement version, description, short name, current state, history of states (created, changed, implemented, tested), source code file, and test case file. Each of these tokens generates its own blockchain ledger that stores the changes that have been validated by trusted certifiers.

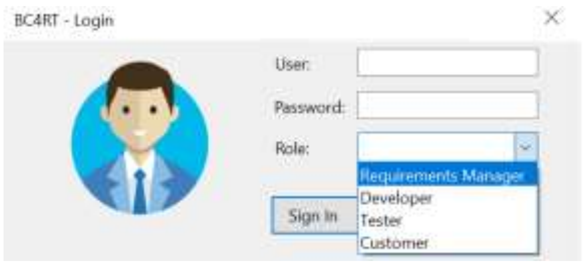
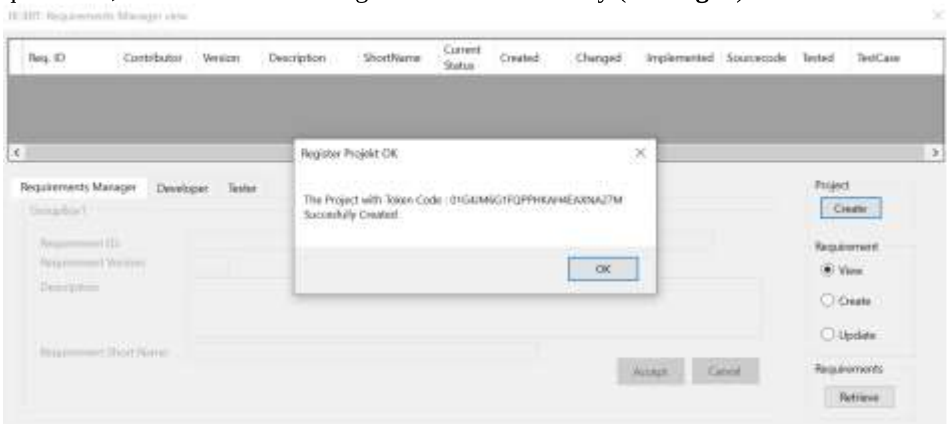


Fig. 3. Login view

First, the requirements manager creates a new project with the assigned project token ID: 01G4JM6G1FQPPHKAH4EAXNA27M (See **Fig. 4**). Then, the requirements manager creates new requirements for the project by clicking on the "*Create*" option of the radio button "*Requirement*", inputs the description and short name of the requirement, while the token ID is generated automatically (See **Fig. 5**).

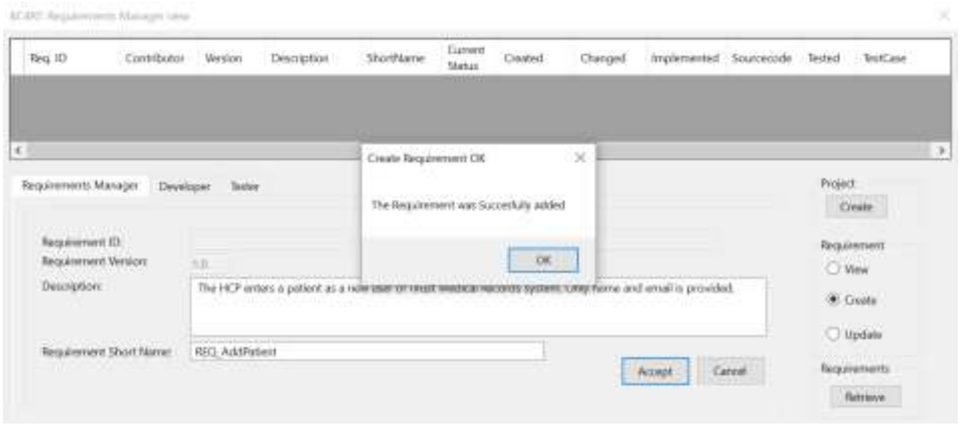


Register Projekt OK

The Project with Token Code : 01G4JM6G1FQPPHKAH4EAXNA27M Succesfully Created

Requirement: (● View) ○ Create ○ Update

Fig. 4. Requirements Manager view ("Create Project")



Create Requirement OK

The Requirement was Succesfully added

Requirement ID:

Requirement Version: 1.0

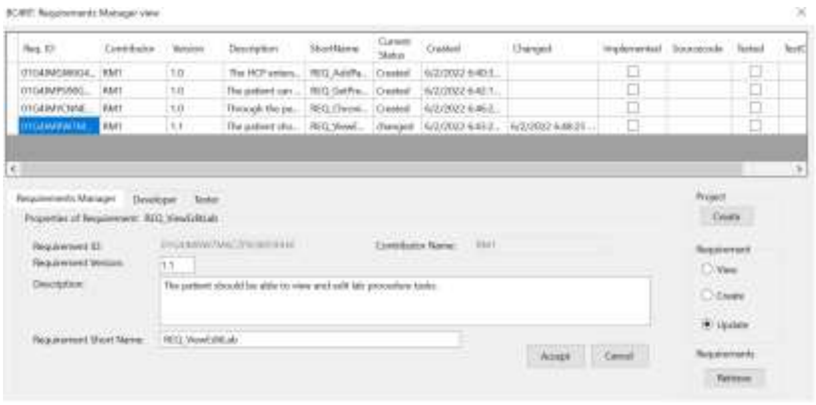
Description: The HCP enters a patient as a [illegible] medical records system. Only name and email is provided.

Requirement Short Name: REQ_AddPatient

Requirement: ○ View (● Create) ○ Update

Fig. 5. Requirements Manager view ("Create Requirement")

Once the requirements manager clicks “*Accept*”, the blockchain ledger is generated for the requirement token. The current state of the requirement is “*created*”, and the timestamp of when the requirement was created is also presented to the user (See **Fig. 6**). The requirements manager can also update previously-created requirements by clicking on the “*Update*” option of the radio button “*Requirement*”. For instance, in **Fig. 6** the requirements manager is updating the requirement with the ID= 01G4JMRW7M6CZPXJJK030H4AKK, by entering the new version=1.1, description=“The patient should be able to view and edit lab procedure tasks” and short name= “REQ_ViewEditLab”. Once the requirement manager clicks “*Accept*”, the requirement token fields are updated, the current status is “*changed*”, and the timestamp of when the requirement was changed is also stored and presented to the user, as depicted in **Fig. 6**.


BC4RT: Requirements Manager view

Req. ID	Contributor	Version	Description	ShortName	Current Status	Created	Changed	Implemented	Sourcecode	Tested	TestC
01G4JMG8MX04...	RM1	1.0	The HCP enters...	REQ_AddPa...	Created	6/2/2022 6:40:3...		☐		☐	
01G4JMP5X6G...	RM1	1.0	The patient can ...	REQ_GetPre...	Created	6/2/2022 6:42:1...		☐		☐	
01G4JMYCNNE...	RM1	1.0	Through the pa...	REQ_Chroni...	Created	6/2/2022 6:46:2...		☐		☐	
01G4JMRW7M...	RM1	1.1	The patient sho...	REQ_ViewE...	changed	6/2/2022 6:43:2...	6/2/2022 6:48:25 ...	☐		☐	

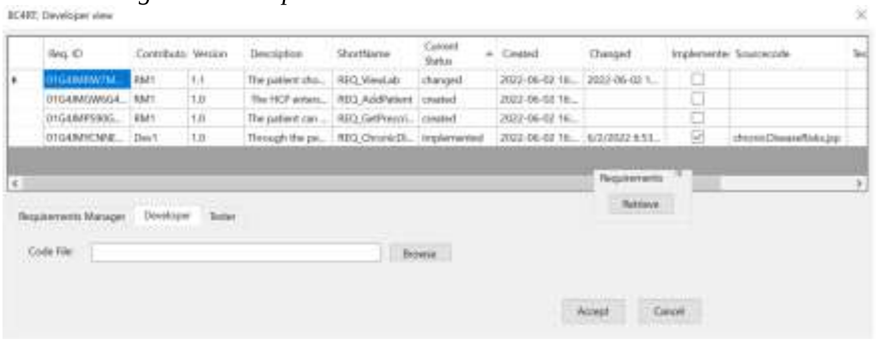
Properties of Requirement: REQ_ViewEditLab

Requirement ID: 01G4JMRW7M6CZPXJJK030H4AKK
Contributor Name: RM1
Requirement Version: 1.1
Description: The patient should be able to view and edit lab procedure tasks
Requirement Short Name: REQ_ViewEditLab

Requirement: ( ) View ( ) Create [✓ Update]

Fig. 6. Requirements Manager view (“Update Requirement”)

Second, the developer logs in the blockchain platform and is allowed to upload the source code file for each requirement (See **Fig. 7**). Once the developer enters a source code file and clicks on the “*Accept*” button, the state of the specific requirement token is updated in three dimensions: (i) *source code* field is updated with the name of the file (ii) the *implemented* field is updated (iii) the *current status* is changed from “*created*” or “*changed*” into “*implemented*”.

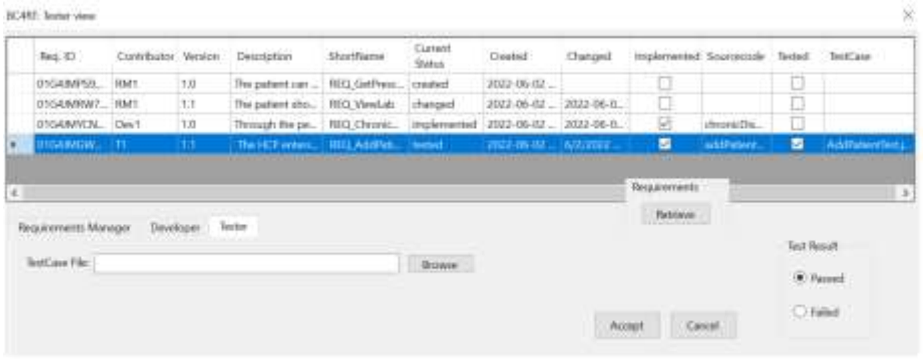

BC4RT: Developer view

Req. ID	Contributor	Version	Description	ShortName	Current Status	Created	Changed	Implemented	Sourcecode	Te
01G4JMRW7M...	RM1	1.1	The patient sho...	REQ_ViewLab	changed	2022-06-02 16:...	2022-06-02 1...	☐		
01G4JMGW6G4...	RM1	1.0	The HCP enters...	REQ_AddPatient	created	2022-06-02 16:...		☐		
01G4JMP5X6G...	RM1	1.0	The patient can ...	REQ_GetPrescri...	created	2022-06-02 16:...		☐		
01G4JMYCNNE...	Dev1	1.0	Through the pa...	REQ_ChronicDi...	implemented	2022-06-02 16:...	6/2/2022 8:53...	✓	chronicDiseaseRisks.jsp	

Code File:

Fig. 7. Developer view (“Upload Source Code”)

Third, the tester logs in the blockchain platform and is allowed to upload the test case file for each requirement (See **Fig. 8**). Once the tester enters a test case file and clicks on the "*Accept*" button, the state of the specific requirement is updated in three dimensions: (i) *test case* field is updated with the test case file name (ii) the *current status* of the requirement is changed from "*implemented*" into "*tested*" (iii) the *tested* field is updated, if the tester clicks on the "*Passed*" option of the radio button "*Test Result*"



BC4RT: Tester view

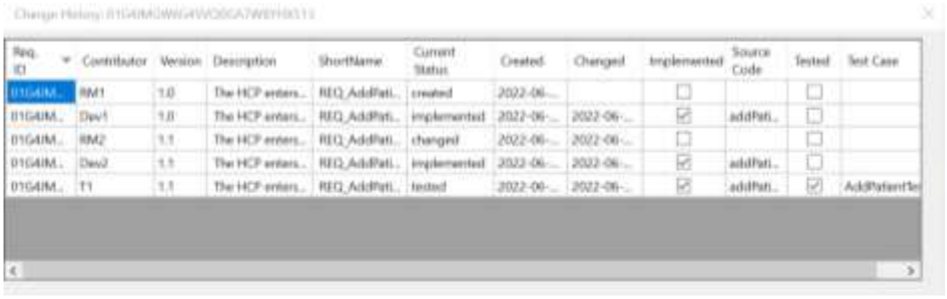
	Req. ID	Contributor	Version	Description	ShortName	Current Status	Created	Changed	Implemented	Sourcecode	Tested	TestCase
	01G4JMP59...	RM1	1.0	The patient can ...	REQ_GetPres...	created	2022-06-02 ...		☐		☐	
	01G4JMRW7...	RM1	1.1	The patient sho...	REQ_ViewLab	changed	2022-06-02 ...	2022-06-0...	☐		☐	
	01G4JMVCN...	Dev1	1.0	Through the pa...	REQ_Chronic...	implemented	2022-06-02 ...	2022-06-0...	✓	chronicDis...	☐	
▸	01G4JMGW...	T1	1.1	The HCP enters...	REQ_AddPati...	tested	2022-06-02 ...	6/2/2022 ...	✓	addPatient...	✓	AddPatientTest.j...

TestCase File:

Test Result
[✓ Passed]
Failed

Fig. 8. Tester view ("Upload Test Cases and Test Results")

Finally, the customer is constrained to only view the state of the project token and requirements tokens. Therefore, the customer can check the list of all requirements, which requirements have been created, changed, implemented, and/or tested. In addition, it is possible to trace the lifecycle of each requirement by double clicking on a specific requirement record. An example of the history of a specific requirement is depicted in **Fig. 9**.



Change History: 01G4JMGWG4VQ0GA7WEYHX51

Req. ID	Contributor	Version	Description	ShortName	Current Status	Created	Changed	Implemented	Source Code	Tested	Test Case
01G4JM...	RM1	1.0	The HCP enters...	REQ_AddPati...	created	2022-06-...		☐		☐	
01G4JM...	Dev1	1.0	The HCP enters...	REQ_AddPati...	implemented	2022-06-...	2022-06-...	✓	addPati...	☐	
01G4JM...	RM2	1.1	The HCP enters...	REQ_AddPati...	changed	2022-06-...	2022-06-...	☐		☐	
01G4JM...	Dev2	1.1	The HCP enters...	REQ_AddPati...	implemented	2022-06-...	2022-06-...	✓	addPati...	☐	
01G4JM...	T1	1.1	The HCP enters...	REQ_AddPati...	tested	2022-06-...	2022-06-...	✓	addPati...	✓	AddPatientTe

Fig. 9. Trace the lifecycle of a specific requirement

5 Conclusion and Future Work

This study presented a blockchain-oriented prototype, namely BC4RT to enable the traceability of software artifacts created by distributed stakeholders throughout the software development lifecycle. For each requirement stored on the distributed ledger, one could trace its origin, updates, the timestamp of when it was created or changed, if it was implemented and/or tested, and by whom, the current status, and related software artifacts, such as source code and test cases, in a scalable, efficient and trustworthy manner. Therefore, requirements managers, developers, testers, customers, along with other stakeholders, e.g., project managers or quality assurance team could benefit from the application of blockchain, since it ensures full visibility on the software development lifecycle and facilitates tracking projects' progress. Enabling full visibility can enhance the performance of practitioners in solving software engineering tasks. For instance, keeping track of all changes in a transparent and reliable manner facilitates the analysis of the impact of these changes on system's cost, time, and quality, which is not a trivial task in distributed settings.

The authors implemented the proposed BC4RT prototype using a novel neural distributed ledger, namely NDL ArcaNet because the inherent features of this platform ensure performance efficiency, sustainability and scalability while retaining security. The authors perceive the potential of third and fourth generation blockchain platforms and encourage further exploration of the benefits and feasibility of such platforms beyond the software engineering context. Domains that need to process massive data, such as Internet-of-Things (IoT) may greatly benefit from the efficiency and increased security of neural blockchain platforms.

Our future work will focus on validating the usefulness, practicality, and validity of the blockchain-enabled prototype through software engineering experts' judgement. Future versions of the prototype may incorporate the emission of tokens to represent other software artifacts, such as source code and test cases, as children of requirements' tokens. In addition, future work may be devoted to automate the registration of software artifacts, their attributes, content and changes, by means of data ingestion tools or plugins that can capture the artifacts generated from a variety of tools used throughout the software development lifecycle [32].

References

1. Ebert C, Kuhrmann M, Prikladnicki R (2016) Global Software Engineering: Evolution and Trends. In: 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE). pp 144–153
2. Niazi M, Mahmood S, Alshayeb M, et al (2016) Challenges of project management in global software development: A client-vendor analysis. Information and Software Technology 80:1–19. https://doi.org/10.1016/j.infsof.2016.08.002
3. Rempel P, Mäder P, Kuschke T, Philippow I (2013) Requirements Traceability across Organizational Boundaries - A Survey and Taxonomy. In: Doerr J, Opdahl AL (eds) Requirements Engineering: Foundation for Software Quality. Springer, Berlin, Heidelberg, pp 125–140
4. Demi S, Sanchez-Gordon M, Colomo-Palacios R (2021) What have we learnt from the challenges of (semi-) automated requirements traceability? A discussion on blockchain applicability. IET Software

5. Gotel O, Cleland-Huang J, Hayes JH, et al (2012) Traceability Fundamentals. In: Cleland-Huang J, Gotel O, Zisman A (eds) Software and Systems Traceability. Springer, London, pp 3–22
6. Wohlrab R, Knauss E, Steghöfer J-P, et al (2018) Collaborative traceability management: a multiple case study from the perspectives of organization, process, and culture. Requirements Eng. https://doi.org/10.1007/s00766-018-0306-1
7. Zheng Z, Xie S, Dai H-N, et al (2018) Blockchain challenges and opportunities: A survey. International Journal of Web and Grid Services 14:352–375
8. Belotti M, Božić N, Pujolle G, Secci S (2019) A Vademecum on Blockchain Technologies: When, Which, and How. IEEE Communications Surveys Tutorials 21:3796–3838. https://doi.org/10.1109/COMST.2019.2928178
9. Vacca A, Di Sorbo A, Visaggio CA, Canfora G (2021) A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software 174:110891. https://doi.org/10.1016/j.jss.2020.110891
10. Pinna A, Ibba S, Baralla G, et al (2019) A Massive Analysis of Ethereum Smart Contracts Empirical Study and Code Metrics. IEEE Access 7:78194–78213. https://doi.org/10.1109/ACCESS.2019.2921936
11. Nakamoto S (2008) Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review 21260
12. Colomo-Palacios R (2020) Cross Fertilization in Software Engineering. In: Yilmaz M, Niemann J, Clarke P, Messnarz R (eds) Systems, Software and Services Process Improvement. Springer International Publishing, Cham, pp 3–13
13. Marchesi M (2018) Why blockchain is important for software developers, and why software engineering is important for blockchain software (Keynote). In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp 1–1
14. Demi S, Colomo-Palacios R, Sánchez-Gordón M (2021) Software Engineering Applications Enabled by Blockchain Technology: A Systematic Mapping Study. Applied Sciences 11:2960. https://doi.org/10.3390/app11072960
15. Lenarduzzi V, Lunesu MI, Marchesi M, Tonelli R (2018) Blockchain applications for agile methodologies. In: Proceedings of the 19th International Conference on Agile Software Development: Companion. Association for Computing Machinery, Porto, Portugal, pp 1–3
16. Yilmaz M, Tasel S, Tuzun E, et al (2019) Applying Blockchain to Improve the Integrity of the Software Development Process. In: Walker A, O'Connor RV, Messnarz R (eds) Systems, Software and Services Process Improvement. Springer International Publishing, Cham, pp 260–271
17. Bose RPJC, Phokela KK, Kaulgud V, Podder S (2019) BLINKER: A Blockchain-Enabled Framework for Software Provenance. In: 2019 26th Asia-Pacific Software Engineering Conference (APSEC). pp 1–8
18. Yau SS, Patel JS (2020) Application of Blockchain for Trusted Coordination in Collaborative Software Development. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). pp 1036–1040
19. Damian D, Chisan J (2006) An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. IEEE Trans Softw Eng 32:433–453. https://doi.org/10.1109/TSE.2006.61

20. Franch X, Fernández DM, Oriol M, et al (2017) How do Practitioners Perceive the Relevance of Requirements Engineering Research? An Ongoing Study. In: 2017 IEEE 25th International Requirements Engineering Conference (RE). pp 382–387
21. Fucci D, Palomares C, Franch X, et al (2018) Needs and challenges for a platform to support large-scale requirements engineering: a multiple-case study. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Association for Computing Machinery, New York, NY, USA, pp 1–10
22. Akbar MA, Sang J, Khan AA, Hussain S (2019) Investigation of the requirements change management challenges in the domain of global software development. Journal of Software: Evolution and Process 31:e2207
23. Jayatilleke S, Lai R (2018) A systematic review of requirements change management. Information and Software Technology 93:163–185. https://doi.org/10.1016/j.infsof.2017.09.004
24. Gotel OCZ, Finkelstein CW (1994) An analysis of the requirements traceability problem. In: Proceedings of IEEE International Conference on Requirements Engineering. pp 94–101
25. Maro S, Steghöfer J-P, Staron M (2018) Software traceability in the automotive domain: Challenges and solutions. Journal of Systems and Software 141:85–110. https://doi.org/10.1016/j.jss.2018.03.060
26. Mäder P, Gotel O (2012) Towards automated traceability maintenance. Journal of Systems and Software 85:2205–2227. https://doi.org/10.1016/j.jss.2011.10.023
27. Velasco C, Colomo-Palacios R, Cano R (2020) Neural Distributed Ledger. IEEE Software 37:43–48. https://doi.org/10.1109/MS.2020.2993370
28. Swan M (2015) Blockchain Thinking : The Brain as a Decentralized Autonomous Corporation [Commentary]. IEEE Technology and Society Magazine 34:41–52. https://doi.org/10.1109/MTS.2015.2494358
29. Arca. In: ByEvolution Creative Factory. https://byevolution.com/en/arca/. Accessed 8 Jun 2022
30. iTrust. In: SourceForge. https://sourceforge.net/projects/itrust/. Accessed 23 May 2022
31. Meneely A, Smith B, Williams L (2012) Appendix B: iTrust electronic health care system case study. Software and Systems Traceability 425
32. Demi S, Sánchez-Gordón M, Colomo-Palacios R (2021) A Blockchain-Enabled Framework for Requirements Traceability. In: Yilmaz M, Clarke P, Messnarz R, Reiner M (eds) Systems, Software and Services Process Improvement. Springer International Publishing, Cham, pp 3–13