TU Delft

Delft University of Technology

Blockchain-based Software Engineering

Beller, Moritz; Hejderup, Joseph

Publication date
2018

Citation (APA)
Beller, M., & Hejderup, J. (2018). Blockchain-based Software Engineering.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Blockchain-based Software Engineering

Moritz Beller
Delft University of Technology
The Netherlands
m.m.beller@tudelft.nl

Joseph Hejderup
Delft University of Technology
The Netherlands
j.i.hejderup@tudelft.nl

***Abstract*—Blockchain technology has found a great number of applications, from the banking sector to the Internet of Things (IoT). However, it has not yet been envisioned which problems in Software Engineering Blockchain technology could solve. In this paper, we coin this field "Blockchain-based Software Engineering" and exemplify how Blockchain technology can be applied to two core Software Engineering problems: Continuous Integration Services such as Travis CI and Package Managers such as apt-get. We believe that Blockchain technology could help (1) democratize and professionalize Software Engineering infrastructure that currently relies on free work done by few volunteers, (2) improve the quality of released artifacts, and (3) increase trust in ubiquitously used infrastructure such as GitHub Travis CI.**

***Index Terms*—Blockchain, Distributed System**

I. Introduction

The advent of Blockchain technology, i.e., systems that rely on a distributed, tamper-proof ledger shared among multiple parties, is said to revolutionize industries across the board—and has partly done so already [1]: Bitcoin [2], the world's first cryptocurrency, allows one to send money on a peer-to-peer basis without a central intermediary, disrupting the banking sector [3]. Ethereum [4], a Blockchain protocol often referred to as "Blockchain 2.0," allows parties to indulge in *smart contracts,* a way of codifying the exchange of goods or the execution of other, predefined behavior in a *transaction*. The content of a smart contracts can be of great flexibility because they are written in a pseudo-Turing complete language. A smart contract can be seen as the automated execution of a contract between parties [5]. All contracts and transactions are stored for everyone to inspect on the Blockchain, serving both as a historical archive as well as a protection mechanism against fraudulent behavior. Despite the public visibility, both the sender and the receiver of a transaction on the Blockchain can remain anonymous.

Blockchain technology, however, has more applications than these two widely known examples of permissionless Blockchains, i.e., Blockchain protocols in which any party without prior trust can simply partake. Companies are exploring permissioned Blockchains as a way to do mutual business more securely, which should not be visible to outsiders. For example, the startup Provenance offers supply-chain auditing of consumer goods ("is my fish sustainably harvested?") [6], [7]. Governments are looking to formalize transactions with their citizens; even IoT devices could make use of it [8].

Given that large Software companies such as IBM have started to heavily invest in Blockchain technology, it is somewhat surprising that application possibilities of Blockchain-technology have not yet been extended to how we make software itself. In this visionary paper, we want to explore how Blockchain technology could revolutionize Software Engineering (SE) under the term Blockchain-based Software Engineering (BBSE): We first describe the core principles and guarantees Blockchain technology provides and which of them would be useful for SE. We then explore a handful of existing SE problems and sketch how we might use Blockchain technology to solve them. While this paper cannot (and does not aim to) provide a complete list of SE problems that would benefit from Blockchain support, we sketch how BBSE promises to solve fundamental issues plaguing SE as a craft: Professionalization, quality, and trust.

II. Related Work

To the best of our knowledge, this vision paper is completely different from most existing works in that it introduces and applies the ideas behind Blockchain to the practice of SE itself.

Perhaps most closely following this principle is Nikitin et al.'s work on Chainiac [9], a proactive software-update system. Their practical implementation shows among others the feasibility of verified builds, which we rely on as part of a Blockchain-based CI system. The majority of existing work, however, addresses SE challenges on Blockchain, such as smart contract verification [10], [11], vulnerability detection [12], and interoperability [13]. Bell et al. [14] propose a data management system with Blockchain guarantees to ensure researchers publish tamper-proof and reproducible data sets.

III. A Blockchain Primer

In this section, we present some of the core ideas behind Blockchain technology. We mean to equip readers unfamiliar with the technology with its basic ideas and focus on concepts we consider relevant for SE.

A. Terminology

Formally, by Blockchain, we mean a single file that represents an interlinked set of blocks, each daisy-chained to its parent.[1] The Blockchain protocol ("specification") governs what constitutes a valid block. The implementation of that is a

[1]In Bitcoin's case, this file was 161GB large on 2018-6-15, per https://charts.bitcoin.com/chart/Blockchain-size.

client.[2] Therefore, most colloquial references to "Blockchain" refer to the protocol behind it, rather than the file itself. A Blockchain itself is stateless; the current state is implicit and could be verified by every client through executing all of its blocks in order (or, to abbreviate the process, by relying on the Merkle tree). Every block contains a finite number of transactions (e.g., in Bitcoin's case, "Send money from A to B"). The low number of transactions that can be included in a block[3] and the limited growth[4] are the prime reasons for Bitcoin's low throughput [15].

B. Types of Blockchains

In principle, we distinguish two forms of Blockchain protocol: permissioned and permissionless Blockchains. Permissioned Blockchains are restricted to a set of known actors, implying the need to identify themselves (examples are most business-to-business applications). Permissionless or open Blockchains allow anyone to participate in the network, either as a miner or as a regular full node.

Miners bring order to the Blockchain by picking a set of transactions from the transaction pool (typically the most lucrative ones, i.e., willing to pay the highest fee for their inclusion in the chain) and generating a suitable block that can be appended to the chain. In the case of Bitcoin, this means solving a hash puzzle, an otherwise pointless operation of trying to find a hash h for the current block, a triple $\langle$`parentHash, salt, transactions`$\rangle$ such that h begins with an ever-growing number of 0s. At its essence, this *proof-of-work* hash algorithm makes Bitcoin consume more power than the Czech Republic as of June 2018 [16]. Permissioned Blockchains typically do not require proof-of-work, and newer open Blockchain protocols try to employ a concept called *proof-of-stake* to fix the power consumption. The incentive to create, i.e., "mine", new blocks B_{n+1} stems from the rewards: once a block has been found and successfully been appended to the chain, the miner receives a reward, in addition to the transaction fees contained in the block. Thus, $\text{reward}(B_{n+1}) = \text{block\_bounty}(B_{n+1}) + \sum \text{transaction\_fees}(B_{n+1})$.

Regular full nodes are nodes which do not mine, but pertain full copies of the Blockchain and the set of waiting-to-be-included transactions in the *mempool*. Bitcoin participants do so by specifying the transactions they want to commit, as well as the maximum fee they are willing to pay for the transaction to be included in the Blockchain. Full nodes can accept these requests from users and put them onto the mempool.

C. Trust and History

Because there is no central authority to trust, Blockchain-based solutions offer a high degree of robustness against single-point-of-failure type scenarios. However, when presented with two different, valid Blockchains, how would a client know which one to trust? This has been the question of so-called consensus protocols such as Paxos, which have existed long before the idea of Blockchains came up [17]. In Bitcoin's case, the so-called Nakamoto consensus is achieved by selecting the longest chain, i.e., the one which most computational work has gone into. Achieving consensus on a Blockchain is a hard problem, especially in the presence of adversarial nodes (so-called byzantine behavior), but practical solutions such as Bitcoin or Ethereum show that it is feasible.

We can summarize that Blockchain-based technology allows us to build a robust, distributed system including a pay-mechanism (the currency, e.g., Bitcoins or Ethers), the execution of predefined behavior (via smart contracts), a working consensus protocol, and a tamper-proof history of every transaction. The actors in such a system can be anonymous, pseudo-anonymous, or known. We argue that these characteristics are of high significance for certain SE problems. Core among them stand the Blockchain's promises to *decentralized trust* and *verifiability*.

D. Smart Contracts

Ethereum allows the execution of smart contracts. A smart contract is nothing more than a program, i.e., a set of functions, which are stored on the Blockchain. Since space on the Blockchain is expensive, Ethereum contracts are practically confined to about 600 lines of code. In Ethereum's case, the contract is compiled to opcodes and executed by the EVM, the Ethereum Virtual Machine. In our vision, the ability to execute code has the strongest implications for Blockchain-based SE.

E. Security

While the applications for Blockchain-based software are seemingly endless, so are the mistakes one can make using them. Several attacks have been tried on both the Bitcoin and the Ethereum networks. Per design, so-called 51%-attacks, in which a malicious attacker controls the majority of the network, pose the largest threat.[5]

However, in practice, successful attacks have been confined largely to exploiting user- rather than protocol-induced bugs. For example, the DAO, an investor-backed venture capital fund implemented on Ethereum, lost 55 million US-$ because of a relatively trivial bug in its smart contract [19]. Nevertheless, Blockchain-based technology is just in its infancy.

IV. Blockchain-based Continuous Integration

If Continuous Integration is a success story in SE, then Travis CI is the epitome of a success story: More than half of all projects doing CI on GitHub, do it with Travis CI. Travis CI provides a build environment for developers to test and deploy their code changes. Travis CI internally does this by renting computing power from AWS. Moreover, Travis provides a historic view of previously executed builds, a log that lends itself to a Blockchain application.

[2]The Ethereum project, for example, maintains three such equitable clients; `Geth` written in Go, `Eth` in C++, and `Pyethapp in Python`.

[3]Defined by the block size ranging on average below 900kB, see https://bitinfocharts.com/comparison/bitcoin-size.html

[4]On average, a new block is found only every 10 minutes

[5]Eyal and Sirer have shown that in the case of Bitcoin, even controlling less than 50% might suffice. [18]

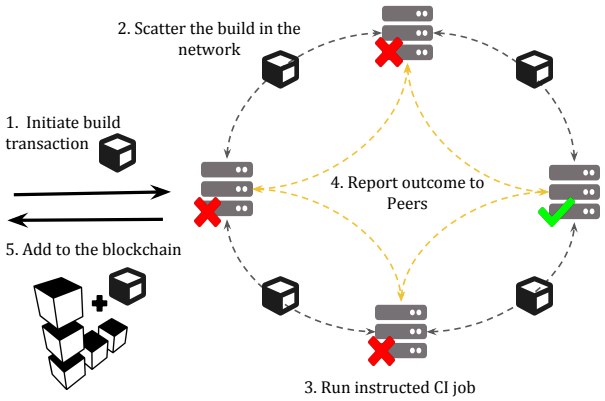

Fig. 1. Example of a BCI

In Figure 1, we propose a Blockchain-based CI system called BCI. After a developer enters a build and its reward price (similar to gas price in Ethereum) in the mempool (1), it gets picked up faster or slower (2) by the workers (3), who perform the build. They report the build outcome to each other (4) and, if they find consensus (e.g., "pass"), add the transaction to the Blockchain (5). Inspired by Etherum's ability to execute functions, our nodes perform build tasks in a shielded environment similar to the EVM (e.g., Docker images to isolate the builds from the host system). The execution of build tasks could serve as a viable proof-of-work by hashing and signing the build log output, and replace the wasteful proof-of-work concept Bitcoin currently employs. Moreover, the highly parallel nature of CI with its many build and test execution environments lends itself toward expediting via a large set of worker nodes.

This design alleviates a number of problems of traditional CI (TCI) services. First, as monolithic systems, TCI outages happen and every outage disrupts the workflow of its users. Implemented as a distributed system running on the Blockchain, there would no longer be a single point of failure. As a quasi-monopoly, a TCI dictates private hosting prices. A BCI opens a market for computing power in which everyone can partake and earn money with otherwise idling resources. This in turn also means that the economic rules of demand and supply regulate build prices, not a single company. Most TCI providers only give their users a fixed number of n ($n \leq 4$) build workers. There is no flexibility to deal with an urgent bug fix that needs to be tested as soon as possible. A Blockchain-based CI would allow developers to specify a high transaction cost on this one build. All nodes would prioritize the execution of this build job, since they earn a high margin. Developers would be more flexible, but only always pay what they need in the moment. Lastly, the history of builds might be lost. If Travis CI or another TCI goes bankrupt, we might lose access to its "treasure trove of build log data" (Travis CEO). A Blockchain of CI builds would self-serve as a distributed archive.

Practical challenge of implementing a BCI remain.

1) How to store build logs? Should the full logs be part of the Blockchain, will this lead to a quickly-growing file that could soon outgrow Bitcoin's 150 GB ledger. Possible solutions could be the light clients of e.g. Etherum, where light nodes do not need to know the full ledger, thanks to Merkle trees. Other solutions could be heavy compression, storing information off-the-chain, or relying on existing Blockchain-based file share solutions such as FileCoin.[6]

2) How to resolve non-deterministic builds? For others to verify a build, builds themselves need to be reproducible. An obvious solution is to execute builds in encapsulated containers. But how can we combine this with the advantages of having a multitude of different build environments at hand, which containers would diminish?

V. Blockchain-based Package Manager

Reuse is a central element in SE. We not only use package repositories such as Debian's apt-get to manage binaries, but also to drive software development itself, with library repositories such as Java's Maven Central or JavaScript's npm. npm used to do essentially no vetting of the quality of new releases. This was painstakingly obvious in the left-pad incident, in which an Open-Source Software (OSS) developer who turned malicious caused great disturbance among the community by removing a trivial package on which thousands of other npm packages relied [20].

Limited measures to avoid such ripple effects of breaking releases already exist. For example, Debian uses a trust network that relies on personal connections. With their signature, Debian maintainers verify that they have done proper integration testing of a package.[7] Even so, this 1) cannot prevent random malicious actions, 2) puts a lot of work on few shoulders, 3) does not prevent package breakages,[8] and, 4) implies that testing is limited to what a package maintainer can do.

To remedy these problems, we propose BAPT, a Blockchain-based package repository. BAPT encapsulates a verifiable community-driven regression test framework for package repositories. Similar to BCI, every participant in BAPT can pick a new release candidate from the "mempool", verifies that the release works as intended and does not break compatibility with downstream clients.

Important questions on how to implement BAPT remain:

1) How do we find consensus on what is a "good release?" Traditional majority-based consensus voting means that it would be enough if 51% of integrators verify the package works for them. This is clearly not suitable for bugs that only appear in some environments. On the other hand, a veto-based consensus would mean that one malicious attacker could prevent a release. We need to find the right middle ground, perhaps by replicating integrations in the same environment and comparing their outcome. This might require base research on new consensus protocols.

2) How do we test packages? Should we rely on manual testing like Debian does? What is the proof of work in this case? For

[6]https://filecoin.io/

[7]https://debian-handbook.info/browse/stable/sect.becoming-package-maintainer.html

[8]https://lists.debian.org/debian-devel/2002/09/msg00066.html

semantic versioning, we could make downstream clients of a library run their tests to verify that a proclaimed non-breaking change is indeed not breaking. This adds a layer of trust by third parties, essentially democratizing the decision to properly release a package (i.e, putting it on the Blockchain) to a wider audience of the community. It empowers the users of a library. Note that it does not take power from the creator, as they can still release software as they wish—they just have to adhere to the correct semantic version.

VI. Discussion

In the following, we discuss the larger implications that BBSE could bring with it to the SE landscape.

Professionalization. In SE, there exist many micro tasks. These are not per se difficult to execute, but require human diligence, e.g., to maintain packages, or CPU power, e.g., to execute builds. Like other distributions, Debian uses a fixed pool of mostly voluntary maintainers. While this core principle of OSS is not per se bad, unfortunately, it leads to a significant amount of peripheral packages being outdated because of inactive maintainers. Instead, in BAPT, an open package marketplace could flourish in which everyone could participate, propose new packages be included, and verify the work of others. The reward for taking part in this transparent and open process comes in the form of a small payment, both as a transaction fee and bounty. The market could reward urgent updates or builds with a higher bounty.

Improved Quality. In addition to being out of date, packages often break in different integration environments than what the maintainer has available to them. In a distributed system where a transaction gets verified by others, such as BAPT or BCI, the quality of releases or builds could improve due to a greater diversity in testers and equipment available to them. Moreover, it is of vital importance then that these results be reproducible, a requirement the Debian community strives toward, but that is still often violated.[9]

Trust. Whenever there is a monopoly, trust becomes an issue—be it for pricing or other reasons. While Debian's maintainer system is a meritocracy, the left-pad incident has shown that it is no guard against malicious behavior. With Blockchain technology, we can replace a centralized system—be it person-centralized or technically likes Travis CI or GitHub—with a system verifiable by everyone. Additionally, as a decentralized system, it might offer a higher-availability and be more resilient to partial network outages.

VII. Conclusion

While a lot of hype surrounds Blockchains, behind the noise is a possibly breakthrough-enabling technology. This paper can merely scratch the tip of the iceberg of the opportunities Blockchain-based SE (BBSE) can enable. We have exemplified that we can use BBSE to solve elementary problems in SE. In particular, we have sketched designs for

- a distributed, democratized build service, called BCI,
- a user-run package management system, called BAPT.

As these examples show, Blockchains seem particularly promising to problems that deal with centralization and trust. Many examples include a monetary component, a question often raised in SE ("but how to make money with this?"). We believe that paying small amounts, instead of relying on free work by few volunteers in the OSS community for micro tasks such as verifying a new release, can have far reaching consequences in the professionalization of the entire SE infrastructure. While we outlined ideas for how to tackle SE challenges using Blockchain technology, many exciting research opportunities and their implementation remain for future work, e.g., a truly distributed version of Git on the Blockchain or, abstracting from BCI, a decentralized computing platform competing with the likes of AWS.

[9] https://wiki.debian.org/ReproducibleBuilds

References

[1] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[3] Allen Scott. 10 reasons why bitcoin is better than paypal, 2018. https://cointelegraph.com/news/10-reasons-why-bitcoin-is-better-than-paypal.

[4] Wikipedia. Ethereum, 2018. https://en.wikipedia.org/wiki/Ethereum.

[5] Primavera De Filippi De Filippi. *Blockchain and the Law: The Rule of Code*. Harvard University Press, 2018.

[6] Project Provenance Ltd. Blockchain: the solution for transparency in product supply chains, 2015. https://www.provenance.org/whitepaper.

[7] Henry M Kim and Marek Laskowski. Toward an ontology-driven blockchain design for supply-chain provenance. *Intelligent Systems in Accounting, Finance and Management*, 25(1):18–27, 2018.

[8] M Walport. Distributed ledger technology: Beyond blockchain. uk government office for science. Technical report, Tech. Rep, 2016.

[9] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. Chainiac: Proactive software-update transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, 2017.

[10] Quanqing Xu, Chao Jin, Mohamed Faruq Bin Mohamed Rasid, Bharadwaj Veeravalli, and Khin Mi Mi Aung. Blockchain-based decentralized content trust for docker images. *Multimedia Tools and Applications*, pages 1–26, 2017.

[11] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. Blockchain-oriented software engineering: challenges and new directions. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 169–171. IEEE Press, 2017.

[12] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering? In *Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on*, pages 19–25. IEEE, 2018.

[13] Peng Zhang, Jules White, Douglas C Schmidt, and Gunther Lenz. Applying software patterns to address interoperability in blockchain-based healthcare apps. *arXiv preprint arXiv:1706.03700*, 2017.

[14] Jonathan Bell, Thomas D LaToza, Foteini Baldmitsi, and Angelos Stavrou. Advancing open science with version control and blockchains. In *Software Engineering for Science (SE4Science), 2017 IEEE/ACM 12th International Workshop on*, pages 13–14. IEEE, 2017.

[15] Wikipedia. Bitcoin scalability problem, 2018. https://en.wikipedia.org/wiki/Bitcoin_scalability_problem.

[16] Digiconomist. Bitcoin energy consumption index, 2018. https://digiconomist.net/bitcoin-energy-consumption.

[17] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[19] Matt Leising. The ether thief, 2017. https://www.bloomberg.com/features/2017-the-ether-thief.

[20] Rabe Abdalkareem, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid, and Emad Shihab. Why do developers use trivial packages? an empirical case study on npm. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 385–395. ACM, 2017.