Applying Blockchain to Improve the Integrity of the Software Development Process

Murat Yilmaz[1], Serdar Tasel[5], Eray Tuzun[2], Ulas Gulec[3], Rory O'Connor[4,1], and Paul Clarke[4,1]

[1] Dublin City University, Ireland
murat.yilmaz@dcu.ie, rory.oconnor@dcu.ie, paul.m.clarke@dcu.ie
[2] Bilkent University, Turkey
eraytuzun@cs.bilkent.edu.tr
[3] Hasan Kalyoncu University, Turkey
ulas.gulec@hku.edu.tr
[4] Lero, the Irish Software Engineering Research Center
[5] Çankaya University, Turkey
fst@cankaya.edu.tr

Abstract. Software development is a complex endeavor that encompasses application and implementation layers with functional (refers to what is done) and non-functional (how is done) aspects. The efforts to scale agile software development practices are not wholly able to address issues such as integrity, which is a crucial non-functional aspect of the software development process. However, if we consider most software failures are Byzantine failures (i.e., where components may fail and there is imperfect information on which a component has failed.) that might impair the operation but do not completely disable the production line. In this paper, we assume software practitioners who cause defects as Byzantine participants and claim that most software failures can be mitigated by viewing software development as the Byzantine Generals Problem. Consequently, we propose a test-driven incentive mechanism based on a blockchain concept to orchestrate the software development process where production is controlled by a similar infrastructure based on the working principles of blockchain. We discuss the model that integrates blockchain with the software development process, and provide some recommendations for future work to address the issues while orchestrating software production.

Keywords: Software Production, Blockchain, Software Development Integrity, Test-driven Software Development.

1 Introduction

The communication structure of a software development organization might be likely to affect the software development process which may be based on a set of different methodologies combined for software production [1]. Accordingly, a goal of the continuous integration practice is to bring up different software development roles together and formalize cross-functional teams that are expected to be mirrored in their software design [2]. Although the quantity of software development releases increases and automated deployment becomes more essential in many software development landscapes, there remain some challenges when applying agile development methodologies to the development of large and complex software systems [3]. Recently, many software development organizations have started to use integrated DevOps approaches which heavily rely on automation tools [4]. To verify the desired functionality, an agile practice called test-driven development (TDD) is frequently used. TDD relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the these tests [5]. TDD is philosophy and can be applied to any aspect of software development (and not just requirement-to-system based testing), for example, with unit testing, it is possible to code the unit tests prior to writing the code itself, thereby another example of TDD.

Typically, software development organizations rely on a central control or an authority such as project or development manager for production management. Designing a software development incentive model for such an orchestration requires modularity and connectivity (i.e. interoperations among modules and repository) that seems incredibly difficult when there would be no central control which promotes or enforces the trust. However, as the source code grows and new features are developed, complexity increases and reaching a stable code base becomes more challenging. We argue whether a large scale agile development production could be achieved by decentralization without losing integrity and trust. For example, the scaled agile framework [6] proposes a release train metaphor to highlight the importance of a fast and flexible flow with connected compartments to deliver a part of a software product. In particular, such connected (i.e. release train-like) structures may be constructed using the blockchain technology.

Blockchain technology can address the Byzantine Generals' Problem (BGP) [7] which (in this context) queries how software practitioners, when separated, can totally agree on the integrity of software production (i.e. software delivery, incremental releases). Let's imagine an army with a group of generals (i.e. leader-follower setup) which is planning to attack. However, a consensus is required to perform an attack where the commanding general and all lieutenants must agree on the same decision (i.e. attack or retreat). The generals are apart from each other and messenger are utilized for communication. There are some lieutenants who are hidden traitors who would be likely to cripple the situation. An algorithm confirms that all loyal lieutenant generals follow the action plan, traitors can play as they wish, and a small number of traitors are not be able to change loyal lieutenants minds for adopting a bad strategy. The algorithm should provide that loyal lieutenants should reach an agreement and agree on a reasonable plan. BGP can be illustrated by the needs for consensus for distributed ledgers (i.e. nodes) which should work collaboratively to agree on a set of rules and able to agree on a particular transaction before it is added to a repository. However, it is also vital to protect the information to be sabotaged by traitor agents.

Blockchain is a transparent distributed digital ledger technology which is operational as a decentralized system. There is no possible (single) point of failure scenario that may bring down the whole structure. The chain structure consists of immutable batches called blocks (i.e. information baskets) that are tamper-proof. These blocks have the following components: (i) current timestamp, (ii) version as an index number for active transaction, (iii) information regarding transaction, (iv) a code called hash that links the block to the previous block, (v) a number (i.e. nonce) that needs to solved by using the computing power of participants, (vi) hash, which is a code that needs to be generated from all identified information to form the block. A blockchain database is distributed over a network of computers called nodes which stores a copy of all activities or transactions individually. Each block should reference through the previous block other than the Genesis (initial) block. To deal with BGP, Bitcoin has been implemented using the blockchain technology where a block is linked to another using a random number which requires a high amount of computation to generate. Based on the computational power, mining is used as a competitive technique of collecting and adding transactions as blocks to the ledger. The participants with enough computational power could be able to add a new block to the tail of blockchain by verifying the block and broadcast it to the network that ultimately solves the "proof-of-work" issue. Consequently, selecting a random number and creating the longest chain requires honest participants to create at least 50% of the total computation power solves the BGP problem. To sum up, blockchain is a novel kind of data structure which can be operationalized on participants' computers as a distributed blockchain database (i.e. a ledger) which transmits each transaction to any identical repository where a consensus-based agreement is essential for any alteration. Therefore, the ledger is immutable, all changes should be validated by participants and be captured as new blocks.

The aim of this investigation is to create an incentive mechanism to orchestrate software development production using blockchain technology. This technology enables software artifacts to be incremented by means of interconnected blocks that encompasses a piece of working software by decentralizing the production line. We aim to reduce the centralized management costs by introducing the notions of proof-of-work and proof-of-stake to improve the activities of software development. The rest of the paper is organized as follows. Section 2 reviews the previous studies to explore the possible implementations of blockchain technology both in software engineering and in different domains. Section 3 is concerned with the methodology employed for this study. The purpose of the final section is to discuss the extent to which this study has proposed and future work.

2 Literature Review

Although there is still a relatively small body of literature that is concerned with blockchain technology, more recent attention has focused on the application of blockchain in a variety of domains including but not limited to data storage and management [8], internet of things (IoT) [9], a reputation and rating system based on Bitcoin transactions [10], software licence validation [11], modern banking with ledgers [12], online learning [13], educational record and reputation [14], decentralization and auditability [15], and various reviews for blockchain usage for IoT [16–18].

Recently, researchers have shown an increased interest in applying blockchain in software development and propose a new area for software engineering research. Porru et al. [19] coin the term Blockchain-Oriented Software (BOS) to refer the software that operationalized by using an effective implementation of the blockchain technology. Authors claim that BOS should enable large scale team collaboration, enhance testing activities and ultimately more tools should be developed to foster the creation of smart and self-executing contracts (i.e. *transaction protocol that executes the terms of a contract in a blockchain network*). This study has shown that a blockchain implementation should benefit from the features that blockchain offers e.g. an improvement in data redundancy, designing the software in sequentially ordered blocks, practitioner regulation by using the public-key cryptography, etc. Marchesi [20] suggests that the application of practices to Blockchain software development is important for improving the software development process (e.g. analysis, design, quality improvement, metrics). In addition, he mentions the benefits of blockchain such as (i) data reliability in empirical research, (ii) a possible blockchain based payment system for software practitioners, (iii) performing acceptance test using smart contracts that do not require human intervention to complete, and (iv) allowing token-based payment for software usage. In addition, there have been a limited number of research involving smart contracts that have reported the importance of addressing the issues posed by a potential application that should operationalize on the blockchain networks.

Destefanis et al. [21] highlight the significance of Blockchain-Oriented Software Engineering (BOSE) that seeks to enhance blockchain applications in terms of security, reliability, modeling, and verification. Authors conduct a case study that has given consideration to the advantages of using software engineering standards, design patterns, and best practices while developing blockchain applications. However, what is not yet known is the extent of how we can bridge the gap between conventional software engineering and blockchain software development. Wohrer and Zdun [22] propose six design patterns to mitigate security problems for smart contracts, and authors aim to construct a design pattern language to improve the software development quality and trace the refactoring process. Beller and Hejderup [23] propose a blockchain-based continuous integration approach includes a build service and a package system. They highlight the importance of reaching a consensus for professionalizing the software engineering tasks, improve the quality of products and services and trust in GitHub or Travis CI. In addition, they envision a more democratized open package system for Linux systems with a participation awards. However, the literature on blockchain based software development is still limited, and a detailed model for how to realize blockchain in software development practices has not been detailed yet.

2.1 Blockchain Basics

Blockchain is an advanced data structure (sometimes called a database) based on the notion of decentralized ledger which aims to provide data integrity and mutual trust without a central authority. A blockchain implementation consists of nodes where has a copy of the whole system. The data included in the blockchain is collected by transactions which are considered as a valid messaging between two authorized participant. A typical blockchain implementation should also encompass a consensus (i.e. validation) mechanism, a messaging protocol, and the strategy to update the data stored in the blockchain. A well-known implementation of blockchain is the Bitcoin i.e., a digital cryptocurrency, which can be considered as an innovative digital money exchange model.

A conventional blockchain, a distributed consensus state machine, comprises immutable blocks to store the transaction data where blocks are digitally signed (i.e. through the medium of cryptographic hashing to assure that the decentralized ledger is well-protected) and ultimately linked to the previous block. The primary goal is to maintain trust and promote integrity while creating a block. Consequently, the process of block creation is a common task among the trusted network nodes. The nodes listen to the requested transactions which are queued for validation. A transaction encompasses the data regarding a list of sender and receiver information and the data that needs to be transferred which can be validated only if a consensus among the nodes has been reached. For Bitcoin's blockchain, block creation is evaluated with a proof-of-work schema which is a cryptographic puzzle where its difficulty needs to be adjusted based on the total computing power of the blockchain network. The puzzle solvers are also called miners who need to prove their computational power to process transactions, and their progress would be awarded by Bitcoins.

As one recent example of a next-generation decentralized ledger is the Ethereum blockchain (state machine) which is designed as a platform to build and distribute decentralized applications. Ethereum blockchain could function as a decentralized application store that would function using the notion of smart contracts. These contracts are computer programs that are able to store and compute the data and implements the terms and conditions of a contract. These bits of the source code enables individuals and computers to initiate and complete transactions which might be useful for validation, permission management, checking data availability. Consequently, these self-executing applications guide us to exchange values and goods (e.g. money or shares) without using a mediator in a conflict-free manner. There are various applications of smart contracts including but not limited to banking, digital identity, tax recording, insurance, and real estate businesses especially they are useful for forming decentralized organization structures [24].

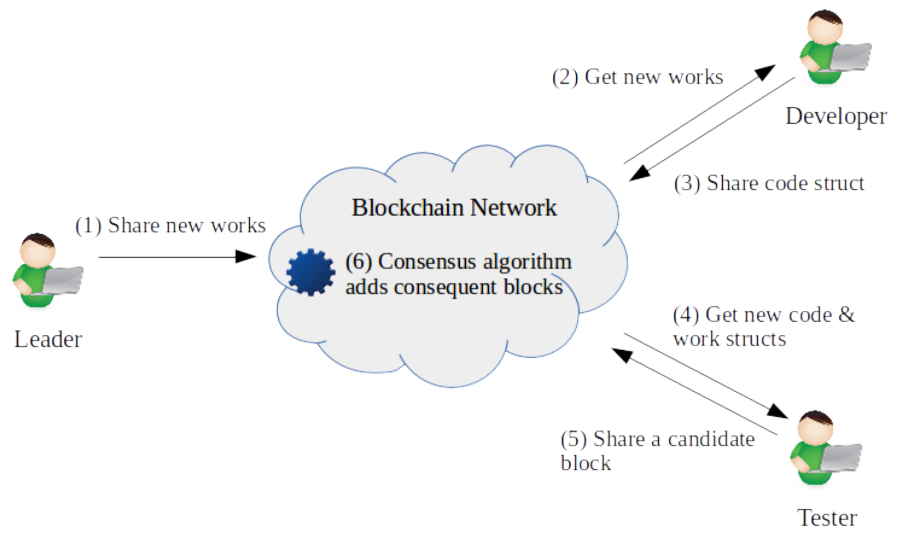
(2) Get new works
Developer
(3) Share code struct
Blockchain Network
(1) Share new works
(6) Consensus algorithm adds consequent blocks
Leader
(4) Get new code & work structs
(5) Share a candidate block
Tester

Fig. 1. Conceptual design of proposed model

2.2 Proposed Model

Here, we propose a blockchain oriented incentive mechanism to address the problems that exist in the software production. Firstly, we consider developers who create software bugs as Byzantine participants and aim to address the Byzantine general problem using what blockchain technology offers. To minimize the implication of these bugs, we use the Nakamoto Consensus which consist of four main parts; (i) proof of work (PoW), (ii) block selection, (iii) scarcity, (iv) incentive structure.

The goal is to keep the consensus effective based on the assumption that majority of effort contributed to the software development process is in the control of software practitioners who would cause less bugs. Consequently, a ledger is used to validate every piece of software using such a consortium. Secondly, we hypothesize a test-driven software production line based on a blockchain. The goal is to start with an automated test case with simple functionality that is needed to implement before start coding and the code should be developed to fulfill the test requirements and ultimately aims to make the test pass.

Analogically speaking, software developers can be considered as miners in a production system that develop code and software testers are the validators who would increment a working piece of software in the form of a block. The production line shall be formed as a release train that consists of concrete and immutable wagons that carries minimum viable products or artifacts. Most importantly, however, this model does not promote a task assignment scheme for software practitioners. Instead, practitioners could be able to select the tasks they prefer to perform. However, it is essential to note that there could be a set of duplicate works that may be performed by software developers. The goal for the testers are to find the best performance among proposed solutions.

Therefore, the structure of blockchain shows the workflow of the development, e.g. we intend to explore the creation and execution of each block and hence measure the velocity of software production. The key point here is that the code base is completely decentralized. As the conceptual design of the proposed system is depicted in Fig. 1, the project leader identifies and introduces new works to the blockchain network. Developers share their code (implementations) fulfilling desired works. Afterwards, testers acknowledge the work done by developers and shares a candidate block. Finally, all candidate blocks are used to form a subsequent block by a consensus algorithm.

In order to initialize the blockchain, we expect to form a genesis block which includes a project description and a number of work structs which would be digitally signed by an authority such as project leader. Each work struct consists of the work description, test description and is signed by the creators of work that might be the project or team leaders. Moreover, this work package additionally needs to have reputation/reward points (to be distributed to developers and testers) and expiration time based on its priority and challenge level. Growing speed of the blockchain will mostly depend on the expiration dates of works and also developer's response time. All blocks other than the genesis block are called a consequent blocks which will be generated in collaboration and digitally signed by software testers. A block generated by each software tester can be considered as a candidate block which has a potential to be a part of the actual next block in the blockchain after having approved by the consensus algorithm in which a variant of a proof of stake mechanism is employed. A consequent block shall consist of additional work structs (which could be null in some cases), code structs and a reference to the previous block. Each code struct involves a piece of code that claims to fulfill the requirements of work and provides a reference to the associated work struct (via using an ID number generated by the hash of the work struct). As a whole, this would be called a transaction which needs to be signed by a tester in order to establish the proof of work. Ultimately, candidate blocks are merged by the consensus algorithm to produce the actual consequent block which is to be added to the blockchain. Similar to the other blockchain applications, a tail with a reasonable length must be formed in order to ensure that the added block is secured. Normally, all new blocks should be subjoined to the lattermost block. However, in some special cases, a block can be specifically attached to a former block, which is called a fork. This pattern analogically corresponds to fork events occured in software repositories. Furthermore, a block can be arranged to refer to some particular locations of multiple blockchains, which means, in this case, merging.

The proposed software production line would be based on a reputation/reward sub-mechanism which would potentially address starvation problems (i.e. participants should be matched with tasks at hand) that the system might possibly encounter. In this context, software practitioners in a software development organizations should be assigned to an issue based on their incentives and skillset. In particular, every participant should be able to deliver a piece of work and get a reputation and/or reward based on their proper progress.

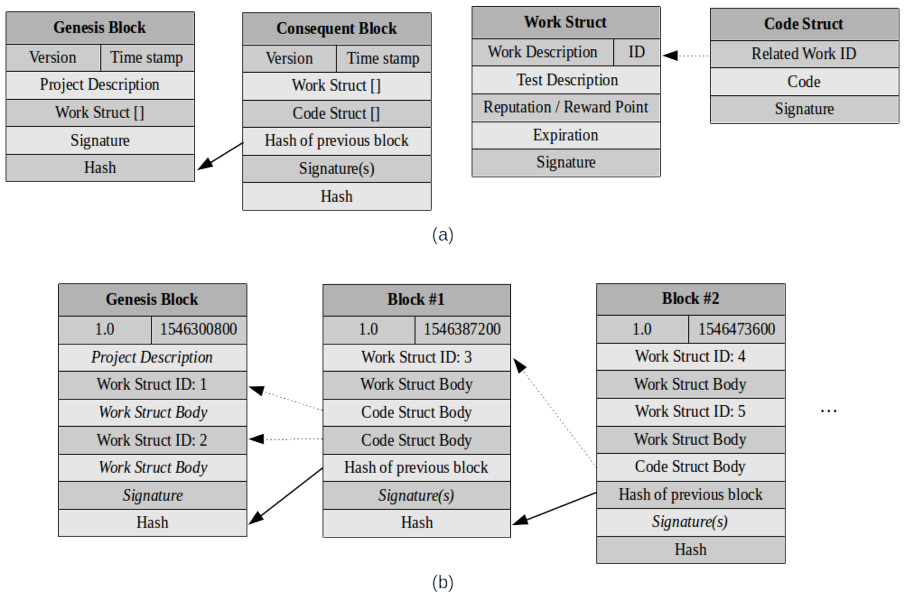

Fig. 2. Structures of genesis and consequent blocks, work and code structs (a); sample blockchain for the proposed model (b).

Fig. 2(a) illustrates the genesis and subsequent block structures such that the latter establishes a link to the previous block whereas code struct refers to work struct by providing its ID number. Fig. 2(b) shows a sample blockchain in which the blocks are linked together as indicated by the solid arrows. On the other hands, dashed arrows indicate the works accomplished by the pieces of code contained in the code structs.

To produce a block in the blockchain each candidate block shall be validated by a consensus algorithm. The consensus algorithm consists of five steps to finish the process:

1. Collect the candidate blocks generated by the testers to form a set of the work and code structs called a merged block.
2. In the merged block, identify all code structs that refer to a particular work struct. If a code struct contained in the merged block uniquely refers to a work struct, then confirm.
3. Otherwise:
   (a) Compute the contribution (i.e. total reputation/reward) of each developer who claims to accomplish the same work.
   (b) Calculate the waiting time for each developer (i.e. the elapsed time since the last activity of the developer recorded in the blockchain).
   (c) Count how many times the same code struct exists in the merged block (i.e. the number of votes for each code struct)
   (d) Construct a probability distribution function (PDF) of developer considering 3a, 3b, and 3c and choose a random developer in accordance with PDF. Confirm the code struct of the chosen developer and remove the others.
4. Repeat steps 2-3 for the remaining code structs until all work structs are uniquely referenced.
5. Add the digital signatures of the testers that have contributed the block. The testers get their reputation/reward share based on this information.

Note that the testers -the block creators- register their shares with their signatures in consequent blocks and the developer prove their ownership of the work done with their signatures in code structs according to the aforementioned consensus algorithm and proposed blockchain system. In addition, for each consensus phase, a PDF should be constructed based on the developer's activity and testers' preference. If the contribution (reputation/reward, or stake in a sense) or waiting time of a developer is higher, then the probability of the developer being chosen randomly will be higher. This scheme is expected to reduce the effect of the starvation. Similarly, if a developer is voted by more testers, the probability of his/her work being confirmed will be higher. The randomness can be realized by non-uniform pseudo-random generators seeded by the synchronized blockchain data.

3 Discussion

Previously, we have proposed a model for employing the economic mechanism design concept in the software development process. The aim was to consider software development as an economic activity inside an information exchange economy [25], and consequently, a basic model was formed on game theoretic principles to maximize the delivered value of a software project. The present study is designed to construct an incentive mechanism based on a blockchain to orchestrate software production. To promote trust and integrity, we form a blockchain oriented workflow model which would be valuable to address trust issues, especially when conducting large scale agile development [26]. We argue that there is a need for a reputation and a reward-based consensus approach for addressing the possible issues of software development more effectively. It has been observed that software development organization usually consider the software development process as a trust contract where trustworthiness demands a process-oriented viewpoint [27]. However, a blockchain-based decentralized production system can help us to construct an incentive system where practitioners compete for building the best work struct rather than having their tasks assigned by a central authority such as project manager or a team leader. Therefore, software practitioners should have to compete to create a consequent block using such a self-organizing approach.

This study defines *the blockchain-based software development* as all software activities are working with an implementation of a blockchain which is a data structure characterized by redundancy and validation for the sequentially ordered recording of data that uses public-key cryptography. We aim to create a distributed record of software development events based on a consensus method to agree whether a module (i.e. new set of code) is legitimate that can be assessed from many different perspectives. We aim to add a proof of work structure for validation of coding and testing to this framework. Therefore, this research initially aims to derive a framework based on a smart contract system to assess validation factors and other benefits of a consensus approach.

In addition, authors have observed that a significant number of agile software development organizations have started to rely on tools more than the process [28, 2]. We report that when software practitioners lose trust with their teammates, they are looking for a workaround on the working software process or sometimes request mediation from third parties. We hypothesize that a blockchain based software development model guides practitioners to solve decentralized trust issues in software development. In particular, a test-driven development environment can be operationalized, and collective code ownership and coding standards can be guaranteed using the smart contracts.

Further research should be undertaken to investigate the issues raised by our proposed model and its implementation both in a simulation environment and in a software development organization. There are still many unanswered questions about the description of the model, such as, how a fork is handled, how the growing speed of the blockchain is adjusted and how the reputation/reward points are distributed to the contributors. Furthermore, it is also necessary to elucidate the issues about the underlying network issues, such as, how the data packets are transferred and secured. A further study will be carried out with more focus on fake developers/testers and simulation of the system. It is also required (i) to investigate our approach for robustness to attacks such as altering a block in the chain, (ii) to establish a linkage to smart contracts, (iii) to explore how a payment system can be established using cryptocurrencies. Finally, to develop a full picture of a blockchain-based software production line, additional studies will be needed for the realization of the system as a whole.

References

1. Conway, M.E.: How do committees invent. Datamation **14** (1968) 28–31
2. Clarke, P., O'Connor, R.V., Yilmaz, M.: In search of the origins and enduring impact of agile software development. In: Proceedings of the 2018 International Conference on Software and System Process, ACM (2018) 142–146
3. Clarke, P., O'Connor, R.V., Leavy, B.: A complexity theory viewpoint on the software development process and situational context. In: Proceedings of the International Conference on Software and Systems Process, ACM (2016) 86–90
4. Verona, J.: Practical DevOps. Packt Publishing Ltd (2016)
5. Beck, K.: Test-driven development: by example. Addison-Wesley Professional (2003)
6. Hayes, W., Lapham, M.A., Miller, S., Wrubel, E., Capell, P.: Scaling agile methods for department of defense programs. Technical report, Carnegie Mellon University, Pittsburgh, United States (2016)
7. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4** (1982) 382–401
8. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops, IEEE (2015) 180–184
9. Zhang, Y., Wen, J.: An iot electric business model based on the protocol of bitcoin. In: 2015 18th International Conference on Intelligence in Next Generation Networks, IEEE (2015) 184–191
10. Vandervort, D.: Challenges and opportunities associated with a bitcoin-based transaction rating system. In: International Conference on Financial Cryptography and Data Security, Springer (2014) 33–42
11. Herbert, J., Litchfield, A.: A novel method for decentralised peer-to-peer software license validation using cryptocurrency blockchain technology. In: Proceedings of the 38th Australasian Computer Science Conference (ACSC 2015). Volume 27. (2015) 30
12. Peters, G.W., Panayi, E.: Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. In: Banking beyond banks and money. Springer (2016) 239–278
13. Devine, P.: Blockchain learning: can crypto-currency methods be appropriated to enhance online learning? (2015)
14. Sharples, M., Domingue, J.: The blockchain and kudos: A distributed system for educational record, reputation and reward. In: European Conference on Technology Enhanced Learning, Springer (2016) 490–496
15. Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: Architecture, consensus, and future trends. In: 2017 IEEE International Congress on Big Data (BigData Congress), IEEE (2017) 557–564
16. Conoscenti, M., Vetro, A., De Martin, J.C.: Blockchain for the internet of things: A systematic literature review. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), IEEE (2016) 1–6
17. Novo, O.: Blockchain meets iot: An architecture for scalable access management in iot. IEEE Internet of Things Journal **5** (2018) 1184–1195
18. Fernández-Caramés, T.M., Fraga-Lamas, P.: A review on the use of blockchain for the internet of things. IEEE Access **6** (2018) 32979–33001

19. Porru, S., Pinna, A., Marchesi, M., Tonelli, R.: Blockchain-oriented software engineering: challenges and new directions. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), IEEE (2017) 169–171
20. Marchesi, M.: 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), IEEE (2018) 1–1
21. Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., Hierons, R.: Smart contracts vulnerabilities: a call for blockchain software engineering? In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), IEEE (2018) 19–25
22. Wohrer, M., Zdun, U.: Smart contracts: security patterns in the ethereum ecosystem and solidity. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), IEEE (2018) 2–8
23. Beller, M., Hejderup, J.: Blockchain-based software engineering. In: 41st International Conference on Software Engineering (ICSE), New ideas and Emerging Results (NIER) track. (2019)
24. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: International Conference on Financial Cryptography and Data Security, Springer (2017) 494–509
25. Yilmaz, M., O'Connor, R.V., Collins, J.: Improving software development process through economic mechanism design. In: European Conference on Software Process Improvement, Springer (2010) 177–188
26. McHugh, O., Conboy, K., Lang, M.: Agile practices: The impact on trust in software project teams. Ieee Software **29** (2012) 71–76
27. Yang, Y., Wang, Q., Li, M.: Process trustworthiness as a capability indicator for measuring and improving software trustworthiness. In: International Conference on Software Process, Springer (2009) 389–401
28. O'Connor, R.V., Elger, P., Clarke, P.M.: Continuous software engineering—a microservices architecture perspective. Journal of Software: Evolution and Process **29** (2017) e1866