

BIM and Blockchain

A Decentralized Solution for a Change Management Workflow in Construction Projects

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

David Peherstorfer, BSc

Matrikelnummer 00929021

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Mag. Dr. Manuel Wimmer

Mitwirkung: Univ.-Ass. Dipl.-Ing. Galina Paskaleva

Wien, 10. April 2019

David Peherstorfer

Manuel Wimmer

BIM and Blockchain

A Decentralized Solution for a Change Management Workflow in Construction Projects

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

David Peherstorfer, BSc

Registration Number 00929021

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Mag. Dr. Manuel Wimmer

Assistance: Univ.-Ass. Dipl.-Ing. Galina Paskaleva

Vienna, 10th April, 2019

David Peherstorfer

Manuel Wimmer

Erklärung zur Verfassung der Arbeit

David Peherstorfer, BSc
1100 Wien, Österreich

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. April 2019

David Peherstorfer

Danksagung

Ich möchte besonders meinen BetreuerInnen Galina Paskaleva und Manuel Wimmer danken, die mich während dem Verfassen dieser Diplomarbeit unterstützt haben. Des weiteren möchte ich Juan Franco vom Nethereum Projekt für die technische Unterstützung bei der Arbeit mit der C# Ethereum Library danken.

Zu guter Letzt gilt mein Dank meinen Eltern Cornelia und Hans, meiner Freundin Kristina, meinem Bruder Tobias und meinen FreundInnen, die stets an mich geglaubt und mich unterstützt haben.

Acknowledgements

I would like to thank my advisors Galina Paskaleva and Manuel Wimmer for supporting me while writing this diploma thesis. Furthermore, I want to thank Juan Franco from the Nethereum project for the technical support when working with the C# Ethereum library.

Last but not least I want to thank my parents Cornelia and Hans, my girlfriend Kristina, my brother Tobias and my friends, who always believed in me and supported me.

Kurzfassung

Durch den Digitalisierungsgap in der Baubranche gibt es ein großes Potential zur Prozessoptimierung. Neue digitale Technologien, wie das Building Information Modelling (BIM), werden immer mehr von den Stakeholdern in diesem Bereich angenommen. Die Blockchain auf der anderen Seite ist eine sehr neue und innovative Technologiedomäne, welche in den letzten Jahren extrem gewachsen ist und in der momentan die passenden Use Cases für die neue Technologie gesucht werden. Im Speziellen das noch neuere Untergebiet der Smart Contract Entwicklung hat die Vision von vielen neuen Anwendungen beflügelt, wobei nicht immer klar ist, ob diese so wie in der Vorstellung umgesetzt werden können, beziehungsweise ob dabei überhaupt ein Bedarf für eine dezentrale Lösung besteht. Bei einem Bauprojekt müssen Änderungen in einem BIM Modell immer vom passenden Stakeholder freigegeben werden. Deshalb haben wir in dieser Arbeit BIM Modelle, welche in einem Git Repository verwaltet werden, mit einem Freigabeverwaltungsworkflow, welcher als ein Smart Contract auf der Ethereum Blockchain umgesetzt wurde, kombiniert. Dadurch soll der Workflow transparent, nachvollziehbar und dessen Ergebniss im Nachhinein unveränderlich werden. Das Ziel dieser Arbeit ist es einen Smart Contract Prototypen zu erstellen und diesen mit anderen (off-chain) Lösungen im Bezug auf Kosten und Sicherheit zu vergleichen.

Abstract

There is a big potential for process optimizations, due to the digitalization gap in the construction business. New digital technologies, as the Building Information Modelling (BIM), are increasingly being adapted by the stakeholders in this area. On the other hand, blockchain is a very new and innovative technology domain which has grown immensely in the last several years, and where people are now trying to find the right use-cases. Especially, the even newer field of smart contract development has opened the door for a large amount of possible applications, where it is neither clear if these can actually be implemented as envisioned, nor if there is even a need for a decentralized solution at all. In a construction project, changes on BIM models are only to be approved by the appropriate stakeholder. Therefore, we have combined the BIM models, which are stored using a Git repository, with a release management workflow, which is realised as a smart contract on the Ethereum blockchain. This enables the workflow to be transparent, traceable and its results to be immutable. The goal of this work is to create a prototype and compare it to other (off-chain) solutions and to evaluate if an application of a combination of BIM and blockchain yields an advantage in terms of costs and security.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Context	1
1.2 Problem Statement	2
1.3 Solution and Methodological Approach	4
1.4 Structure of this Work	5
2 Preliminaries	7
2.1 Building Information Modeling	7
2.2 Introduction to Blockchains	8
2.3 A Combination of a smart contract and BIM	14
3 State of the Art	15
3.1 BIM	15
3.2 Blockchain	16
4 Realization	21
4.1 Requirements	21
4.2 Operating Principles of the Prototype	22
4.3 Architecture	23
4.4 SIMULTAN Software Tool	24
4.5 Git	25
4.6 Smart Contract	26
4.7 The Frontend	33
5 Evaluation	39
5.1 Goal	40
5.2 Questions and Metrics	40
5.3 Evaluation Plan	41

5.4	Evaluation Results	43
5.5	Evaluation Summary	58
5.6	Threats to Validity	61
6	Summary and Future Work	63
	List of Figures	67
	List of Tables	69
	Bibliography	71

Introduction

1.1 Context

A McKinsey report from 2017 shows that the construction sector is one of the world economies' largest industries [BWM⁺17]. Nevertheless, compared to other domains, it is lacking in terms of productivity and innovation. According to the McKinsey report, it is also one of the least digitized industry sectors, which means there is a huge digitalization gap that can be filled. This could offer new profitable business opportunities for construction companies.

Moreover, another problem that this domain is facing is the prevailing over-regulation which binds stakeholders of construction projects to very strict laws. They have to abide by local laws and the project managers have to hold every one of the stakeholders accountable for the fulfillment of their commitments. This accountability process is often realised using paper intensive workflows and traditional contracts.

In a construction project there are multiple stakeholders who occupy different roles. There is the role of the construction developer, who represents the client and the construction manager, who oversees the construction site on behalf of the building's designers and communicates with the construction developer, building authorities and other external parties. The construction manager is an important liaison for all workers on the construction site. Additionally, depending on the type of construction project, there are architects, statisticians, building authorities, various contractors, etc. All these stakeholders have a different function in and perspective of the project, and as a result, there needs to be a traceable and well-documented agreements on the current state and the target state of a project, often on a daily basis.

There is a multitude of documents used in construction projects. These documents range from contracts over construction diaries to models and plans. Additionally, photo and video documentation is used to record the actual state and provide a comparison to the

target state. The construction manager is in charge of the creation and collection of all relevant documents. These documents then need to be signed by the construction developer, who commissioned the project. It is very important that every contract partner receives the needed documentation, so that everyone involved has a thorough perspective on what has already been done and what is still missing. In case of a dispute the court consults these as a base of their decision. Those documents can be used as evidence for construction delays or unscheduled additional costs, which often happen in these projects [Kva18].

Documentation is still often created on paper and only later digitized to make the storage easier and to facilitate the information search. Additionally, there are various highly specialised software products used to create and handle the documentation. These are used in all areas of a construction project, from the management of 3d models to accounting and budgetary estimations. This type of software has a very narrow view of the project, and the transfer of information between different products can prove very cumbersome. Due to this problem, the construction industry has developed a solution for a unified data management approach, which can be applied throughout the whole project by different stakeholders.

BIM (Building Information Modeling) is slowly being adopted in construction projects and is set out to be the industry standard of the future. This progress in adoption is vastly driven by requirements of using BIM models in public construction projects. BIM models are a multidimensional digital building information representation, on which multiple users are able to work collaboratively. Since everyone has to be held accountable for adaptations on such a model, the changes made have to be traced back to their authors. This work describes a solution for a change management system where the committed adaptations of the model and their respective approval are not only securely stored, but also fulfill the properties of traceability and immutability. To achieve these features we have implemented the change management system in a smart contract on the Ethereum blockchain. Blockchains with their combination of hash trees and distributed ledger Technology are inherently created to be decentralized. Once the information has been included into a block, it is completely traceable and immutable. Furthermore, we are going to present and evaluate a prototype for our solution.

In this chapter we are first going to present our use case, then give an introduction to our methodology and finally describe the structure of this work.

1.2 Problem Statement

In this section we will model our use case and describe the resulting requirements for our solution. The described use case is a simplified version of a project's building model change management.

We have a construction project, which has the goal to construct a new building for a private company with one construction manager who is responsible for the whole project. The construction project has multiple stakeholders who are working in different domains.

There are not only technical but also legal and financial specialists involved. Additionally to the construction manager, the project has a developer, an architect, a person from the building authorities and a contractor as stakeholders. The developer makes a request to the project manager to make a change to the building's roof. As a next step, the construction manager refers the request to the architect who implements the changes it into his plans. Then the updated plans need to be checked and approved by the construction manager, the owner, the building authorities and the contractor:

- The project manager has to check if the changes made fit into the budget.
- The developer has to check if his request has been fulfilled.
- The building authorities need to check if the changes of the roof are compliant to local law.
- And the contractor has to check if the changes can be implemented by her workforce.

Often in construction projects the architect sends the updated plan to the construction manager who forwards the plans to the other stakeholders. It's still a common practice that these print the plans, sign them and send them back to the construction manager via mail. Once he receives all the signatures on the plans he can instruct the contractors to start implementing the changes. If one party does not approve the plan a new version has to be proposed and the process repeats itself. This workflow is unnecessarily time consuming because of the non digital way of information transportation and the multiple iterations of the process. There is also no single source of truth because there could be multiple versions of a plan existing at the same time, which could lead to costly confusions. In the current process, the signatures on printed plans can be used for holding stakeholders accountable. If something does not work as expected the construction manager can use the signed plans in arbitration or in court, if it comes to a lawsuit.

We propose the following combination of multiple standards and technologies and a new process for change management as a new solution to this use case. To ensure there is only a single source of truth we propose to use the Building Information Modeling (BIM) standard for construction communication which will be hosted on a Git repository. A BIM model not only provides a means to save 3D information but additional information about a construction project, such as time estimations and costs, as well. The Git repository will ensure that all stakeholders have reliable access to the current version of the BIM model and will enable them to commit new changes to it.

Furthermore, we propose a change management workflow that ensures that the stakeholders can be held accountable for their approval of a change. Based on the contracts information the construction manager will be able to name responsible stakeholders for each change, so that they can give their approval to it. The Ethereum network, with it's ability to process smart contracts, will provide a means to implement this workflow. It provides immutability and traceability properties by using a distributed ledgers technology and digital signatures.

This change management process can not only be used for construction projects but also in other domains, such as, e.g., software development. However, we will only focus on our described use case due to the limited scope of this work.

1.3 Solution and Methodological Approach

The goal of our work is to create the first working experimental prototype, the ChangeManager, and to compare it to a completely on-chain and to a completely off-chain solution. The usage of experimental prototyping aims at finding the best solution for a software system as described by Kappel and Nierstrasz [KN89].

The prototype is going to be a solution that fulfills the basic requirements, which can be found in section 4.1 and will not yet be the perfect solution for our use case but a starting point. The goal is to implement a system which can be used for our evaluation as a point of reference. Possible improvements on the prototype and further findings will be discussed in chapter 6.

For the subsequent evaluation we will use the GQM (Goal, Question, Metric) process [CR94] and apply it to our ChangeManager prototype and two other solutions. For this we will define different goals, which the solutions should fulfill, e.g. "The solution needs to be fast". Then we will find questions to test for this goal, e.g. "How fast can you store files with this solution?". As the third and last step, we will develop metrics to answer the questions. A metric for our example could be, "How long does it take to store a 5 GB file with this solution?". After the setup of the GQM framework, we will use it to evaluate the different change management solutions which will consist of the following:

- The first solution is a Git only approach, where no blockchain technology will be involved.
- The second solution is our ChangeManager prototype which we will describe in detail in chapter 4.
- And the last solution is uses only the Ethereum Blockchain as the underlying technology and completely disregards Git as a means to store the models.

Due to the limited scope of this work, we discuss the first and the last solution only on a theoretical basis. Based on this evaluation, we will present a comparison of the described approaches.

1.4 Structure of this Work

In chapter 3 we are going to give an overview on the current research done in the areas of BIM, blockchain, smart contracts and the combination thereof. Then, in chapter 4 we are going to describe our ChangeManager prototype and its architecture and the reasons why we have chosen this hybrid approach. We will also give an introduction into the utilised technologies, such as smart contracts, Git and BIM. In chapter 5 we will present an evaluation, where we describe the Goal, Question, Metrics approach, define our evaluation plan and evaluate a Git only, our hybrid (blockchain combined with Git) and a blockchain only approach using the previously defined metrics. Furthermore, we will discuss possible threats to the validity of the evaluation and its final results. In conclusion, in chapter 6 we will discuss our results and give an outlook into further research, which can be done in the area of the combination of BIM and blockchain.

Preliminaries

2.1 Building Information Modeling

BIM is a digital process with the potential to deeply transform the construction business and its workflows. It provides an integrated approach for the management and the creation of digital representations of buildings, as it can be used throughout all phases of the lifecycle of a building, from the planning to the maintenance phase. BIM also supports the stakeholders of the project by facilitating the exchange and the interoperability of digital building information. A BIM model not only contains a 3D (width, height and depth) representation of the building but also provides much information including time and costs. Project stakeholders, such as architects, engineers, main contractors, building operators, construction managers, etc., can use this virtual representation in each step of the project and adapt the model for the needs of their domain. When fully implemented, BIM is a huge improvement in knowledge management, since there is only one single source of truth to which all stakeholders can refer to.

The idea of having a central model for construction projects originated in the 1970s [EDG⁺74]. ArchiCAD, which was launched in 1987, is considered the first commercial BIM product. Autodesk published a whitepaper in 2002 on the topic of BIM, which put a lot of software vendors onto the track of digital building models. This was also the year when the term BIM started to become part of the standard nomenclature [Aut02][KP12].

BIM can be both a data exchange format and a communications standard. However, in its most fully realised form, it is a process. There are mainly proprietary file formats developed by various software companies, such as Autodesk. Proprietary file formats have the disadvantage of only being open- and editable in the software for which they were initially been developed for, because the vendor in many cases does not disclose how the files are structured internally. However, there are also non-proprietary file formats, which are neutral and developed by international consortiums. Most commonly used

is the Industry Foundation Class (IFC) specification, which is a standard for openBIM. It has been developed by buildingSMART¹, which is a non-governmental organization. The IFC file format is vendor-neutral and can even be imported or exported by multiple proprietary tools. Its intended use is for transferring data from a proprietary software to an open source software and back.

SIMULTAN is a research project involving various research partners from the TU Wien. It "addresses questions about the city of the future with a view to planning sustainable, liveable cities of tomorrow. The goal is to produce a workable tool, in the form of software to support the planning and decision-making process, which will allow experts from various disciplines to jointly design, optimise, build and operate building complexes."² In the context of this project, a proof of concept for a BIM data model and an interactive tool to be used by an interdisciplinary team has been developed [BBF⁺18]. We use this tool as a reference for our use case and also for the development of our prototype, because it has been tested on real and very well documented construction projects, which are already in the construction phase. We will give a further overview of the software in chapter 4.

One concern that is not adequately addressed in BIM models is the accountability of all stakeholders who can make changes to the model. The issue of handling permissions, who is allowed to publish what, and what changes get accepted, is what we address in this work. BIM models in combination with a change management system is an integrated process in our prototype. The change management part is implemented on the blockchain as a smart contract. In the next section we give an introduction into the area of blockchains.

2.2 Introduction to Blockchains

In 2008 Satoshi Nakamoto published a paper called "Bitcoin: A Peer-to-Peer Electronic Cash System" wherein he proposed a new way to securely handle monetary transactions between untrusted participants through the combination of multiple already known and before only separately used technologies [Nak08]. This proved to be the theoretical framework for the upcoming cryptocurrencies, which started off with the first one, Bitcoin. The open source software responsible for the creation of this first cryptocurrency, is nowadays called Bitcoin Core and has generated the Genesis Block¹ and therefore the first 50 Bitcoins on the 3rd of January 2009.

Blockchain data structures had already been used before, for example in Git, the widely used version control system. The real innovation was the Proof of Work (PoW) mining/consensus algorithm, which was invented by Nakamoto and ensures that the agreed upon state of the data is backed by at least 51% of the mining nodes. It is used for the creation of new blocks for the blockchain and determines which blocks get accepted.

¹<http://www.buildingsmart-tech.org/specifications/ifc-overview>

²<https://simlab.tuwien.ac.at/simultan/>

These blockchains are presented with the problem that, in theory, everyone can join a network and launch multiple nodes on the network. Therefore, a one vote per node system would not work as a consensus finding mechanism in public blockchains with a permission-less system. The idea behind the algorithm is that nodes need to spend something upfront, so that it is in their interest not to corrupt the network. PoW lets the mining nodes solve a computationally intensive puzzle, which can be easily verified once solved. This computation is costly in terms of energy, but the miners get rewarded if their solution gets accepted into the blockchain.

2.2.1 Decentralization

For our use case, it is important to consider the feature of decentralization not only when it comes to the storage of data but also to voting on changes. A centralized system is one where the control of the data lies in the hands of a single entity. If this entity turns malicious, not only the data integrity but also the process built on top of the data get compromised. In a decentralized system, multiple entities share the power to control data. In public blockchain systems those are the mining nodes. Centralized versus decentralized systems can be compared to autocratic versus democratic political systems. In a centralized system there are only few entities who are in control of the data, whereas decentralized systems use a consensus algorithm (comparable to voting in a democracy) to agree on a valid state of the data.

The advantages of decentralized systems are:

- No single point of failure
- Power over the state is distributed
- Prevents collusions

Since the data storage on a decentralized system is distributed over all nodes in a network, there does not exist one single point of failure. If a node does not work correctly anymore the other nodes can maintain the network and the availability of the data. This is a big advantage in comparison to a centralized database, where, in the worst case only one node is holding the data. If this node gets corrupted the data is compromised without any possibility for recovery.

The second advantage is the distribution of power in a perfectly centralized system. One entity can not arbitrarily change the state of the data, since consensus always needs to be found before a state change gets accepted. In addition, the more spread out the entities which control the data are the more difficult it would be for them to get together and collude against other participants in the network. In a centralized system, it is easier to collude with the controlling entity because, by definition, you have to corrupt less entities to get control over the data.

Public blockchains, such as Bitcoin, are decentralized systems where the mining nodes are spread throughout the world, but there are also different solutions for enterprises, which provide consortium and private blockchains. They provide a system where mining nodes need a permission for participation and for the ability to create new blocks, which restricts the number of participants. This configuration can be seen as a database with multiple administrators where each change has to be accepted by a certain percentage of the administrators. This is a more centralized version compared to public blockchains, which means that collusions between nodes become easier to realize. For example, in a construction project, the construction manager could allow each stakeholder to have one permissioned node while also keeping one herself. If the number of stakeholders in the project is low, it is relatively easy for them to work together and make changes to the database, which the other stakeholders do not agree too. Using a public blockchain, on the other hand, provides a tamper-proof means by which data can be stored. It also ensures immutability. The stakeholders will not be able to tamper with data that has already been included and confirmed on the blockchain. For this reason we focus our research on public blockchains only. A comparison to consortium and private blockchains is out of the scope of this work.

The disadvantage that is inherent in a decentralized system is the performance. When multiple entities need to verify a state the system becomes slower in comparison to a single entity which verifies data in a centralized setup. This is a trade off which prevents blockchains from scaling to the performance of centralized systems. There are already many different researcher groups working to solve this issue [Kar16][KKC18][CMVM18].

In chapter 5 and in the following chapters we will not only evaluate and discuss our prototype's performance but also its political aspects. We also describe possible scenarios of collusion.

2.2.2 Ethereum

Bitcoin was designed not only for simple transactions from one account to another, but also includes a simple scripting functionality, which was inspired by the programming language Forth and is simply called "Script"³. It is stack-based, has support for cryptography, but it does not support loops and, therefore, it cannot be considered a Turing-complete programming language.

In 2013, Vitalik Buterin, an active member of the Bitcoin community, described his research concerning a smart contract architecture and the Ethereum protocol [But13]. Later on, he started to work together with Gavin Wood, who developed the concept of the Ethereum Virtual Machine (EVM) on the Ethereum platform [Woo14]. This platform supports smart contracts and allows the creation of decentralized applications (DApps), which can be seen as an alternative architecture to the common client-server architecture, which is used by current web applications. Traditionally, local apps communicate with a centralized server to handle data and, consequently, this server has to be trusted.

³<https://en.bitcoin.it/wiki/Script>

In decentralized apps, the actual smart contract code runs on every single node that is connected to the Ethereum platform. Theoretically, everyone with a computer can participate in this network and contribute to the decentralized structure by running a mining node. Through this smart contract platform, a multitude of applications can be developed and run on the blockchain. Examples for this could be token systems, financial derivatives, identity and reputation systems, decentralized file storage, decentralized autonomous organizations and more [But13].

2.2.3 Smart Contracts

As already described, the Ethereum ecosystem provides a pseudo Turing-complete smart contract environment. This is one of the big features which separate it from Bitcoin, the pioneer in the blockchain domain. Another difference is the account system, in comparison to Bitcoins UTXO (Unspent Transaction Output) system, particularly when it comes to state and transactions.

On the Ethereum network, users can have access to accounts that can hold funds of ether, which is the main currency used on the Ethereum blockchain. Every valid public-private key pair can be seen as an account. The the public key is used like an address, where the funds are stored, and the private key as the PIN code to unlock the funds in this account. Users can create a private key without having access to the Ethereum network, or even a connection to the internet, by means of cryptography. Having only the private key, it is always possible to get the public key of the account, but not vice versa. In other words, the private key a user generates and plans to use is her identity. If she loses this key, or the key gets compromised, the whole account and her identity is lost or compromised.

Furthermore, on Ethereum there are two types of accounts ⁴:

1. Externally owned account, which:

- has an ether balance
- can send transactions (ether transfer or trigger contract code)
- is controlled by private keys
- has no associated code

2. Contract account, which:

- has an ether balance
- has associated code
- code execution is triggered by transactions or messages (comparable to function calls) received from other contracts
- can perform operations of arbitrary complexity

⁴<http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html>

- can manipulate its own persistent storage, i.e., can have its own permanent state
- can call other contracts through transactions or messages

Smart contracts are, at the moment, mainly written in Solidity⁵, a language similar to JavaScript, which compiles down to EVM bytecode. EVM is the Ethereum Virtual Machine, which runs on every full Ethereum node and executes the bytecode. There are also other high level programming languages available, e.g., Vyper⁶ or Bamboo⁷, which also compile down to EVM bytecode, but are not as commonly used right now, which is the reason why we have implemented our prototype using Solidity.

A mining node is an Ethereum client which downloads and verifies the blockchain and mines (creates, verifies and appends) new blocks. Examples for Ethereum clients are geth⁸ or parity⁹. These nodes mine new blocks because they are rewarded for doing so with the transaction fees and a mining reward, which are both included into the mined blocks.

When a developer wants to deploy a smart contract the compiled code has to be put into a transaction and sent to the Ethereum network. When the block, in which the transaction has been included, gets added to the blockchain, the smart contract is live, which means that it is waiting at an address for users to start interacting with it by sending transactions or messages to its address. When a function gets called, this function will run on every mining node on the network, which means that every mining node verifies if the function call is included into the block correctly, and if the state change that it induces is valid. On the one hand, this creates the desirable decentralized security, which blockchain technologies are known and used for, but, on the other hand, this also produces a huge overhead which results in a worse performance compared to a traditional client server architecture.

The big advantage of a smart contract on a blockchain is the property of immutability. Once it is deployed to a certain address on the network, it stays there and cannot be changed. A normal computer program on a server can be changed by whoever has administrative access to the server. In contrast, smart contracts provide a secure way to make logic available that cannot be altered once it is set up. There are some ways to deploy bug fixes, which we will discuss in chapter 5, but every change is always public and traceable.

2.2.4 The Concept of Gas

To prevent system abuse by a malicious actor who wants to run a Denial of Service attack on the network through spamming the network with a multitude of transactions

⁵<https://Github.com/ethereum/solidity>

⁶<https://Github.com/ethereum/vyper>

⁷<https://Github.com/pirapira/bamboo>

⁸<https://Github.com/ethereum/go-ethereum>

⁹<https://Github.com/paritytech/parity-ethereum>

the concept of gas has been developed and implemented ¹⁰. Standard transactions on Ethereum are used to send ether from one account to another. A transaction costs 21000 gas and the user has to set a gas price in ether for it. Similar to this concept, every EVM instruction, which can be used in smart contracts, has its gas cost. The exact values can be found in the Ethereum Yellow Paper [Woo14]. Additionally, every block that gets mined has a gas limit, which restricts the total amount of gas that can be included into one block, and, therefore sets a limit on the amount of transactions in one block. This limit is currently set to 8000000 ¹¹. If a block is successfully mined, all the gas included by the transactions in it goes to the miner who has mined it. The concept of gas prevents the halting-problem, which normally goes along with Turing-completeness [Wan17]. Through the system of gas the EVM bytecode can actually be defined as "pseudo Turing-complete", because it can not fully simulate a Turing machine. It will stop a function call as soon as it runs out of gas. A transaction can not even be included into a block, if the gas limit for a transaction is set to high. For this reason, an endlessly running loop is not possible on the EVM.

If a user wants to run a function on a smart contract, he has to know the address where the contract was deployed and determine a gas limit (maximum amount of gas spent on this transaction) and a gas price (conversion rate to ether). In the next step, the user signs his transaction and sends it to the Ethereum network, where it goes into the mempool. The miners now collect the transactions with the biggest reward in gas attached to them and put them into a new block, which they start to mine. This means the higher the gas fee on a transaction the quicker it gets included into a block. If a block has been successfully mined, the gas goes to the miner. Since it is not always obvious how much gas a function call is going to consume, it can happen that the user who sent the transaction has set the gas limit too low and the function is not able to finish all included instructions. In this case, the state of the function is completely rolled back to the beginning state, but the miner gets to keep the gas. Therefore, it is very important to, on the one hand, develop the smart contract to consume as little gas as possible and, on the other hand, to set the gas limits as high enough for the function call to complete. If a user has set the gas limit too high, the leftover gas is just returned to his account. ETH Gas Station is a service to calculate a reasonable gas price before one sends a transaction. It allows users to select an average confirmation time and get the approximate gas price that has to be set in order to get the transaction included into a block in this time. Furthermore, a user needs to know how much gas a function on a smart contract consumes. However, depending on the application, the logic of the gas estimation is often automated, so that he does not need to concern himself with this anymore.

Gas costs also play an important role for the evaluation in chapter 5, where we take a closer look at the gas costs of our prototype.

¹⁰<https://Github.com/ethereum/wiki/wiki/Design-Rationale#gas-and-fees>

¹¹<https://medium.com/@piyopiyo/how-to-get-ethereum-block-gas-limit-eba2c8f32ce>

2.3 A Combination of a smart contract and BIM

Nowadays, when BIM models are used in a construction project the approval of a plan is often handled using paper. A construction manager sends the plan to the responsible stakeholder, who prints it in order to sign and approve it, and then sends it back. This workflow is not only slow, but also costly. For this reason we want to combine the BIM process with the smart contract technology of the Ethereum platform. Every stakeholder has access to the whole BIM model to view and edit at all times, and will be able to give her approval to any changes through the on-chain change management process with less delay and less cost.

In this work we present a prototype, the ChangeManager, which uses a smart contract on Ethereum and a Git repository as the basis for a change management system. The smart contract, called ChangeManagerContract, ensures the traceability and immutability of the stakeholders approval, and can be used to hold them accountable for their decision to approve or reject a change. The Git repository is used for storing the BIM files and as a system enabling their controlled change through new commits. The exact functionality and the reasons for the choice of this hybrid approach , consisting of a smart contract and an off-chain Git repository, is explained further in chapter 4.

State of the Art

Blockchain technology and, especially smart contracts, is a relatively new area of research and, therefore, papers regarding this topic are still quite rare, compared to other topics. In addition, in the domain of blockchains, there is a strong bias towards Bitcoin. There was a systematic study in 2016, which showed that out of 41 primary papers from a scientific database 80% focused on the Bitcoin blockchain and only 20% on other blockchain applications [YHKC⁺16].

3.1 BIM

On the topic of the combination with BIM, we found some informal blog articles^{1,2}, which give a broad outlook on the possibility of using smart contracts for construction projects. They describe a visions of securely storing contract documents in a construction project on the blockchain to ensure transparency, accountability and traceability, and to avoid traditional contractual conflicts.

The french startup Bimchain.io announced on their website³ that they are currently in proof of concept trials for a process, which also combines 3D digital modelling and the formal and legally binding paper-based process [Cou18]. They are currently running a beta program and are planning on releasing a software product based on the blockchain and BIM models in 2019. This should include features such as a digital history of commitments, an electronic signature system and an automatic payment process. According to their homepage, they have also partnered with Autodesk and will be providing plugins for various software products, which makes those useable with the

¹<https://constructible.trimble.com/construction-industry/from-bim-to-blockchain-in-construction-what-you-need-to-know>

²<https://www.bim-world.de/de/bim-blockchain-part-2-blockchain-bim/>

³<https://bimchain.io/>

Bimchain.io system. In their software architecture the actual BIM model data is stored in their BIM cloud and only proofs or hashes of the models are stored in the blockchain. This is a similar architecture to the one we use, but we also outsource the storage of the data to a decentralized system, namely Git. However, in general, there is no concrete public research at the crossroad between these two technologies and for this type of use case. Therefore, we present the first prototype as proof of concept, and a comparison between this prototype and two alternative approaches in our evaluation in chapter 5.

Bowe et al. present an outlook on the organizational limitations in the Architecture, Engineering, Building Owner and Operations (AECOO) industry [BRM17]. In addition, they propose various technologies, such as AI, Machine Learning and blockchain, to eliminate these. They give an overview of a possible BIM and blockchain system, which can incentivize various stakeholders through electronic tokens to work on the BIM database, without the need for a 3rd party intermediary. This system should eventually replace the traditional hierarchy with a network structure, because it is more efficient, enjoys a higher market valuation and is more fault tolerant and self regulating. This paper gives a theoretical outlook on a future application of blockchain technology in combination with BIM, but lacks a technical study and evaluation which we present in this work.

Kvasina presents the documentation process of a tunnel construction project [Kva18]. She conducts a case study of the current state through a qualitative observation analysis. Building on the results of the observation she creates a model of the analog documentation process for the project and, finally, showcases an alternative, digital process. Kvasina concludes that the current paper-based workflow that uses non-machine-readable documents, proves to be more error-prone and that the conventional documentation process has the largest digitalization potential.

We, on the other hand, do not cover the various different types of construction documents, which vary from project to project, but focus on BIM models that become more and more common in all types construction projects. However, the non-BIM documentation, mentioned in the referenced work, could also be documented and stored by means of our prototype.

3.2 Blockchain

Papi et al. present a way to ensure accountability for Multi Agent Systems (MAS) [PHdB17]. They introduce various approaches and combine blockchains and MAS by providing different abstractions, followed by a discussion of their advantages and disadvantages. They show a solution where only the state is stored on the Bitcoin blockchain, and another, for which they implement a smart contract on the Ethereum network. They come to the conclusion that it is cheaper to only use a blockchain without smart contract capabilities for communication between MAS, but argue that this would not suffice for the accountability property in their system. In contrast, blockchains, such as the Ethereum, provide a safe logic of authorization and incentivization, which can be

used for MAS. The authors also provide a smart contract as a prototype for an auction of house building tasks. They also name the trade off in speed and the challenges with the scalability of blockchain systems as the biggest issues.

For our prototype we also create a smart contract prototype, but specifically for the use case of a construction project. In addition, in chapter 5 we will evaluate the prototype using the "Goal, Question, Metric" framework [CR94].

Further, Neisse et al. describe the use of blockchain technology for the storage of consent to data access, as accountability mechanisms and for data provenance tracking. They embed their research into the context of the European General Data Protection Regulation⁴, which has the aim to protect EU citizens from data breaches and has been applicable in the European Union since May 25th, 2018. They discuss several solution design choices, describe three different models and have implemented two of them. Their analysis of the prototype focuses on data accountability and provenance tracking and, therefore, on granularity, privacy, anonymity, performance and scalability.

We, on the other hand, evaluate our prototype in the context of a construction project, where we focus on performance, scalability, security, costs and, especially, on viability for our use case.

Wenisch presents research in the area of law regarding smart contracts [Wen17]. He examines the validity of smart contracts in contemporary contract law in Austria and in the United States, and concludes that there are some major differences between the contract law systems of these two countries. According to Wenisch, smart contracts in Austria might not be valid before a court with the applicable jurisdiction, because contracts have to be written in understandable language. Therefore, smart contracts cannot be seen as traditional contracts.

Future work could be done in examining the legal aspects of our solution, since our smart contract works as a change management system rather than a traditional contract. It provides accountability features using digital signatures. The question if this would suffice in front of a court could be further researched.

Falazi et al. present a prototype for collaborative development of application deployment models [FBF⁺18]. In addition, their focus is on the accountability property, for which they use the Ethereum network. Their prototype allows them to ensure integrity and provenance for a collaborative development workflow on cloud based systems. They have a similar architecture to the one we are using, which is also divided into two parts. One part, which is used for accountability measures, exists in a smart contract on the blockchain. The other part is a decentralized file storage system. They refrain from storing data on Git because they argue that the Git protocol is not enough to support the desired accountability property since the history in a Git repository can be changed. Instead, they use swarm⁵, a distributed storage network.

We, on the other hand, postulate that Git with its decentralized properties, combined with the hash tree architecture, is sufficient to provide the accountability for our use

⁴<https://gdpr-info.eu/>

⁵<https://swarm-guide.readthedocs.io/en/latest/introduction.html>

case. A further explanation of the Git protocol and our implementation presented in section 4.5. There we also show how the Git tree commit structure produces properties that make malicious changes on a centralized repository visible to all other users.

Furthermore, there are two projects in development, which should bring a more decentralized approach to the hosting of Git repositories. Gitchain⁶ is a project started by Yurii Rashkovskii which uses Bitcoin, Namecoin and distributed hash tables. It provides properties, such as decentralized redundant storage, encrypted storage for private storage, tamper-proof history and Proof of Work, contribution and storage rewards.

The second project is GitTorrent⁷, which was initiated by Chris Ball. It handles a similar use case, but uses a peer-to-peer network, based on a BitTorrent protocol extension and distributed hash tables.

Both projects currently are not actively maintained anymore (Latest commit from both Github projects: 3 years ago).

Projects like these, or the beforementioned swarm, could be used for a more decentralized hosting instead of the centralized providers like the Github platform.

Bhargavan et al. present a short paper on the topic of formal verification of smart contracts [BDLF⁺16]. They use F* based on shallow embeddings and typechecking within an existing verification framework to verify example smart contracts. They plan to complete a verified reference implementation of the Solidity compiler to verify that its output is functionally equivalent to the source contracts.

A formal verification of our prototype could be an interesting topic for further research in terms of security. Since we only implement an explorative prototype we will not specifically focus on security during the implementation. However, as part of chapter 5, we evaluate the security of the Ethereum platform in general.

Concerning the security of smart contract development, Whorer and Zdun present six design patterns for Solidity which can be used to mitigate typical attack vectors [WZ18]. In future work, they plan to extend their collection of design patterns to a structured and informative design pattern language for Solidity, which will provide guidance to smart contract developers. Related to this, Mense and Flatscher present common vulnerabilities found in smart contracts and sort the vulnerabilities into three different categories: Solidity, EVM bytecode and general blockchain characteristics [MF18]. This taxonomy describes the vulnerabilities on different levels of the application. They stress that there is room for improvement through further taxonomization, automated test environments and closing the research gap in order to mitigate vulnerabilities. The considerations mentioned in these two papers are also discussed in our evaluation chapter.

Martinez et al. describe a hashing mechanism, which can be used to protect valuable assets in complex industrial environments in the context of model-driven engineering [MGC18]. They have explored robust hashing techniques, which have already proven to be valuable for the protection of intellectual property, authenticity assessment and fast

⁶<https://github.com/gitchain/gitchain>

⁷<https://github.com/cjb/GitTorrent>

comparison and retrieval solutions in various domains. The authors provide a prototype implementation, which uses their hashing technique, and an experimental evaluation. They too describe a blockchain infrastructure as a possible way of storing the hashes of the models to ensure additional accountability.

Although robust hashing techniques do not apply to our use case, because we do not need to compare similar models, the approach of storing hashes on the blockchain is also used by our prototype.

In the next chapter we first describe the use case that is the basis of our prototype, present the extracted requirements and then show how the prototype was implemented using different technologies. We also give insights into the inner workings of the underlying Git technology and the Ethereum smart contract platform .

Realization

In this chapter we describe the requirements for and the architecture behind our Change-Manager prototype and we discuss the implementation. In the following we first specify the requirements, which resulted from the analysis of our previously described use case, then we outline the functionality of the change management process and, subsequently present a detailed description of the single components.

4.1 Requirements

The prototype should provide a change management workflow for data which represent a construction project. In such a project, there should only be one shared digital representation of this data as a single point of truth, to which everyone can refer. This helps to prevent confusion and subsequent errors, which occur when stakeholders are working on different versions of the model at the same time. The data will be stored as a BIM model to provide the properties which are needed for a construction project. As we have already seen, a project in this domain can have many different stakeholders, such as a construction manager, an architect, contractors, regulators, etc. In the following we are going to describe the change management workflow from a high-level point of view with a focus on the requirements of this process.

We have one party, the construction manager, who is responsible for the communication within the project. In our workflow, any stakeholder who has access to the BIM model can commit a change to the model parts they are liable for. The construction manager has to make sure that changes on the BIM model are approved by the authorised people. Therefore, she is the first person who needs to accept a change requested by any stakeholder. Furthermore, for every change there are multiple additional parties who have to consent to a change so that it can be approved. This could, for example, be the building developer, who has to check if the change affects the budget or interferes with other requirements. The construction manager is also responsible for contacting the

responsible parties to set up a vote on a proposed change. This whole process has to be transparent, traceable and it has to be possible to hold the stakeholders accountable for their approval. The goal of the ChangeManager project is to create a solution in which stakeholders can commit change requests to a BIM model and vote on these change requests, if they have the permission to do so. The process is managed by the construction manager.

In regards to the data resulting data from this process, there are two kinds which have a different set of requirements.

The first issue is the storage of the comprising from the BIM model. This data can become really big, and incremental changes on the model have to be possible for multiple stakeholders at the same time. Therefore, we are use the approach of storing the data in a Git repository for our prototype. The data also has to be stored in a forgery-proof way. A Git repository, is by definition, decentralized since every user has a copy of the whole repository on his computer. Git also provides an integrity system using SHA-1 hashes for every commit. In our architecture, the Git hashes are the indexes which point to exactly one commit of a change request on a BIM model. We describe this feature in more detail in section 4.5.

The second issue is the storage of the votes on committed change requests. The votes will only be conducted once a change commit, which has to be approved by other stakeholders, has been made to the BIM model. The data does not contain a lot of information, so it is generally small in size. However, once a vote is cast, it has to be immutable and its history has to be traceable. Therefore, this part of the process is implemented as a smart contract on the Ethereum network. The smart contract handles the permission management and the change approval process. We describe this in more detail in section 4.6

4.2 Operating Principles of the Prototype

After a change request has been committed to the Git repository, anyone can create a new ChangeRequest. A ChangeRequest is the representation of a Git commit in the ChangeManagerContract smart contract. As a first step, the construction manager can accept or reject a ChangeRequest. If she accepts it, she can assign multiple responsible parties to vote on the ChangeRequest. If all of them accept it, the ChangeRequest goes into the state accepted and is regarded as approved. If one responsible party rejects it, the ChangeRequest has been rejected and cannot be approved without creating a new Git commit. In this state, the ChangeRequest cannot be voted on again.

To include each vote into the Ethereum network the user puts the vote into a transaction to write onto the immutable ledger. The transaction has to be signed with a private key and, when it gets included into a block, the address and its vote are logged in the network. Therefore, if a stakeholder has accepted or rejected a change on the BIM model this vote on the blockchain can be used to hold the voter accountable. It is important that every stakeholder who participates in this process hands his public key to all other

stakeholders, so that everyone can map their digital identity to a real identity. This could be handled using a traditional legal contract.

Change Management Process

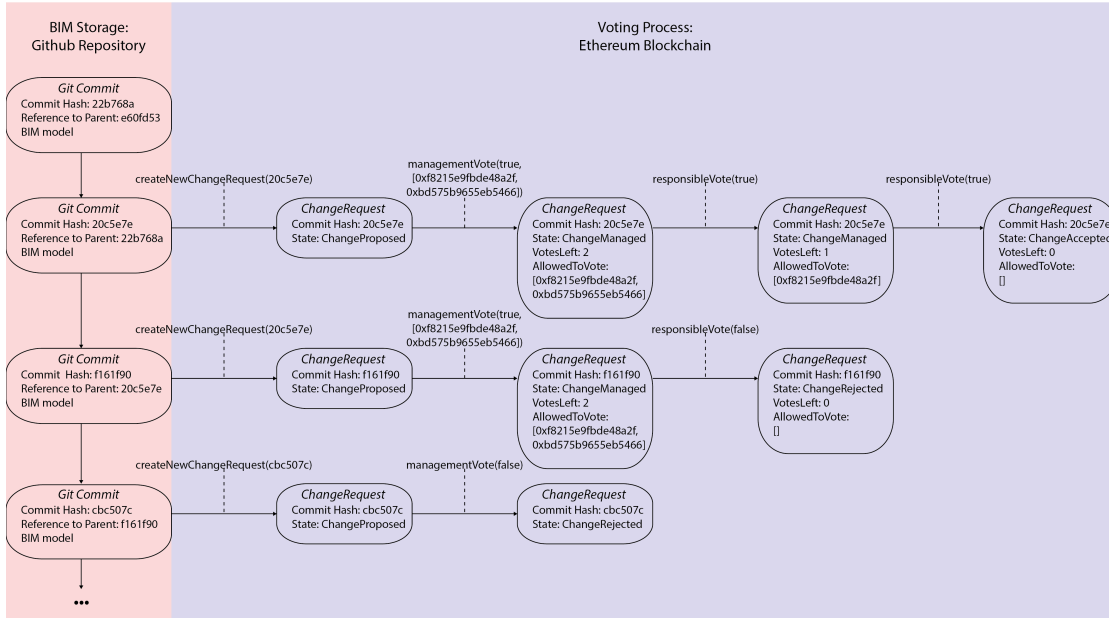


Figure 4.1: Parallel processes of committing a change on Git in red and handling a ChangeRequest on the Ethereum network in purple.

In the following we present the architecture and functionality of the ChangeManager prototype. Afterwards, we provide a detailed breakdown of its single components.

4.3 Architecture

The ChangeManager prototype is composed of components which use the following technologies:

- Github Repository
- Ethereum smart contract (ChangeManagerContract)
- Windows Presentation Foundation C# Frontend (ChangeManagerWPF)

The Github repository is used to store the BIM model, the ChangeManagerContract (ChangeManagerContract) represents the voting workflow on the blockchain and the ChangeManagerWPF (ChangeManagerWPF) provides the user with a graphical user interface which is used to communicate with the ChangeManagerContract.

These technologies were chosen by us because they are common and, therefore, enough documentation is provided by the community developing them. Github is the biggest Git repository hosting service with more than 38 million projects[flo16]. The Ethereum environment not only provides the third biggest cryptocurrency (ether) by market cap¹, but also the biggest developer space for smart contracts.

We chose C# and WPF for our frontend because of the SIMULTAN project's software tool, which was created in cooperation with multiple research partners by TU Wien [BBF⁺18]. This project is also developed with C# and WPF and could potentially be extended by our prototype to provide a more integrated BIM model creation and change management process.

Each of the used technologies can be replaced by another, depending on the needs of the BIM process. For example, the Git repository does not have to be hosted on Github. A solution like GitTorrent or Gitchain would actually be more suited for improving the decentralization and, therefore, ensure the independence from central entities.

In the following we provide further insights into the underlying technologies.

4.4 SIMULTAN Software Tool

The SIMULTAN software tool has a focus on the simultaneous technical presentation and design of BIM models. It provides a digital platform for a construction project, which is being developed by multiple stakeholders with diverging requirements and interests. It provides not only a means for management and integration of their different viewpoints of the BIM model, but also a tool to interact with it. It fulfills the following requirements [BBF⁺18]:

- Data created by different stakeholders must not produce an inconsistent state or redundancies.
- The progress of the project must be visible and traceable at any point in time.
- The data model must be accessible for any other application with reasonable effort.
- The resulting data model should be adequately detailed and be adjustable at any time, and act as a digital twin to the project.

The SIMULTAN tool delivers collaboration in real-time, filter functionality according to access, category and custom criteria, and works with a Git repository. Its data model consists only of very few generic elements and can manage varying levels of detail. It provides an access and permission control for each role occupied by a stakeholder.

¹<https://coinmarketcap.com/>

The SIMULTAN tool can be used to create, edit and manage a single BIM model for multiple users at the same time. It also includes a change management process, where stakeholders can vote on changes made by other stakeholders. However, it handles this process on a centralized infrastructure, which makes it vulnerable to a malicious actors who might attempt to forge voting results. The ChangeManager prototype extends the tool by transferring the change management workflow to a decentralized and immutable datastructure, the blockchain.

The tool is written in C# and uses the Windows Presentation Foundation (WPF) libraries to provide a graphical user interface. Our prototype uses the same underlying technology to facilitate the integration into the SIMULTAN tool. A proof of concept integration of the ChangeManager into the SIMULTAN tool has already been implemented. However, the evaluation of our prototype as an extension to the SIMULTAN tool is out of our scope and we evaluate only the ChangeManager in this work. The evaluation of the integration of both tools could be subject to further research in this area, which we describe in chapter 6.

4.5 Git

Git [CS14] is a decentralized VCS (Version Control System). The technology was first initiated by Linus Torvalds in 2005 to serve as a source code management software for the Linux kernel. The system is organized in repositories, which do not need a central server. Every user has a local copy of the whole repository on her computer. All local repositories can talk to each other and they do not differ from a repository stored on a central server. Furthermore, the user can work on his local repository without having an active connection to other repositories. However, usually, a project has one repository, which resides on a central server, onto which the users push their changes to have a central point of storage. For our prototype we have chosen Github² as a hosting platform for our central repository, because Github provides an API, which lets us read and collect the changes easily.

In Git, changes on data are packaged into commits. These represent a change in files on the repository and can always be identified by a unique SHA-1 hash. This hash includes not only the current changes, but also a hash of the parent commit. The reference to the parent commit produces a tree structure, known as a Merkle tree, and represents the Git history. This is, in essence, the same system used in a blockchain and makes the files in a Git repository tamper-proof as long as one has a local copy of the repository. If a malicious actor tries to change one commit in the Git history, every commit and hashes which were created after the tampered-with commit would change as well. This means that if one has a commit hash and you have the same hash on a central server one can be sure that it contains the exact same files in the exact same version. This mechanism ensures that changes on the BIM model can be securely stored on Git, and we can just store the Git commit hash as a reference into the blockchain. This is enough information

²<https://Github.com/>

to prove that a commit hash of a change describes a certain state of a repository. Since every stakeholder can and should have a local copy of the repository on their computer, they can prove that a `ChangeRequest` in the smart contract represents a certain state of the BIM model.

Our prototype does not include the functionality of creating new commits. It only reads the commits from a Git repository on Github. The feature of adding new commits is already provided by the `SIMULTAN` tool, which can be used as a BIM model collaboration tool, and uses Github in the background. We will still take this functionality into consideration in our evaluation in the context of the different solutions.

The `ChangeManagerWPF` in our prototype only fetches all commits from a Github repository and presents them to the user, who is able to select one and create a new `ChangeRequest` on the `ChangeManagerContract`. In the next section we describe the functionality of the `ChangeManagerContract`.

4.6 Smart Contract

4.6.1 Environment

We started the development of our prototype with the implementation of the `ChangeManagerContract`. We used a test-driven software development approach for its implementation. The idea behind this strategy is that the test cases are written before the prototype so that the use cases are already defined programmatically in code. The tests are implemented so that they fail at first; they pass once the actual prototype implementation is complete [MW03]. This ensures that the final prototype fulfills the requirements of the use case. Therefore, as a first step, we defined our use case and, only then, began to implement tests according to this use case. We used the tests as a basis for the specification of the following implementation for the `ChangeManagerContract`. To develop it we used `Truffle`³, which comes with a Solidity compiler and a testsuite. To test our smartcontract without the need of spending gas we used `Ganache`⁴ to run a local simulation of the Ethereum network. It provides not only a simulation of the Ethereum network and an API to communicate to it, but also creates 10 accounts with public and private keys, which can be used for testing purposes.

³<https://truffleframework.com/>

⁴<https://truffleframework.com/ganache>

We use two models in the ChangeManagerContract: ChangeRequest and Vote.

A ChangeRequest represents a Git commit and is uniquely identified by its Git hash. The ChangeRequest additionally contains costs, time estimation and additional information, and it is always in one of these states:

- ChangeProposed
- ChangeManaged
- ChangeRejected
- ChangeAccepted

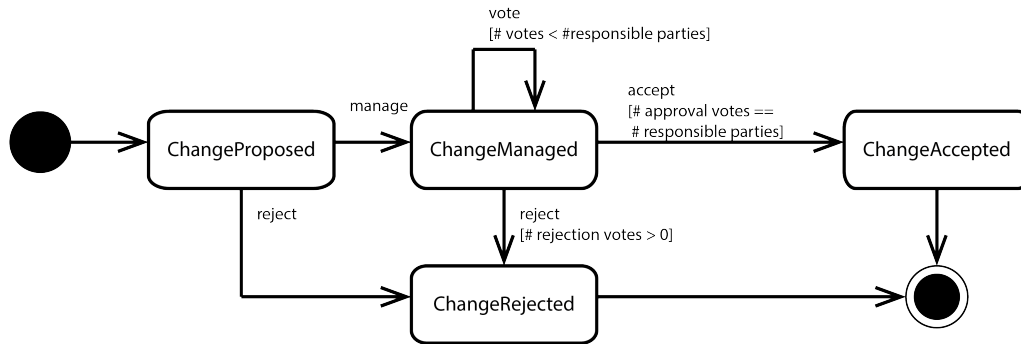


Figure 4.2: Possible states of a ChangeRequest and the transitions between them including the conditions

A Vote represents either the acceptance or the rejection for a ChangeRequest by a stakeholder. It contains the address of the voter, the Git hash of the ChangeRequest, the acceptance or rejection Vote and vote information. Every newly created ChangeRequest and every casted Vote get propagated as events on the blockchain, which can be read by everyone who runs a blockchain node. This feature works as an immutable persistent logging system on the Ethereum network.

The ChangeManagerContract implementation is developed using Solidity 0.4.23. It is neither optimized for performance, nor cost-efficiency, nor security. On the one hand, it is used as a proof of concept, and on the other hand, it serves as a first reference point for our evaluation.

The code logic is organized into two files (ChangeManager.sol and ChangeTracker.sol) that contain a data structure similar to classes in other languages, which are named contracts in Solidity. We call them classes in this work in order to avoid confusion. We have reserved the name contract for the logic on the blockchain as a whole.

ChangeTracker is the parent class of the ChangeManager. It contains the state entities State, Change, NewChangeRequest and NewVote. State and Change are of type enum and struct, respectively. These are data structures used to store information. ChangeRequest contains the Git hash, has an owner ($\hat{=}$ creator), a state ($\hat{=}$ one of the before-mentioned states), a mapping ($\hat{=}$ key-value data structure in Solidity) to store the addresses, which are allowed to vote on the ChangeRequest, and the vote count. Some additional fields are used to specify a ChangeRequest further (costs, estimation, additional information).

NewChangeRequest and NewVote are events, which are called in code, and their signature is written to the blockchain. APIs, which are able to communicate with the Ethereum network, can filter for certain event types on a contract address. We use this feature to track newly created ChangeRequests and the votes in our frontend application.

In the ChangeManager class we provide the logic for our smart contract. It has a constructor, which gets called when the smart contract is deployed to the blockchain. This function sets the deployer of the smart contract to be the construction manager. He is identified through his address and has special permissions in the applied logic. There are three functions in the ChangeManager, callable via the user interface:

1. createNewChangeRequest
2. managementVote
3. responsibleVote

In the following we describe the process of creating and voting on a ChangeRequest in chronological order.

First, the `createNewChangeRequest` function, as we show in Listing 4.1, needs to be called. A user passes a Git hash, which references a certain state of a BIM model in a Git repository, and some additional information to the function. The `ChangeManager-Contract` saves this `ChangeRequest` into a mapping, if it does not already exist, and sets its state to `ChangeProposed`. This check is done using the `require()` function, which, if a condition is not met, halts the execution of a function call, reverts its state and hands back the yet unspent gas to the caller. At the end of the function call, the event `NewChangeRequest` is triggered, which can be logged from outside the blockchain.

```

1 // Creates a new ChangeRequest contract and saves it in the _changes array.
2 function createNewChangeRequest (
3     bytes20 gitHash,
4     string additionalInformation,
5     uint256 costs,
6     uint256 estimation
7 )
8 public
9 {
10     require(_changes[gitHash]._gitHash == bytes20(0), "ChangeRequest already
        exists");
11
12     ChangeRequest memory change;
13
14     change._gitHash = gitHash;
15     change._additionalInformation = additionalInformation;
16     change._costs = costs;
17     change._estimation = estimation;
18     change._changeOwner = msg.sender;
19     change._state = State.changeProposed;
20
21     _changes[gitHash] = change;
22
23     // Emit a NewChangeRequest event
24     emit NewChangeRequest(gitHash, additionalInformation, costs, estimation);
25 }

```

Listing 4.1: The solidity code of the function `createNewChangeRequest()`

As the next step in the change management process, the `managementVote` function, as we show in Listing 4.2, is called. First, the construction manager can give his approval of the `ChangeRequest` and contact other responsible parties, who he also needs the approval from. The function uses a Git Hash as identifier for the `ChangeRequest`, a boolean value indicating whether to accept it, an array with addresses, which are allowed to vote in the next step, and additional information. The function can only be run by the the construction manager and the `ChangeRequest` has to already exist on the smart contract and to have the state `ChangeProposed`. The `require()` function is used to ensure these conditions. If the boolean value indicating to accept the change is true, the `ChangeRequest` transitions into the state `ChangeManaged`, otherwise to `ChangeRejected`. Once a `ChangeRequest` is in state `ChangeRejected` it remains in this state and cannot be changed anymore.

```
1 // Function can only be run by the owner of the ChangeManager contract (
  construction manager). The construction manager does the first review of
  the ChangeRequest, can reject it or employ the responsible parties who
  are allowed to vote on the change.
2 function managementVote(
3     bytes20 gitHash,
4     bool acceptChange,
5     address[] responsibleParties,
6     string voteInfo
7 )
8 public
9 {
10     ChangeRequest storage change = _changes[gitHash];
11     require(msg.sender == _constructionManager, "Sender not construction
        manager");
12     require(change._state == State.changeProposed, "State not ChangeProposed"
        );
13
14     if (acceptChange) {
15         change._voteCount = responsibleParties.length;
16         for (uint i = 0; i < responsibleParties.length; i++) {
17             change._allowedToVote[responsibleParties[i]] = true;
18         }
19         change._state = State.changeManaged;
20         emit NewVote(gitHash, msg.sender, acceptChange, change._state,
            voteInfo, change._voteCount);
21     }
22     else {
23         change._state = State.changeRejected;
24         emit NewVote(gitHash, msg.sender, acceptChange, change._state,
            voteInfo, 0);
25     }
26 }
```

Listing 4.2: The solidity code of the function `managementVote()`

There is only ever one voting process possible on a specific Git commit. In case a

stakeholder wants to restart the voting process a new Git commit with a different commit hash has to be created, so that a new `ChangeRequest` can be added in the `ChangeManagerContract`.

After the vote is managed, an event `NewVote` is triggered and logs the `Vote` to the blockchain.

As the final step, `responsibleVote`, as we show in Listing 4.3, is called on a `ChangeRequest`. This function accepts the same input arguments as `managementVote`, except for the array containing addresses. It requires the caller to use one of the addresses assigned to the responsible parties and the `ChangeRequest` to be in the state `ChangeManaged`. If a negative vote is cast the `ChangeRequest` goes into the state `ChangeRejected` and the voting ends. If the vote is positive and there are some responsible parties who have not yet voted, it stays in the state `ChangeManaged`. If the last responsible party casts a positive vote the `ChangeRequest` goes into the state `ChangeAccepted`. This means that the change has been accepted by all parties, who the construction manager has contacted to vote. The change has been approved and can be implemented.

```
1 // The allowed parties can vote. As soon as everyone has voted the
   ChangeRequest is either accepted or rejected
2 function responsibleVote(
3     bytes20 gitHash,
4     bool acceptChange,
5     string voteInfo
6 )
7 public
8 {
9     ChangeRequest storage change = _changes[gitHash];
10    require(change._state == State.changeManaged, "State not ChangeManaged");
11    require(change._allowedToVote[msg.sender], "Sender not allowed to vote");
12    change._allowedToVote[msg.sender] = false;
13    if (!acceptChange) {
14        change._state = State.changeRejected;
15        change._voteCount = 0;
16        emit NewVote(gitHash, msg.sender, acceptChange, change._state,
17                     voteInfo, 0);
18    }
19    else {
20        change._voteCount = change._voteCount - 1;
21        if (change._voteCount == 0) {
22            change._state = State.changeApproved;
23            emit NewVote(gitHash, msg.sender, acceptChange, change._state,
24                         voteInfo, change._voteCount);
25        }
26        else {
27            emit NewVote(gitHash, msg.sender, acceptChange, change._state,
28                         voteInfo, change._voteCount);
29        }
30    }
31 }
```

Listing 4.3: The solidity code of the function `responsibleVote()`

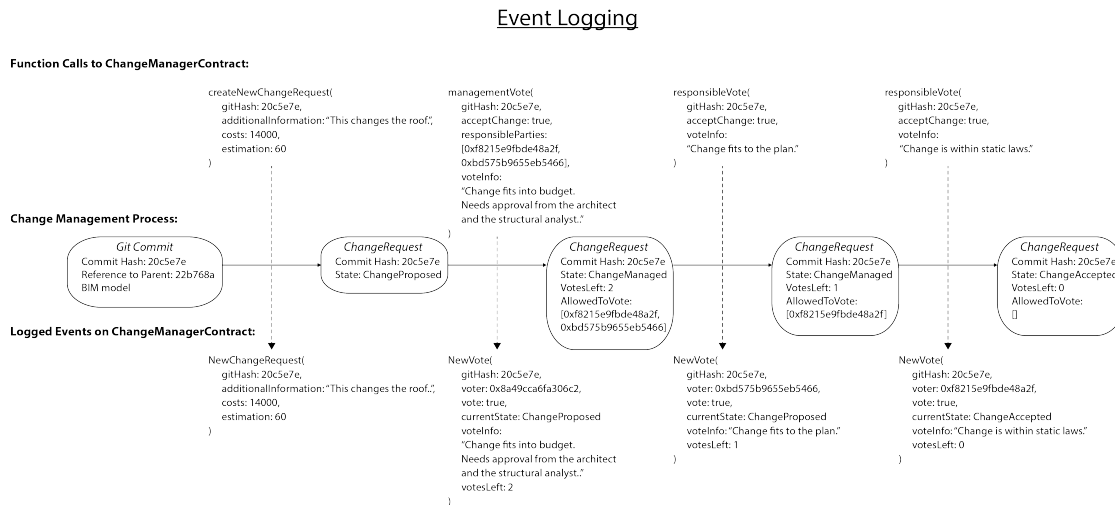


Figure 4.3: Change management process and the different possible states for a ChangeRequest.

4.7 The Frontend

The frontend is implemented in C# and uses the WPF (Windows Presentation Foundation) as a framework. In the ChangeManagerWPF we use the following libraries and APIs:

- Nethereum⁵
- Github API⁶
- Newtonsoft.Json⁷

We use Nethereum to talk to an Ethereum node. It is the .Net integration library for Ethereum. In our application it takes care of deploying the ChangeManagerContract, calling the functions in the ChangeManagerContract and watching for new events.

The Github API lets us query commits on a repository. The repository needs to be public for our prototype, because we have not implemented any way of authenticating to the API. We only use the Git hash of the commits, but more information, such as the commit message, could also be used to display in the ChangeManagerWPF.

Newtonsoft.Json helps us process the responses from the Github API through deserialization of the returned JSON. As a next step we will give an overview on how the frontend is used.

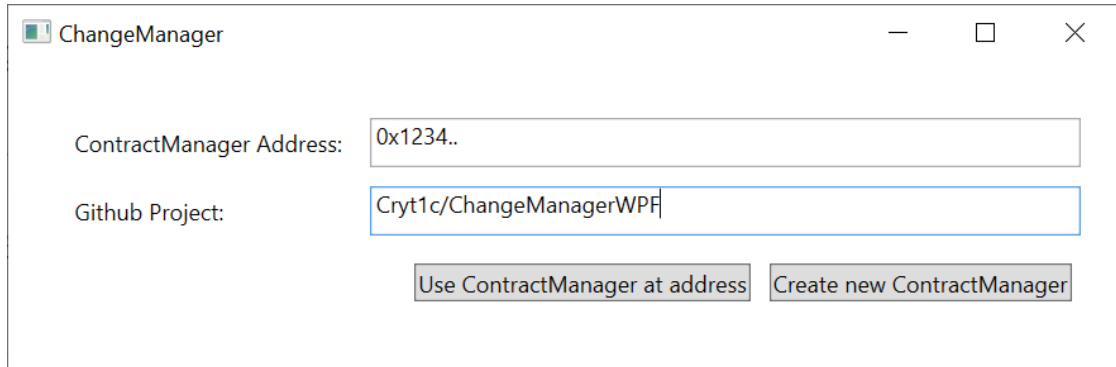
⁵<https://Github.com/Nethereum/Nethereum>

⁶<https://developer.Github.com/v3/>

⁷<https://Github.com/JamesNK/Newtonsoft.Json>

4. REALIZATION

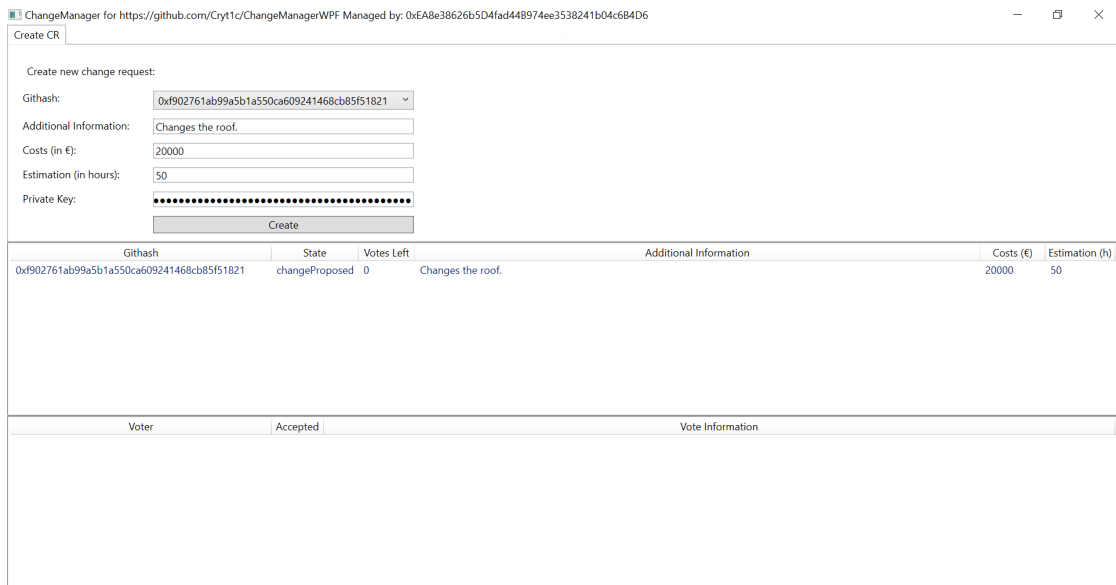
In the first window, when the user starts the frontend he must provide the reference to a public Git repository (Figure 4.4). Either a new `ChangeManagerContract` can be deployed, or an already deployed `ChangeManagerContract` at a certain address can be used.



The screenshot shows a window titled "ChangeManager". It contains two input fields: "ContractManager Address:" with the value "0x1234.." and "Github Project:" with the value "Cryt1c/ChangeManagerWPF". Below these fields are two buttons: "Use ContractManager at address" and "Create new ContractManager".

Figure 4.4: Creation or Reuse of a `ChangeManagerContract`

In the next step, a commit hash representing a change request on Github can be selected and additional information can be filled in (Figure 4.5). Then a new `ChangeRequest` can be created, which sets its state to "changeProposed".



The screenshot shows a window titled "ChangeManager for https://github.com/Cryt1c/ChangeManagerWPF Managed by: 0xEA8e38626b5D4fad448974ee3538241b04c684D6". It has a tab labeled "Create CR". Below the tab is a form titled "Create new change request:" with the following fields:

- Githash: 0xf902761ab99a5b1a550ca609241468cb85f51821
- Additional Information: Changes the roof.
- Costs (in €): 20000
- Estimation (in hours): 50
- Private Key: [Redacted]

Below the form is a "Create" button. Below the button is a table with the following data:

Githash	State	Votes Left	Additional Information	Costs (€)	Estimation (h)
0xf902761ab99a5b1a550ca609241468cb85f51821	changeProposed	0	Changes the roof.	20000	50

Below the table is a section with tabs "Voter" and "Accepted", and a "Vote Information" section.

Figure 4.5: Proposing a new `ChangeRequest`

When the user clicks on a proposed ChangeRequest the Management Vote tab opens (Figure 4.6). The project manager (owner/deployer of the ChangeManagerContract) can fill in vote information and the addresses of the responsible stakeholders who are allowed to vote on the ChangeRequest. If accepted, the ChangeRequest transitions into the "changeManaged" state.

ChangeManager for <https://github.com/Cryt1c/ChangeManagerWPF> Managed by: 0xEA8e38626b5D4fad44B974ee3538241b04c6B4D6

Create CR Management Vote

Manage the selected Change Request: [Link to the Github Commit](#)

Vote: ☒ Accept ☐ Reject

Addresses (0x..., 0x...):

Vote Info:

Private Key:

Manage

Githash	State	Votes Left	Additional Information	Costs (€)	Estimation (h)
0xf902761ab99a5b1a550ca609241468cb85f51821	changeProposed	0	Changes the roof.	20000	50

Voter Accepted Vote Information

Figure 4.6: Managing a ChangeRequest

4. REALIZATION

When the user clicks on a managed ChangeRequest, the Responsible Vote tab opens. The responsible stakeholders can now vote on the ChangeRequest and fill in their vote information (Figure 4.7).

ChangeManager for https://github.com/Crypt1c/ChangeManagerWPF Managed by: 0xEA8e38626b5D4fad44B974ee3538241b04c6B4D6

Create CRResponsible Vote

Vote on the selected Change Request:[Link to the Github Commit](#)

Vote:

☒ Accept ☐ Reject

Vote Info:

The structural calculations are still correct.

Private Key:

.....

Vote

Githash	State	Votes Left	Additional Information	Costs (€)	Estimation (h)
0xf902761ab99a5b1a550ca609241468cb85f51821	changeManaged	2	Changes of the roof.	20000	50

Voter	Accepted	Vote Information
0xea8e38626b5d4fad44b974ee3538241b04c6b4d6	True	Structural engineer and architect have to accept.

Figure 4.7: Conducting a responsible vote on a ChangeRequest

If one stakeholder rejects a ChangeRequest it goes into the state "changeRejected" (Figure 4.8). In this case a new Git commit needs to be created in order to create a new ChangeRequest in the ChangeManagerWPF.

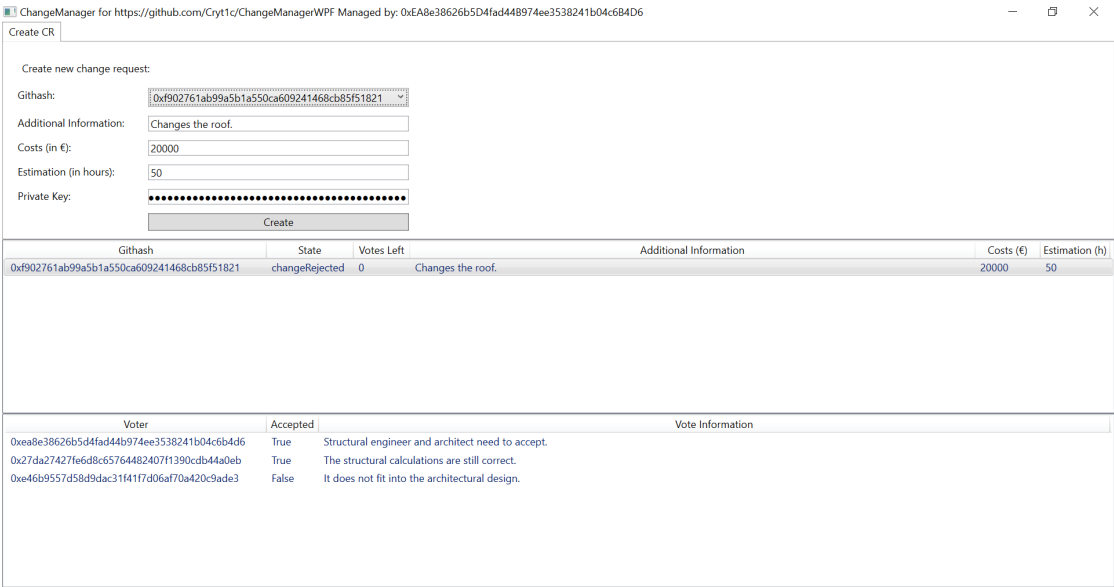


Figure 4.8: Rejected ChangeRequest

In the following chapter we describe our evaluation methodology, which uses the "Goal, Question, Metrics" approach and present the results of applying this method to our ChangeManager prototype.

Evaluation

For our evaluation we have chosen the Goal, Question, Metrics approach which was first described by Caldiera et. al and has become a standard for creating adequate software metrics [CR94]. The result of the approach is the specification of a measurement model on multiple levels:

1. Conceptual level (Goal)
2. Operational level (Question)
3. Quantitative level (Metric)

The whole process provides a hierarchical structure, which can be represented as a tree, with a goal as the root, the questions as nodes and the metrics as leaves. Traversal from the bottom to the top serves to define software metrics. Traversal from the top to the bottom is used to interpret the measurements.

The GQM process consists of the following steps:

1. It starts with the creation of a goal for products, processes or resources by defining the purpose of the measurement, the object and the issue to be measured and the viewpoint, which is taken to measure. ("Which goal should the evaluated object reach?")
2. The next step of the process is the definition of questions to partition the issue into its parts. These questions should always have the goal as a starting point. ("What should be measured and which questions should the measurements answer?")
3. As the last step in the hierarchy, the questions are broken down into objective and subjective metrics. ("Which metrics are able to describe the necessary properties?")

It should be noted that one metric can be used to answer multiple questions and one question can be used to answer multiple goals. After the definition of the model, the researchers apply the measurements. They subsequently interpret the results by using the measurements to answer the questions and evaluate if and how the goal is reached.

In the following section we present the GQM model for our use case and in the summary thereafter discuss three different solutions, for comparative purposes, by presenting their advantages and disadvantages.

5.1 Goal

From our previously defined use case we can derive the context and the subsequent goal:

- Context: The goal is defined for the process of storing BIM models and managing change requests. It is viewed from the point of a manager of a construction project.
- Goal: The goal is to create an economical, fast and secure solution for storing data and managing changes.

Based on this goal we define the questions in the following section.

5.2 Questions and Metrics

To provide a better structure to our evaluation we categorize the questions into four areas of comparison:

1. Economic Questions
2. Technical Questions
3. Political Questions
4. Security Questions

The categories are not mutually exclusive and overlap often. Security in particular is considered a meta topic which spans many others and can be considered part of all other areas.

5.3 Evaluation Plan

In this section we present each category with its questions, followed by the derived metrics. The first question presented is always a broad question which we use to do research in order to raise more questions. These don't provide concrete metrics but are a means to discuss the area in a broader way.

1. Economic Questions

- a) What are possible economic issues?
 - i. General economic concerns
- b) What are the costs for storing big files?
 - i. Costs of data storage on the Ethereum blockchain of 10 gigabytes
 - ii. Costs of data storage in Git repository of 10 gigabytes
- c) What are the costs for the change management process?
 - i. General costs of processes on the Ethereum blockchain
 - ii. Costs of the voting process for the ChangeManager prototype
 - iii. General costs of processes in a Git repository

2. Technical Questions

- a) What are possible technical issues?
 - i. General technical concerns
- b) Does the solution allow the storage of big files?
 - i. Storage limitations on the Ethereum blockchain
 - ii. Storage limitations in a Git repository
- c) How much time does it take to push data onto the storage?
 - i. Time to get data into a block
 - ii. Time to get a data into a Git repository
- d) How much time does it take to cast a vote?
 - i. Time to get a function call into a block
 - ii. Time to create a new Git commit and push it
- e) How does the underlying technology work?
 - i. Scalability issues for Ethereum
 - ii. Scalability issues for Git

3. Political Questions

- a) What are possible political issues?

- i. General political concerns
 - b) How centralized is the solution?
 - i. Data storage location of the Ethereum blockchain
 - ii. Data storage location of a Git repository
 - iii. Conditions to create a collusion to falsify the stored data on the Ethereum blockchain
 - iv. Conditions to create a collusion to falsify the stored data on a Git repository
 - c) Who is in control of the data?
 - i. Conditions to make unauthorized data changes on the Ethereum blockchain
 - ii. Conditions to make unauthorized data changes on a Git repository
 - d) How does the ecosystem around the solution work?
 - i. Development process of Ethereum node software
 - ii. Development process of Git
 - iii. Who is developing the open source software?
 - iv. Number of maintainers of the Ethereum node software and the Git software in comparison
4. Security Questions
- a) What are possible security issues?
 - i. General security concerns
 - b) What are possible attack vectors on the data storage?
 - i. Conditions to bring the system to a standstill
 - c) How private is the data?
 - i. Authorization permissions to read data on the Ethereum blockchain
 - ii. Authorization permissions to read data on a Git repository

This plan was created in an iterative approach and we present the results of the evaluation in the next section.

5.4 Evaluation Results

In this section we will apply the previously described evaluation plan to our prototype and ,afterwards, discuss three different solutions:

1. Code storage and voting using Git (off-chain)
2. Code storage using Git and voting using the Ethereum blockchain (hybrid) - our ChangeManager prototype
3. Code storage and voting using the Ethereum blockchain (on-chain)

Using the evaluation plan we not only evaluate the ChangeManager prototype we presented in chapter 4 but also compare it to two other possible solutions.

The first solution completely disregards blockchain technology and only uses Git both to store the BIM data and also to save the votes. The voting process in this solution is not implemented in a smart contract but in a conventional application.

The last solution uses only blockchain technology. Both the storage of the BIM data and also the votes are implemented as a smart contract on the Ethereum network. The voting process is on-chain as well, the same way as in our ChangeManager prototype.

In the next section we present our evaluation question by question, provide the measurements along our defined metrics, discuss them and formulate answers based on the results. Finally, we present the comparison of the three solutions, based on the evaluation results, in the evaluation summary.

5.4.1 What are possible economic issues?

General economic concerns

Generally, economic issues are expressed in the costs of a project. These costs differ between the used technologies and solutions. To provide a solution, which is accepted by the stakeholders in the construction domain, it needs to be more cost efficient than current workflows. In this part we break down our solution into the part responsible for storing files and the costs of the change management process itself.

When we look at the costs for a system involving the Ethereum blockchain there is an unavoidable dependency on the price of ether. This is the cryptocurrency which is used on the Ethereum network, and it is, indirectly through the gas system, used to buy computing power. This can be a big disadvantage because the price of ether has a high volatility of up to about 10% for the "30 Day ETH/USD Volatility"¹. Construction businesses need to plan and calculate with the current and future costs of the used systems and, therefore, for the construction businesses it is better to have a low volatility.

¹<https://www.buybitcoinworldwide.com/ethereum-volatility/>

We use the price of \$110.34 for 1 ether for our calculations. It is the price observed on the 05.12.2018. We round all monetary values to two decimal places. Every instruction on the Ethereum network has a fixed gas cost, which has to be converted into ether by setting a gas price. For our evaluation we choose 7 gwei which translates to 0.000000007 ether, and is our conversion standard between gas and ether for this evaluation. ETH Gas Station describes this value as being the most inexpensive value while still being able to get a transaction into a block within a maximum of 30 minutes². Everyone who wants to get a transaction into the Ethereum network is in competition with all other transactions, which are also queued to go into the blockchain. Therefore, if there is a high usage of the network, the gas prices are also rising which can lead to higher prices, which can in turn become an economic issue for construction businesses.

5.4.2 What are the costs for storing big files?

To be cost-efficient, the cost of storing the BIM models has to be as small as possible. We evaluate the Ethereum blockchain and the Git repository based cost to store 10 gigabytes of data.

Costs of data storage on the Ethereum blockchain of 10 gigabytes

According to the Yellow Paper, which is the technical specification of the EVM, and was written by Wood, the current cost to store a 256 bit word is 20,000 gas [Woo14]. This means that the costs of storing a 256 bit word translate into \$0.02 at the previously set gas and ether price. This means that for 10 gigabytes, which translates to storing 3,906,250 256 bit words, the price would be at \$607,031.25.

Costs of data storage in Git repository of 10 gigabytes

Storing files using Git is generally free, only the hosting of the repository produces costs. Probably the most popular hosting services are Github, Bitbucket and Gitlab³. For example, storing files on GitHub is completely free for public projects; however, it has a repository size restriction of a maximum of 1 gigabyte⁴. For Bitbucket, a repository owner has to pay 25\$/month to be able to store up to 10 gigabytes of files⁵. However, they also have a hard limit on the repository size of 2 gigabytes⁶. Another provider, GitLab.com, has a file size limit of 10 gigabytes⁷. Git itself does not have a limit on file or repository sizes. Therefore, if a user wants to use a self-hosted system, she would need to buy or rent a server and run Git on it. Since all these services differ in price, we

²<https://ethgasstation.info/gasrecs.php>

³<https://medium.com/flow-ci/github-vs-bitbucket-vs-gitlab-vs-coding-7cf2b43888a1>

⁴<https://help.github.com/articles/working-with-large-files>

⁵<https://bitbucket.org/product/pricing>

⁶<https://confluence.atlassian.com/bitbucket/what-kind-of-limits-do-you-have-on-repository-file-size-273877699.html>

⁷https://docs.gitlab.com/ee/user/admin_area/settings/account_and_limit_settings.html

cannot give a definitive number here but we can safely assume that a hosted Git version would always be cheaper for storing 10 gigabyte compared to the on-chain solution.

We can conclude that the storage of big files on the Ethereum blockchain is not viable, because of the high cost. The hosting of a Git solution is more cost-efficient.

5.4.3 What are the costs for the change request management process?

General costs of processes on the Ethereum blockchain

The cost of processes on the Ethereum blockchain depend on the EVM instructions used for smart contracts. Information about instructional costs can be found in the Ethereum Yellow Paper [Woo14].

Costs of the voting process of the ChangeManager prototype

We use our ChangeManager prototype to evaluate how much gas is spent for the function calls, which are, in fact, transactions in the Ethereum system. Since every piece of information saved on the blockchain needs additional gas, we will only store the most important information (Git Hash) and leave the rest empty (Costs, Estimation, Additional Information) to keep the storage cost as low as possible.

We use the following parameter for our evaluation:

- GitHash: "0x802f99e42756b405081a36ee8e9a4b19393dad64"
- additionalInformation: ""
- costs: 0
- estimation: 0
- responsibleParties: ["0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db"]

The gas expenses for the ChangeManagerContract and its functions are the following:

- Deploying the ChangeManagerContract on the Ethereum network: 1,480,358 gas (\$1.14)
- Creating a new ChangeRequest on the ChangeManagerContract: 108,634 gas (\$0.08)
- Manage a ChangeRequest with one responsible party: 90,745 gas (\$0.07)
- Vote on a ChangeRequest: 22,052 gas (\$0.02)

As we can see, the voting process is quite cheap in comparison to the price of storing data on the blockchain. Deploying the ChangeManagerContract on the Ethereum network is by far the highest expense, but it is a one time fee, which has to be spent once per project. Once the contract is in place it can be used for many ChangeRequests. Creating new ChangeRequests and voting on them costs less than \$0.10 per transaction but we have to be aware that this price is also dependent on the parameter we are choose for these calls. In our example, we have chosen the smallest sized parameters possible to isolate the storage from the processing costs.

It is also important to note that retrieval of the data saved by these functions does not involve any gas costs. A user only has to download the blockchain as a whole and scan the address where the ChangeManagerContract was deployed for the predefined event types. There are no transactions needed to read events.

General costs of processess in a Git repository

The votes can also be directly recorded using Git. In this scenario, a stakeholder adds a digital signature to a commit made by another stakeholder to show his acceptance. On Github, such a system is free for a public project or it costs 7\$ per month for private repositories. Bitbucket and GitLab allow free repositories to be private as well, so there aren't any costs. Whether or not a private repository is needed for a construction project depends on whether the stored BIM models should be available for the public or not. A self hosted Git repository is another solution. Its cost depend on server prices.

5.4.4 What are possible technical issues?

General technical concerns

Possible technical issues for a solution built on the Ethereum ecosystem could include the very early development state in which most used technologies are at. Although they are used by a lot of people and adoption is growing they still have to prove themselves.

Another technical issue with the Ethereum network is that the blocktime is only one block produced in about 15 seconds^{8,9}. These blocks are also limited in size by the block gas limit of 8.000.000 gas¹⁰, which means that, as a result, the number of transactions included per minute is also limited. We have already discussed the issue of growing gas prices in times of high network usage. This phenomenon can also lead to a congested network, when more transactions are coming in than the mining nodes can put into the blocks. This can become a technical issue where stakeholders have to wait a long time for their votes to be registered.

Furthermore, the blockchain size, which, for Ethereum, has already reached 116.92 gigabytes on the 30.11.2018, could grow to become an issue. It grows linearly, which

⁸<https://etherscan.io/chart/blocktime>

⁹<https://medium.com/coinmonks/blockchain-scaling-30c9e1b7db1b>

¹⁰<https://medium.com/@piyopiyo/how-to-get-ethereum-block-gas-limit-eba2c8f32ce>

means that this growth could lead to substantial hardware costs in the future¹¹, when each stakeholder needs to run her own node.

5.4.5 Does the solution allow the storage of big files?

Storage limitations on the Ethereum blockchain

On the Ethereum network the block gas limit is currently set to 8,000,000 gas and it costs 20,000 gas to store a 256 bit word. This means, that there is a theoretical maximum of 102,400 bit (about 1 megabyte) in one block. This maximum is too low in reality because the process of storing this data, such as the transaction, is not included in this calculation. This means that, to store a file of 10 gigabytes, it would need to be split up into about 10,000 smaller pieces, which creates an additional overhead.

Storage limitations in a Git repository

There is no limitation of file sizes in the Git protocol. However, as we already described hosting providers not only have limitations on repository sizes but also file size restrictions. Github, e.g., allows a maximum of 100 megabytes per file, Bitbucket and GitLab only list repository size limits on their websites. To enable the storage of big files on Github, there is the project Git Large File Storage (LFS), which allows the storage of large files¹². It is an open source project, that replaces big files in a Git repository with a text pointer and transfers the file itself onto a remote server, such as, e.g., Github.com. It provides the usual Git workflow for handling repositories and commits. Large files are filtered out of a repository before pushing them to another server and are replaced by the text pointer. The text pointer contains a hash of the content so that the integrity of the repository is still safeguarded. The files which have been filtered out are stored on a remote server.

5.4.6 How much time does it take to push data onto the storage?

Time to get data into a block

The time it takes a transaction to be included into a block on the Ethereum network depends on the gas price, which is set by the transaction creator. In the previous examples we chose a gas price of 7 gwei (0.000000007 ether) so that a transaction would get included into a block within a maximum of 30 minutes. The ETH gas station lists 7.5 gwei as the gas price needed to include the transaction within 5 minutes and 18 gwei to have it included within 2 minutes. This is always dependent on the usage of the Ethereum network and is, therefore, constantly changing.

If one wants to split up a file of 10 gigabytes to include it into multiple blocks, as in the previously mentioned example, one would have to wait at least 10,000 times 30 minutes

¹¹<https://etherscan.io/chart2/chaindatasizefast>

¹²<https://Git-lfs.Github.com/>

if you wanted to get the cheapest price within a reasonable time. This would amount to about 208 days, which is not acceptable for our use case.

Time to get data into a Git repository

On Github it is possible to push a 100 megabytes file in under a minute, depending on the user's connection speed capacity. However, files bigger than that get rejected.

On Github, the user has to pay 5\$/month for 50 gigabyte of bandwidth and 50 gigabyte of Git LFS storage. However, there is also a file size limit of 2 gigabytes¹³.

On Bitbucket, it costs 10\$/month for 100 gigabytes without bandwidth restrictions and the service can be used by free accounts¹⁴.

The time it takes to upload files which are bigger than 1 megabyte to a Git repository or a remote server used by Git LFS is dependent on the user's internet connection speed and on the server connection. However, assuming a common broadband connection and a server, such as the ones provided by Github or Bitbucket, the transfer of files is faster than saving data to the Ethereum network. On Ethereum it takes 30 minutes for a file with a size of only 1 megabyte, whereas in our test we were able to push a file with a size of 100 megabytes in under 1 minute onto a Github repository.

5.4.7 How much time does it take to cast a vote?

Time to get a function call into a block

Votes, which have to be recorded on the Ethereum network and fit into one transaction, would get included within a maximum of 30 minutes when using a gas price of 7 gwei. Assuming best case conditions in terms of transaction congestion and worst cost efficiency, the data could theoretically be included into a block in about 15 seconds. This is the blocktime which we already described in the section about the general technical concerns. However, this calculation does not account for the time delay which occurs when a transaction gets propagated through the network. To make sure that a transaction has received sufficient finality, a user would also have to wait for 11 additional blocks to be added on top of the block where original transaction was included. That takes about 3 minutes (165 seconds). After this point, the user can be certain that her transaction cannot be removed anymore. In the case of using 7 gwei as gas price he would have to wait for 30 minutes plus the 3 minutes until 11 other blocks are added.

Time to create a new Git commit and push it

Since a vote can be assumed to be of a small size, it can be pushed to a Git repository very quickly and will in general not take more than 15 seconds, let alone 3 minutes.

¹³<https://help.github.com/articles/about-git-large-file-storage/#pointer-file-format>

¹⁴<https://confluence.atlassian.com/bitbucket/Git-large-file-storage-in-bitbucket-829078514.html>

5.4.8 How does the underlying technology scale?

Scalability issues for Ethereum

We already described that the time it takes to include a transaction into a block is dependent on the cost a user is willing to spend and on the workload of the network. Another problem on the protocol level is the continual growth of the blockchain file, which every full node needs to process. Those are two of the biggest problems blockchain systems are facing nowadays and many companies and researchers are working on solutions to improve this situation [Kre18][RV17]. The issue is that, if you want to provide better scalability, which translates into more performance, there has to be a trade-off, which is often at the expense of decentralization. Decentralization translates into security.

For a construction business, there exists the possibility to use a node hosting provider, such as infura.io¹⁵. This service provides scalable blockchain infrastructure, so that stakeholders would not need to host their own Ethereum node. The tradeoff is that they need to trust infura.io, which defeats the purpose of decentralization. Another way to circumvent this issue is to use a light client¹⁶. These light clients only verify the block headers and rely on full nodes to relay transactions and verify the blockchain as a whole.

Scalability issue for Git

Hosting providers, such as Github and Bitbucket, handle huge software development projects, such as, e.g., the linux kernel¹⁷. These projects have many people working on one repository at the same time. A construction project would normally have far less users contributing to the BIM model. The infrastructure has already proven to be reliable, over many years.

Nevertheless, there are two known scalability issues on the protocol level of Git, which could have a negative impact for a construction project. One issue that is related to scalability is the storage of big files in a Git repository which we already discussed. The project Git Large File Storage could provide a good solution for this¹⁸. The second scalability issue is that Git becomes quite slow performance-wise when it has to handle a lot of files in one repository. This could be prevented by splitting up a big repository into smaller ones.

5.4.9 What are possible political issues?

General political concerns

For blockchain systems, there is the threat of collusion of nodes. The power of a mining node is measured in its hash rate, which means that, due to its hardware setup, it can produce x amount of hashes per second. The more hashing power a node possesses the

¹⁵<https://infura.io/>

¹⁶<https://www.parity.io/what-is-a-light-client/>

¹⁷<https://Github.com/torvalds/linux>

¹⁸<https://www.infoworld.com/article/2955650/development-tools/git-isnt-good-enough-version-control-enterprises.html>

higher the statistical probability that this node will produce the next block and get the block reward.

Since the probability that a single mining node will win this race is extremely small multiple nodes combine their hash rates in a mining pool. The two biggest mining pools, Ethermine and Sparkpool together currently cover about 50% of the hash rate¹⁹. In theory, these mining pools could collude and launch a 51% attack. Such an attack could be used to double spend transactions. This means that the attacker secretly forks off his private chain and spends a transaction on the main chain and, simultaneously, on his private chain. He waits for the transaction on the main chain to be accepted by his victim and then publishes his own forked chain. Since he has 51% of the total hash rate to mine his forked chain, this chain is longer. Defined by the Proof of Work consensus, the longest verifiable blockchain available represents the commonly accepted truthful state²⁰. The more confirmations a block has already received after being accepted into the blockchain the less probable it is that this block can be removed again from the chain. It also gets more expensive for the malicious actor the more confirmations he has to wait for until his double spend transaction is accepted, since he has to secretly mine all the confirmation blocks too.

The creation of an arbitrary transaction from an address, which does not belong to the attacker, is not possible, since he does not know the private key of the address.

The 51% attack is a big risk for small blockchains. However, for Bitcoin or Ethereum, it is extremely expensive to gain such a big percentage²¹ of the total hash rate. Therefore, they can be seen as statistically safe against a 51% attack.

For our prototype, the double spend attack would mean that a malicious actor could accept a change request on the main chain but reject it on his secretly forked chain. This means, one stakeholder could vote for a change request on one chain and vote against it on another chain. The possible harm that such an action would have on a construction project is beyond the scope of this work.

5.4.10 How centralized is the solution?

Data storage location of the Ethereum blockchain

In the Ethereum system, all full nodes have a copy of the whole blockchain state on their physical machine. This means that there are thousands of copies of the data available around the world. If one node has an outage it does not hurt the network overall, since every node tries to constantly be in sync with the canonical state.

In this network, there is also a consensus algorithm, which decides who is allowed to publish the next block. All mining nodes have a certain statistical probability that their mined block is accepted to the top of the blockchain. This probability is directly linked to the previously described hash rate of the node. Since multiple miners can work together in a mining pool, there have been some concerns about the centralization or potential

¹⁹<https://www.etherchain.org/charts/topMiners>

²⁰<https://www.crypto51.app/about.html>

²¹<https://www.crypto51.app/>

collusion of nodes in a mining pool [ale18]. According to the cited article, the miners would want to protect their mining venture and the acquired hardware because every negative influence on the ecosystem would drive the price of the mined currency down. Therefore, it is in their best interest to keep the mining safe and decentralized. This is the premise on which blockchain systems are based. The cost of an attack against such a system has to be higher and the rewards smaller, in comparison to benevolent behavior.

Data storage location of a Git repository

If one uses Git to store the data, it is stored in every repository where the state has been pushed to or pulled. If the hosting provider Github is used, the data would be stored on the central Github server and everyone with read access to the repository can clone the state from the server to a local machine. The state on the remote server is seen as the canonical state. In a system, where a central Git repository is used as a hosting platform, the admin has every permission to create, edit and delete data from this repository. However, if an admin changes or deletes data, and another stakeholder had the data already cloned to her computer, then the commit hash from her history will be different to the one on the central server, which proves that some data has been changed. Either way her local data is independent of the change or deletion of the data on the central repository, which also represents a form of decentralization.

Following this, we can assume that Github, or similar hosted services, are much more centralized, which means that there is one single point of failure that could bring a project to a halt and lead to possible data loss.

5.4.11 Conditions to create a collusion to falsify the stored data on the Ethereum blockchain

On the Ethereum blockchain, to change, i.e., remove data, which has already been included into a block, an attacker needs to remine every block starting from the one that contains the transaction she wants to tamper with. This is the only way to create new hashes that will be positively validated and accepted by other nodes. It is important to note, however, that due to its cryptographic system, counterfeited and therefore invalid transactions cannot be included into the blockchain. An attacker can only remove valid transactions and their data from a block. To create a valid transaction for another user she would need to know his private key.

Once she has changed the block, she then has to build blocks on top of it until her blockchain is longer than the blockchain with the correct block in it. As we have already seen, the more blocks are built on top of another block the higher the finality or immutability property of this block gets. This means that the hash power needed to revert the blockchain gets higher the more blocks need to be remined to tamper with data in an already included block.

One issue, which can also be seen as a feature of blockchain solutions, is that it is not possible to tell which person is in control of how much hashing power. It is also not

possible to count the number of mining nodes because they can just be part of a mining pool, which appears as one node to the network²². The anonymity of the nodes can be a problem, since we are neither able to prove nor disprove that more than 51% of the hashing power belongs to, or can be used by, one entity.

Conditions to create a collusion to falsify the stored data on a Git repository

Git does not include a sophisticated access control mechanism and ,therefore, relies on the permission system of the underlying host system. A repository can either be read-only or certain users can have read and write access. A common strategy is to have two central repositories where one can be accessed publicly and the other one is tightly access controlled. There are various hosting solutions that allow a more fine grained access control to the repository. One example is Gitolite²³. Github and Gitlab support features, such as branch or tag based push permissions.

Generally, the administrator or owner of a Git repository has the full control over the data. The same permissions are given to every other user who is able to push to a repository. Users with push permission can add, edit and remove commits without any restrictions. However, if another user had the state of the Git repository before the changes locally, the Git commits will not be the same on the administrators repository. Therefore, although the administrator has the ability to change the data, the users who possess a clone of the previous state will know that something has changed. However, if there are two versions of the repository (the local and the remote) there is no way in the Git protocol to prove that one has existed before the other and is, therefore, the truth.

Although it is possible to prove that data has been changed on Git, it is not possible to prove which data is correct. And since it is quite easy for administrators to change data on their repository, the administrators have to be rated as a trusted third party. On the Ethereum network, on the other hand, it is very hard to change data that has already been inserted into a block, followed by a certain amount of confirmations. The data itself can not be added or edited without the according authorization either, but a transaction and its data can be removed. It is also possible to prove that the data was inserted into the blockchain at a certain time²⁴. This is not possible on Git because the timestamps can be faked by the users.

²²<https://bitcoin.stackexchange.com/questions/24365/does-getaddr-bitnodes-io-find-all-bitcoin-nodes-or-only-one-node-per-mining-pool>

²³<https://Git-sChangeManager.com/book/en/v1/Git-on-the-Server-Gitolite>

²⁴<https://medium.com/@kiknaio/what-is-proof-of-existence-and-how-will-it-help-to-protect-intellectual-or-private-property-77aa97a3fbb1>

5.4.12 Who is in control of the data?

Conditions to make unauthorized data changes on the Ethereum blockchain

- An Attacker needs to have more than 51% of the hashing power to be able to remove data.
- Users can recognize the reorganization of the blockchain in their client ²⁵.
- It is possible to prove that data has existed at a certain point.

Conditions to make unauthorized data changes on a Git repository

- An Attacker needs to have push access to the repository to be able to add, edit or remove data.
- Users can recognize the changes on the remote repository if they want to pull and get merge conflicts.
- It is not possible to prove that data has existed at a certain point.

5.4.13 How does the ecosystem around the solution work?

Both Ethereum and Git use many different open source technologies.

Development process of the Ethereum node software

For Ethereum, one needs a client to run a node. The Ethereum protocol is a specification²⁶ created by the Ethereum Foundation²⁷. The most widely used ones are geth²⁸, a reference implementation written in Golang by the Ethereum Foundation, and Parity²⁹, a third party implementation written in Rust by Parity Technologies³⁰. At the time of writing, the geth client is used by 76.6% of all nodes followed by Parity-Ethereum (version 2) with 18.3% and Parity (version 1) with 4.5%. Since these percentages amount to a total of 99.4%, all other clients do not have a significant impact. Since the creators of the specification and the client implementations have the potential ability to adapt the blockchain to their wishes, this also adds a layer of concern for everyone using the Ethereum blockchain as a means for securely storing data. We go into more detail concerning the development of these clients in the next section.

²⁵<https://bitcoin.stackexchange.com/questions/1061/can-a-51-attack-be-detected-and-dealt-with>

²⁶<https://Github.com/ethereum/go-ethereum/wiki/Ethereum-Specification>

²⁷<https://www.ethereum.org/foundation>

²⁸<https://Github.com/paritytech/parity-ethereum>

²⁹<https://Github.com/ethereum/go-ethereum>

³⁰<https://www.parity.io/>

Development process of the Git software

Git, on the other hand, is developed by an open source community and, since 2005, its maintainer is Junio Hamano [Neu15]. The maintenance of the project works through the `Git@vger.kernel.org` mailing list.

5.4.14 Who is developing the open source software?

Number of maintainers of the Ethereum node software and the Git software in comparison

In this part we list statistical data about the different open source repositories. The data originates from the respective repositories on Github, where the projects are hosted (Git uses Github as a publish-only and manages the repository through the mailing list).

Project	Number Contributors	Number Watchers	Managed
geth	383	2044	Github
Parity-Ethereum	179	374	Github
Git	1,244	2088	Mailing list

Table 5.1: Number of Contributors and Watchers

Since the maintainer role on Github is not publicly available for a project, we cannot determine who the actual maintainer of geth and Parity-Ethereum are. As we can see, the Git repository has significantly more contributors compared to the other projects, which can be interpreted as an indicator that its development is more decentralized than the geth or the Parity-Ethereum project. The advantage of open source software is that everyone can look at the whole source code. If the maintainers attempt to hide some malicious code into the clients, the probability of getting caught is higher the more people are watching the source code. Github has the "watch" feature, which notifies people who are watching a project if something is happening to the code. The number of watchers of the geth and the Git project are fairly similar - about 2000 users. In stark contrast to this, the Parity-Ethereum Project has only 374 watchers who are subscribed to the project. This means that it would be a lot easier to inject malicious hidden code in to the latter project without being noticed.

5.4.15 What are possible security issues?

General security concerns

As we have already seen, there are multiple attack vectors on the various protocols and their clients.

For the Ethereum network:

- On Ethereum, an attacker who gets 51% of the hash power over a time period can stop transactions from being included, or remove them from blocks.

- A miner centralization could lead to a 51% attack.
- If a user does not wait for enough confirmations after a transaction, the transaction could be removed because it had not been on the longest chain.

For Git:

- On Git, the owner of the central repository can tamper with the data in the repository and the users have no proof if their local or the central repository is the truth.

For open source projects:

- An attacker could inject hidden malicious code into an open source project.

Another security threat are the private keys used in a software system. This affects not only accounts on Ethereum, but also, e.g., SSH keys for a Git repository. If an attacker gets access to the private keys of a user, he henceforth has all the authorizations which the users themselves would have. On Ethereum, he can create transactions in the name of the user and has gotten himself the permission to tamper with the data. The same issue arises if a Git hosting provider like Github or Bitbucket is used and the SSH keys get compromised by an attacker.

Git faced a security issue because of the SHA-1 hash function it uses for various features. In 2017 Stevens et al. demonstrated that collision attacks on the hashing algorithm have become possible in practice [SBK⁺17]. The authors were able to create two different PDF files which resulted in the same hash value. Normally, two different files should always yield two different hashes. This is one of the defining properties of a hash function. The effort needed for this computation was equivalent to $2^{63.1}$ SHA-1 compressions and would take about 6500 years for a single CPU and 100 years for a single GPU to calculate. This means that an attacker could create two different repositories with different contents which could have the same hash as their head commit [Baa17].

Torvalds, who is the initiator behind the Git project, claims that the SHA-1 attack is not critical to Git since it is easy to detect. There is already a transition plan in the working to move from SHA-1 to SHA-256. In addition, since Git version 2.13 there has already been a move to hardened SHA-1, which is not vulnerable to this kind of attacks³¹.

Ethereum is a fairly new platform and, therefore, the research on this system is also quite sparse. Writing secure and well-performing smart contracts is difficult and there are many examples where bugs have already cost a lot of money. It is a new field in computer science and, although Ethereum's main programming language looks similar to JavaScript, there are many things a developer needs to think of when developing smart

³¹<https://mirrors.edge.kernel.org/pub/software/scm/git/docs/technical/hash-function-transition.html>

contracts on a blockchain. This can also be seen as a security vulnerability of the system. Reusable patterns and best practices, which new developers can use, are only just emerging. Wohrer and Zdun present a collection of such patterns that can be used to mitigate typical attack vectors [WZ18]. The authors also describe how the Ethereum ecosystem is evolving at a fast pace while instructions are added to the bytecode and bugs and security risks are discovered.

Furthermore, Mense and Flatscher show common vulnerabilities in smart contracts [MF18] and present a taxonomy, which classifies vulnerabilities on different architectural levels: Solidity, EVM bytecode and general blockchain characteristics. The complexity of the system itself can be considered a security risk.

Furthermore, the immutability of the code deployed on the blockchain can also be a burden, if the code contains a bug. Fortunately, there are already migration patterns in most solidity frameworks so that vulnerable code can be replaced by a new safe version of the code. This works through a migration contract which stores the address of the currently used smart contract. If this contract becomes vulnerable, the owner of the contracts can deploy a new smart contract and point the migration contract to the new and safe one. Since all of this happens on the blockchain the old smart contract will still exist and the switch to the new one will be fully transparent.

5.4.16 What are possible attack vectors on the data storage?

Conditions to bring the system to a standstill

There is also the threat of attacks, like a distributed denial of service (DDoS) attack, on the Ethereum network or on the Git hosting provider. Such an attack on a blockchain network can have different targets. Saad et. al. discuss a DDoS attack on the Mempool, where transactions are collected before they are included into a block, by creating a massive transaction backlog [STTM18].

The Ethereum network suffered a similar kind of attack, which can actually be classified as a denial of service (DoS) attack, in 2016 over the duration of a whole month. A malicious attacker used a bug in the code to create a huge amount of dead accounts, which he used to flood the network [Mee16]. These kinds of attacks, however, could not only be targeted at the blockchain network itself, but also at other parts of the crucial infrastructure of the blockchain project, e.g. its Github repository.

In fact, Github also has already been the victim of multiple DDoS attack. The last one in January of 2018 was the biggest DDoS attack in terms of traffic so far [New18]. Github had an outage of its service for about 20 minutes, during which they were able to successfully reroute the malicious traffic to their DDoS mitigation server. Afterwards, the service continued working as before. Although a DDoS attack cannot change the data, it can prevent users from accessing it, which can lead to a massive damage for the stakeholders who are not able to work with it.

In general, we consider the Ethereum and the Git protocol as similar in terms of vulnerability to a DDoS attack. Both enable the storage of all data both on a local

machine and on a remote server in a decentralized manner so that, if there is an attack on one point of the network, the other stakeholders can still access the data. However, if the Git repository is hosted on a central server or on a hosting provider, the risk of getting attacked is much higher because there exists a single point of failure that can be targeted by an attacker.

5.4.17 How private is the data?

Authorization permissions to read data on the Ethereum blockchain

If BIM models are directly written into the Ethereum blockchain everyone who is running a fully synced node has the data of the BIM models on his computer. The data that is being saved to the Ethereum blockchain is fully available to the public, which is one of the key reasons that the technology provides the traceability property. If data should be written privately to the system, without the possibility of unauthorized user access, it needs to be encrypted with a strong enough algorithm and the key to the encryption has to be shared through a different means.

Authorization permissions to read data on a Git repository

Git does not have any access control systems integrated. However, it can be configured so that only authorized users have access to a repository. Hosted solutions, like Github, do provide a sophisticated permission system enabling the creation of private repositories, where the administrator can add collaborators who are allowed to read or read and write, and everyone else does not have access to the data.

5.5 Evaluation Summary

In this section we summarize the findings of our evaluation.

On the topic of the economic questions, we determined that the volatility of ether could lead to a problem for construction projects. The volatility of the cryptocurrencies' value is especially problematic because the rest of the project is handled in dollar or another traditional currency. This makes the business calculations and, therefore, the budgetary planning difficult. There are some stable coin cryptocurrencies on the market which try to keep a 1:1 ratio to, e.g., dollar. The usage of such a stable coin could be explored as further research.

In addition, we have come to the conclusion that data storage on Ethereum is very expensive compared to our Git solution. The storage of the BIM models on the Ethereum blockchain does not make sense from an economic standpoint. BIM models can quickly grow to a huge size, which would also drive the storage costs up. The evaluation shows that there is also improvement potential for our ChangeManagerContract. Putting the additional information, the cost and the time estimation into the Git repository, instead of the smart contract, would minimize costs for our prototype even further.

On the other hand, smart contract processes are quite cheap on the Ethereum blockchain. The most expensive action is the deployment of the smart contract, because this can also be seen as a data storage operation costing about 1 dollar at the time of the evaluation. Then the subsequent change request creation and voting process is less than 10 cents per call. This could be cheaper than traditional paper workflows. However, this assumption would need further evaluation to confirm or reject it.

Considering the economic evaluation results, we can confirm the decision to use the Ethereum blockchain for the change management processes and Git for the storage as the best choice in comparison to an Ethereum or Git only solution.

Concerning the technical questions, we concluded that the storage of big files on Ethereum is not viable, because the data needs to be split up to fit into blocks, which have a limit of about 1 megabyte. All the parts of the file would need to be included into different blocks, which could take a long time since a new block is only created about every 15 seconds. The time needed to push a large file onto the blockchain is has proven excessive for our use case. The change management processes, which comprise of the creation of a new ChangeRequest on the Ethereum blockchain, managing and voting on it, on the other hand, can be considered viable, because, using this technology, the function calls should be finished in under one hour. This timeframe includes the needed confirmations to ensure certainty in the finality of the transaction. For this solution it is sufficient to store a hashed version, like a Git commit hash, of the BIM model on the blockchain to make the processes as cost-efficient as possible.

For the Git protocol, there is no technical limitation on the file size and the file transfer speed is only limited by the transfer speed of the hosting server. If a commercial Git hosting solution, such as Github or Bitbucket is used there are limitations on the file size

but there is a solution for this. Git Large File Storage is an open source project used by these providers to outsource the storage of big files to a cloud while still keeping a textfile with a hash of the big file in the repository. This ensures that the traceability, which the standard Git protocol provides, is still maintained.

As we have seen, a voting process on Git would also be very quick. A technical issue we would be facing, when trying to implement a voting algorithm into Git, is that it is not possible to prove that the vote has existed at a certain point in time. A timestamp in Git can be forged when committing a new change. This is not possible on the blockchain, because the timestamp gets inserted into a block when it gets mined, and if the block is not within a certain time range relative to the last block's timestamp the block is not accepted into the chain. This mechanism provides means to prove the existence of a transaction and its content within a fixed time period.

There are some concerns regarding the scalability of the Ethereum platform. The network can become clogged by transactions at times of high usage, which can lead to temporary increase of transaction prices. Furthermore, the increasing size of the blockchain itself is considered an issue when a stakeholder wants to host his own Ethereum node. There are many researchers working on finding a way to solve these problems. We see this as one of the biggest issues the Ethereum and other comparable smart contract platforms need to overcome in order to become a sustainable and usable solution.

On the topic of political questions, we discussed the (de-)centralization property of the solutions. In general, we found the Ethereum network to be better suited for the voting process because of the proof of existence mechanism and its better decentralization. On the Ethereum blockchain, an attacker would have to obtain more than 51% of the hashing power over a time period to be able to control the blockchain state. Although she would not be able to create arbitrary transactions, she could remove valid transactions or stop new transactions from being accepted into the blockchain state.

We have come to the conclusion, that even though it is possible to detect tampering with the data on a remote Git repository, the data can be changed by administrators at any time. The Git hash of the changed commit and its children in the tree would change, but there is no inherent way to determine which commit history is the truth.

One issue that we encountered in our evaluation is the rather small community which is developing the Parity-Ethereum client software. There are only 179 contributors registered and only 374 users watching the repository. This amount is rather small considering the distribution of the software on the Ethereum network, which amounts to about 22.8% of all full nodes. However, there is the alternative of the most used client software, geth, whose repository has 383 contributors and is watched by 2044 users. Regarding the Git protocol, we do not have concerns due to a lack of contributors (>1000) or watchers (>2000). Considering these numbers, we suggest using the geth client in combination with Git for the change management process.

On the topic of security, we discussed various concerns and attack vectors. For the Ethereum network, one concern is the 51% attack, which, due to the size of the whole

network, is very expensive in terms of needed hashing power. We discussed the general problem of the compromise of private keys and Git's weakness in the hashing algorithm SHA-1. However, in newer versions of the Git software, the issue of the SHA-1 algorithm has been largely resolved, so that its use can be considered secure.

Finally, we also touched on Ethereum and its novelty, which leads to many security issues because of inexperienced developers and missing best practices. Developers need to be very careful when they design and implement a smart contract. Although there are update possibilities for smart contracts, through switching out whole contracts, bugs can become costly before they are found.

We conclude that there are many attack vectors in the various technologies. As already stated, Ethereum can be considered as uninvestigated due to its novelty. We would advise to do more research before using this technology in a production environment. Formal verifications of smart contracts, which have been the topic of several research groups, already offer a way to ensure that a smart contract is working as intended [BDLF⁺16] [Mue18] [HSZ⁺17].

Git has been used by many developers for years and is considered very secure. If any vulnerabilities appear the open source community around the technology is quick to discuss the implications and provide a fix. This is due to the huge amount of people who contribute to the open source project. In our estimation, Git can be considered secure and safe to use in production.

In conclusion, our evaluation shows that it is not feasible to store BIM models directly on the Ethereum blockchain due to high costs and low performance of the protocol in terms of storage. Git can be used for the storage, but is also not ideal, because of its actual designation to be used for smaller textfiles. The Git Large File Storage could provide a solution for this issue and still offer the same mechanism of fraud detection, as the standard Git protocol does.

We also demonstrated that processes and function calls on Ethereum are cheap and complete in a reasonable time, when the network is not clogged. Smart contracts need to be optimized to store as little data as possible. In the case of our prototype, one could save some cost by only storing the Git hash on the blockchain and putting the additional information, cost and estimation directly into the referenced Git commit.

These insights confirm that a hybrid solution, where Git is used for storage and the Ethereum network is used for the change management processes is the most practical and viable solution of the three proposed.

However, we also discussed a lot of the weaknesses of the Ethereum environment. The issue of scalability is one of the deciding factors in the adoption of the technologie in production applications in the construction sector. E.g., a congestion of the network could be highly problematic for a construction project if time critical decisions have to be made on changes.

Another problem is the volatility of the used cryptocurrency, ether. Construction projects often need to plan their budget over several years. This makes the usage of a highly

volatile currency impractical.

The novelty of the technology is an issue as well. The development of smart contracts requires experienced developers due to the critical workflows they have to handle and the costly consequences that can result from bugs. There is a dearth of established best practices.

We recommend more research before using an application, such as a change management smart contract, in production. However, with further refinement of the prototype and its integration in BIM tools, such as SIMULTAN, a pilot project could be started to further evaluate its practical uses.

5.6 Threats to Validity

We evaluated the three solutions based on external research. Due to the novelty of Ethereum, there is only a small amount of peer reviewed literature in existence, which often times does not focus on our area of research. Therefore, we were not able to find enough relevant information in scientific literature. In addition, we also used information which we found in stackoverflow.com and quora.com posts and a variety of blog posts. The research on blockchain technology in general, but also on smart contract technology in particular, is growing quickly, which gives us hope that a more scientifically sound approach will be possible in future work.

The prototype we have developed is just an explorational prototype and is not yet optimized. The goal was to prove the viability of a change management application on the Ethereum blockchain in combination with a Git repository. Therefore, more design and development iterations are required to present a production-ready prototype, which could be used to further and more accurately evaluate the underlying technologies for the use case of a real construction project. Due to the limited scope of this work, we delegate this task to future research efforts.

Summary and Future Work

In this work we presented a new approach to a change management workflow. Based on the requirements of a construction project with multiple stakeholders with different roles, we used the combination of BIM and blockchain to propose a software solution to model this workflow. Furthermore, we designed and implemented an explorative prototype as a proof of concept using Git and the Ethereum ecosystem.

Finally, in the evaluation chapter we presented three different solutions, two of them only theoretical, and analysed their viability in the context of a construction project. We concluded that the hybrid approach that we used for our ChangeManager prototype is the most practical and viable approach. This solution uses Git as a data storage and an Ethereum smart contract for the actual change management process.

In the course of this work some challenges and aspects of our use case could not be addressed. We will suggest possible future work in this section.

In the course of the evaluation we also found multiple improvement possibilities for the ChangeManager in order to make it more efficient in terms of costs and security. In this part we present the improvement opportunities for future iterations of the ChangeManager prototype.

To decrease the cost, as much data as possible should be stored in the Git repository. It is enough to store the commit hash of a change request and the votes of each stakeholder on the blockchain. Additional information for change requests should be stored in the repository. The additional information for the votes could either still be saved in the smart contract or also put into Git commits, which are then referenced from the smart contract. This depends on the size of the additional information since, as we demonstrated in chapter 5, data storage on the blockchain can become very costly.

In terms of security, we consider more research on formal verification of the ChangeManagerContract necessary, before using it in a construction project. The current smart contract should only serve as a proof of concept and has not been reviewed in terms of security. A formal verification could prove that the ChangeManagerContract behaves

correctly no matter what the users provide as an input.

An evaluation if there is the possibility for the contract to become deadlocked because the stored data in the contract drives up the size of a transaction over the block limit, is an example case for future research in terms of reliability.

In general, to improve costs, performance and security, research on common best practices to be used in smart contracts could also bring further enhancements to the current prototype¹ [WZ18].

Another area of research for future work would be an evaluation in a real-world construction project. New requirements for the workflow would certainly arise and the prototype could be further refined. The evaluation of the integration of the ChangeManager into the SIMULTAN software could be used to generate new results. Since the SIMULTAN software is already used in actual construction projects it has proven to be an effective experimental interface for the interaction with BIM models by stakeholders. The integration with the ChangeManager software can be used to conduct user tests and further evaluate the combination of, on the one hand, Git and the Ethereum ecosystem and, on the other hand, BIM and blockchain. For the evaluation in a construction project, we suggest attempting a parallel use between the traditional workflow and our BIM and blockchain change management process.

Future research, on the blockchain layer future research could include the automatic payments to stakeholders once a change request with a certain cost and time estimation has been accepted. This could automate the monetary transaction process and exclude banks from the process. The advantage would be the decreased time it takes the money to get to and from the stakeholders. However, as we already demonstrated in chapter 5, the volatility of the underlying cryptocurrencies can be an issue when handling monetary value on the blockchain.

In recent years, there has also been some research on stable coins, which are cryptocurrencies whose value is connected to the value of a traditional currency [Orl17]. This is done by backing the stable coin which is, e.g., trading for \$1 per coin, by the same amount of actual dollars. There has to be a trusted third party which guarantees that 1 coin can always be exchanged for \$1. This could be especially interesting for construction businesses, which are calculating in US dollars.

A further area of research on smart contracts in the construction business would be on the question if data committed to the blockchain would be admissible in court. A legal study could evaluate if a vote on the blockchain is enough evidence to hold a stakeholder accountable for his acceptance or rejection of a change request.

¹<https://consensys.github.io/smart-contract-best-practices/>

In conclusion, although our evaluation answers a lot of questions, new questions emerged during the research, implemenatation and evaluation processes. In particular, questions of long term viability and how the BIM and blockchain usage will develop in general can only be answered in the future and through further research. This work and the explorative prototype it presents should be seen as a first step and can be used as the foundation for further research at the crossroads of these two research areas.

List of Figures

4.1	Parallel processes of committing a change on Git in red and handling a ChangeRequest on the Ethereum network in purple.	23
4.2	Possible states of a ChangeRequest and the transitions between them including the conditions	27
4.3	Change management process and the different possible states for a ChangeRequest.	33
4.4	Creation or Reuse of a ChangeManagerContract	34
4.5	Proposing a new ChangeRequest	34
4.6	Managing a ChangeRequest	35
4.7	Conducting a responsible vote on a ChangeRequest	36
4.8	Rejected ChangeRequest	37

List of Tables

5.1	Number of Contributors and Watchers	54
-----	---	----

Bibliography

- [ale18] alethio. Are Miners Centralized? A Look into Mining Pools. <https://media.consensys.net/are-miners-centralized-a-look-into-mining-pools-b594425411dc>, May 2018.
- [Aut02] Autodesk. Building Information Modeling. Technical report, Autodesk, 2002.
- [Baa17] Hans-Joachim Baader. Torvalds zur Zukunft von SHA-1 in Git. <https://www.pro-linux.de/news/1/24497/torvalds-zur-zukunft-von-sha-1-in-git.html>, February 2017.
- [BBF⁺18] T. Bednar, D. Bothe, J. Forster, S. Fritz, M. Gladt, C. Handler, N. Haufe, M. Hollaus, S. Jambrich, T. Kaufmann, L. Kranzl, G. Paskaleva, N. Rab, J. Schleicher, K. Schlögl, C. Steininger, and M. Ziegler S. Wolny. Simultane Planungsumgebung für Gebäudecluster in resilienten, ressourcen- und höchst energieeffizienten Stadtteilen. Technical report, TU Vienna, 2018.
- [BDLF⁺16] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. Formal Verification of Smart Contracts: Short Paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, pages 91–96, New York, NY, USA, 2016. ACM.
- [BRM17] Brian Bowe, Dan Robles, and Mathews Malachy. BIM+Blockchain: A Solution to the Trust Problem in Collaboration? *CITA BIM Gathering 2017*, 2017.
- [But13] Vitalik Buterin. A Next Generation Smart Contract & Decentralized Application Platform (Ethereum White Paper). Technical report, Ethereum Foundation, 2013.
- [BWM⁺17] Filipe Barbosa, Jonathan Woetzel, Jan Mischke, Jan Mischke, Mukund Sridhar, Mukund Sridhar, Mukund Sridhar, and Mukund

- Sridhar. Reinventing Construction: A Route To Higher Productivity. <https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/reinventing-construction-through-a-productivity-revolution>, February 2017.
- [CMVM18] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and Scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2018, Lisbon, Portugal, July 16-20, 2018*, pages 122–128, 2018.
- [Cou18] Stephen Cousins. French start-up develops Blockchain solution for BIM. <http://www.bimplus.co.uk/news/french-start-develops-blockchain-solution-bim/>, June 2018.
- [CR94] Victor R Basili-Gianluigi Caldiera and H Dieter Rombach. Goal question metric paradigm. *Encyclopedia of software engineering*, 1:528–532, 1994.
- [CS14] Scott Chacon and Ben Straub. *Pro Git - Second Edition*. Apress, 2014.
- [EDG⁺74] Charles Eastman, Fisher David, Lafue Gilles, Lividini Joseph, Stoker Douglas, and Yessios Christos. An Outline of the Building Description System. Technical report, Carnegie-Mellon University, Pittsburgh, PA. Institute of Physical Planning, 1974.
- [FBF⁺18] Ghareeb Falazi, Uwe Breitenbücher, Michael Falkenthal, Lukas Harzenetter, Frank Leymann, and Vladimir Yussupov. Blockchain-based Collaborative Development of Application Deployment Models. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences (CoopIS 2018)*, volume 11229 of *Lecture Notes in Computer Science*, pages 40–60. Springer, 2018.
- [flo16] flow.ci. GitHub vs. Bitbucket vs. GitLab vs. Coding. <https://medium.com/flow-ci/github-vs-bitbucket-vs-gitlab-vs-coding-7cf2b43888a1>, September 2016.
- [HSZ⁺17] Hildenbrandt, Saxena, Zhu, Rodrigues, Daian, Guth, and Ro,su. KEVM: A Complete Semantics of the Ethereum Virtual Machine. *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2017.
- [Kar16] Ghassan Karame. On the Security and Scalability of Bitcoin’s Blockchain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1861–1862, 2016.
- [KKC18] Soohyeong Kim, Yongseok Kwon, and Sunghyun Cho. A Survey of Scalability Solutions on Blockchain. In *International Conference on Information and Communication Technology Convergence, ICTC 2018, Jeju Island, Korea (South), October 17-19, 2018*, pages 1204–1207, 2018.

- [KN89] Kappel and Nierstrasz. Prototyping in einer objektorientierten Entwicklungsumgebung. *HMD, Heft 145, S. 116-125*, 1989.
- [KP12] Karen Kensek and Jinhua Peng, editors. *Practical BIM 2012*. School of Architecture, University of Southern California, The USC BIM Symposium, July 2012.
- [Kre18] Karl J. Kreder. BlockReduce: Scaling Blockchain to Human Commerce. *CoRR*, abs/1811.00125, 2018.
- [Kva18] Gabrijela Kvasina. *Dokumentation bei zyklischen Tunnelvortrieb - Erhebung von wesentlichen Parametern von Bauzeit und Kosten als Grundlage für ein digitales Modell*. TU Wien, 2018.
- [Mee16] Danielle Meegan. Ethereum Continues to Suffer From DDoS Attacks. <https://www.ethnews.com/ethereum-continues-to-suffer-from-ddos-attacks>, October 2016.
- [MF18] Alexander Mense and Markus Flatscher. Security Vulnerabilities in Ethereum Smart Contracts. In *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services*, pages 375–380. ACM, 2018.
- [MGC18] Salvador Martínez, Sébastien Gérard, and Jordi Cabot. Robust Hashing for Models. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18*, pages 312–322, New York, NY, USA, 2018. ACM.
- [Mue18] Bernhard Mueller. How Formal Verification Can Ensure Flawless Smart Contracts. <https://media.consensys.net/how-formal-verification-can-ensure-flawless-smart-contracts-cbda8ad99bd1>, January 2018.
- [MW03] E. M. Maximilien and L. Williams. Assessing test-driven development at IBM. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 564–569, May 2003.
- [Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. <https://bitcoin.org/bitcoin.pdf>, November 2008.
- [Neu15] Alexander Neumann. Vor 10 Jahren: Linus Torvalds baut Git. <https://www.heise.de/developer/meldung/Vor-10-Jahren-Linus-Torvalds-baut-Git-2596654.html>, April 2015.
- [New18] Lily Hay Newman. Github survived the biggest DDoS attack ever recorded. <https://www.wired.com/story/github-ddos-memcached/>, January 2018.
- [Orl17] José Ignacio Orlicki. A Stable Coin with Pro-rated Rebasement and Price Manipulation Protection. *CoRR*, abs/1708.00157, 2017.

- [PHdB17] Fernando G. Papi, Jomi Fred Hübner, and Maiquel de Brito. Instrumenting Accountability in MAS with Blockchain. In *Proceedings of the First Workshop on Computational Accountability and Responsibility in Multiagent Systems co-located with 20th International Conference on Principles and Practice of Multi-Agent Systems, CARE-MAS@PRIMA 2017, Nice, France, October 31st, 2017.*, pages 20–34, 2017.
- [RV17] Ravi Kiran Raman and Lav R. Varshney. Dynamic Distributed Storage for Scaling Blockchains. *CoRR*, abs/1711.07617, 2017.
- [SBK⁺17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The First Collision for Full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 570–596, Cham, 2017. Springer International Publishing.
- [STTM18] Muhammad Saad, My T. Thai, and Aziz Mohaisen. POSTER: Detering DDoS Attacks on Blockchain-based Cryptocurrencies through Mempool Optimization. In *13th ACM ASIA Conference on Information, Computer and Communications Security*, pages 809–811, 05 2018.
- [Wan17] Kyle Wang. Ethereum: Turing-Completeness and Rich Statefulness Explained. <https://hackernoon.com/ethereum-turing-completeness-and-rich-statefulness-explained-e650db7fc1fb>, July 2017.
- [Wen17] Jürgen Wensch. *Smart Contracts*. TU Wien, Wien, 2017.
- [Woo14] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger Byzantium Version. <https://ethereum.github.io/yellowpaper/paper.pdf>, March 2014.
- [WZ18] Maximilian Wohrer and Uwe Zdun. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity. In *Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on*, pages 2–8. IEEE, 2018.
- [YHKC⁺16] Jesse Yli-Huumo, Deokyoan Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where Is Current Research on Blockchain Technology? - A Systematic Review. *PLOS ONE*, 11(10):e0163477, oct 2016.