CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Democratization of Voting with Blockchain Technology

An Exploration of Problem and Solution Space in a Blockchain System utilizing a Case Study

Master's thesis in Computer science and engineering

DANIEL KARLKVIST

MARCUS AXELSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Democratization of Voting with Blockchain Technology

An Exploration of Problem and Solution Space in a Blockchain System utilizing a Case Study

DANIEL KARLKVIST
MARCUS AXELSSON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Democratization of Voting with Blockchain Technology
An Exploration of Problem and Solution Space in a Blockchain System utilizing a Case Study
DANIEL KARLKVIST
MARCUS AXELSSON

Supervisors:
Eric Knauss, Department of Computer Science and Engineering
Juho Lindman, Department of Applied IT

Examiner: Lucas Gren, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Democratization of Voting with Blockchain Technology
An Exploration of Problem and Solution Space in a Blockchain System utilizing a Case Study
DANIEL KARLKVIST
MARCUS AXELSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Blockchain technology has emerged in the last decade since the release of the Bitcoin white paper. The technology has later been applied to many different areas to solve recurring wicked problems within software engineering such as errorless security and complete transparency.

The project was designed to fill a research gap in the software engineering field of how to identify requirements (problem space), how to specify technical implementations (solution space), and how the connection between them interacts for a blockchain-based system. This was fulfilled by conducting a case study on the digitalizing of the Swedish election system because of its strong need for high security and transparency.

The study was based on data collected from interviews, literature reviews, and self-exploration by coding. The knowledge gained was used to create a requirements specification representing the problem space, together with an implementation of a smart contract in Solidity and TypeScript functions representing the solution space.

The findings revealed that requirements engineering can be used for blockchain development using traditional requirements engineering techniques. However, challenges emerged when identifying non-functional requirements because of quality factors in terms of security, reliability, efficiency, and transparency since these are more difficult to manage in a decentralized network. Moreover, limitations in the current state of blockchain technology led to that the solution for the case did not fully cover the entire problem space. In addition, it was discovered that the need for support from other systems when developing a blockchain application causes traditional software engineering problems to reappear in blockchain projects. Finally, previous research tells us the importance of focusing on problem space rather than solution space. However, it was observed that the problem and solution space for blockchain systems has a deep impact on each other, which in turn creates an issue if the main focus remains on the problem space.

Keywords: software engineering, blockchain, blockchain voting, requirements engineering, problem space, solution space.

Acknowledgements

We would like to first and foremost thank our supervisors Eric Knauss and Juho Lindman for continuously giving us highly appreciated feedback on the report throughout the project. The structure and content of the report would not have been the same without you.

We would also like to thank our examiner Lucas Gren for reviewing our work and giving us feedback needed to complete this project.

Furthermore, we want to thank our opponents Johan Gottlander and Theodor Khademi for additional input on the report.

Finally, thanks to all the interviewees for participating and giving us valuable information and comments for our thesis. This helped us immensely.

Daniel Karlkvist, Gothenburg, July 2023
Marcus Axelsson, Gothenburg, July 2023

Contents

1
Introduction

Software engineering (SE) is a continuously growing field with diverse applications in various industries. Many problems that arise in the industry are however often difficult to solve using traditional approaches. To ensure high-quality software development, several common SE practices have been established and refined over time. Agile development, continuous integration and deployment, code reviews, and version control are some examples [1, 2, 3, 4]. These practices are here to improve many aspects of a system including security, reliability, and maintainability. Even though these practices are in place, there are still breaches in software happening sporadically, even in big-tech companies such as T-Mobile, Amazon, and Microsoft to name a few [5]. Despite implementing various SE practices, security breaches and vulnerabilities are still a major concern in the industry since hackers and malicious actors are constantly finding new ways to exploit them in software systems. It is within this context of persistent and complex challenges that the concept of "wicked problems" [6] emerges, reflecting the intricacies involved in finding comprehensive solutions.

The wicked problem of ensuring a secure IT system is a complex issue that does not have a straightforward solution. A wicked problem [6] does not have a definite formulation, there is no stopping rule (in this case, a system can always be more secure), and they do not have an ultimate test of a solution. Another wicked problem relating to SE is making sure that a system is transparent. In most software systems, users are not able to see what is going on in the background of an application, which is an important part in order for the user to actually know how their data is being used. Software companies have been under scrutiny for a long time for gathering data on their users, and lawmakers in countries all over the world are constantly trying to make users safer on the web [7]. An issue like this is not easily solved, even with open-sourced code where everyone can review it, since there is no easy way to prove that the publicly available code is the one being used by the delivered software.

For a private company issuing an application, it could be considered "good enough" using a traditional approach with a centralized server governed by a single company. There may however be a bigger reason for concern when for example ensuring democracy in a digital election. This makes digital election an interesting case since it needs to consider secure and transparent aspects of the system differently.

A potentially better option than traditional SE for the problem of creating a secure and transparent digital election that has emerged in recent years is blockchain

technology [8]. Some parts of traditional SE, as mentioned before, are subject to wicked problems. Blockchain could however add new ways to solve a problem such as creating a digital election with its unique set of features [9].

Blockchain's rise in the area of digital elections is based on its decentralized structure [8] and the ownership is shared among the whole network. The interest and use in the technology started in 2008 [8] after a paper was published under the name Satoshi Nakamoto. It detailed an innovative way to transmit data across multiple networks utilizing a distributed database that is maintained by the network itself. This publication served as inspiration for blockchain technology which supports Bitcoin for example. Over time, this decentralized and trustless system has grown from its origins in Bitcoin into new applications allowing secure peer-to-peer transactions without intermediaries involved.

Apart from its use in Bitcoin, blockchain has been applied to various other areas. Another popular cryptocurrency, Ethereum, supports smart contracts [10] which are computer programs that enable the verification or enforcement of negotiations and agreements without human intervention. These self-executing protocols run on Ethereum's distributed ledger technology which guarantees a tamper-proof record where all parties can independently verify contract execution. Smart contracts execute automatically with high levels of transparency ensuring compliance for everyone involved after being triggered by pre-determined functions such as asset transfer, currency exchange, or specific task completion.

The potential of Ethereum smart contracts to decrease the need for middlemen in transactions is one of their main benefits. Contract execution can be automated, allowing parties to do away with the need for middlemen like banks, brokers, or attorneys [10]. As a result, transactions might be completed more cheaply, effectively, and securely.

In the fields of political science, computer science, and blockchain technology, the democratization of voting through the application of blockchain technology is a topic of growing interest and significance [11]. The use of blockchain technology in voting could help to solve many of the issues that traditional voting procedures now have. As mentioned before this includes transparency, security, and even accessibility for voters. Despite the achievements of blockchain voting there are still a lot of obstacles to overcome, for example external threats and transparency for users [12].

An organization named BLockchain IN Government (BLING) [13] which investigates the application of blockchain in government operations provided the inspiration for this project. BLING has been promoting the implementation of blockchain-based voting systems in various government organizations because it believes that the area of voting might substantially benefit from the properties of blockchain security and transparency. This has been done by accelerating the adoption and deployment of next-generation smart services, and this case study was conducted to contribute to the same field.

E-voting for elections has been present in other countries for many years. The most prominent case is in Estonia [14], which had its first digital election in 2005 and was used by more than 50% of voters in 2023. Blockchain-based solutions for voting and elections have also been developed and implemented in practice, such as the Moscow municipal election in 2019 [15] and the primary elections in West Virginia in 2018 [16]. However, these solutions have limitations, as they are either developed with a specific purpose in mind or are proprietary to a specific organization. To date, efforts to address the scalability, security, and accessibility challenges inherent in blockchain-based voting systems have been inconclusive.

The current methods of ballot box voting are plagued by challenges such as a lot of paperwork and transparency [12]. By implementing the voting process on the blockchain, these challenges can be addressed and the process can be made more secure and transparent. This could potentially increase participation and trust in the electoral process, as well as make it more efficient and cost-effective. The table below includes the potential advantages and disadvantages of using a blockchain-based voting system:

Advantages	**Disadvantages**
• Less human resources and paperwork • Faster and easier for the voter • Increased transparency • More secure if implemented correctly • Increased accessibility	• Risk for external threats on voting devices • Less understandable • Less established and not as accepted • Vulnerable to cyber attacks if implemented incorrectly

Table 1.1: Advantages and disadvantages of blockchain-based voting [12]
.

There are instances where conventional SE approaches cannot address a problem, particularly if it involves decentralization, security, or transparency as previously stated. An e-voting system that depends on these characteristics is an example of such a system. While research within blockchain technology has been tested as a potential solution multiple times, existing solutions have not been effective or are not suitable for large-scale elections [17]. The previous research on blockchain voting has also not focused on real-world constraints, or how the requirements for a system and possibilities of blockchain are connected. Considering the aforementioned factors, a lot of investigation has been done in the field but there is still a gap with digital elections in a world where most of our daily tasks are oriented around the web.

Since a lot of research has been done in the field, but there is still no solution that is trustworthy or established, this thesis will look into why that is and how to work with blockchain projects like this. The purpose of this thesis is, therefore, to

investigate how the needs, goals, constraints, and challenges (problem space) and technical implementation (solution space) can be specified for blockchain projects which aim for high security and transparency. Emphasis is put on high security and transparency since these are wicked problems within traditional SE. To do this, a case study was conducted on a blockchain-based voting system, since high security and transparency are needed for such a system. All countries have different laws and directions for elections [18] and could lead to different requirements and implementations of the system. This case therefore focuses on finding out how to implement the system based on one country's perspective. Since the study was conducted in Sweden and resources for the Swedish election are more accessible, it was chosen as the target country of the case.

Three research questions, denoted **RQ1-3**, are the focus of the study.

RQ1: *How can the problem space for a blockchain system be identified?*

The identification process will be conducted using requirements engineering to identify the problem space, and we want to investigate if requirements engineering can be used within blockchain projects.

RQ2: *How can the solution space be specified with blockchain technology?*

We want to evaluate how to develop a blockchain project and the difficulties arising from it. This question will therefore involve the development and implementation of a solution design that fulfills the problem space.

RQ3: *How do the problem space and the solution space interact when specifying a blockchain system?*

We want to investigate how the findings from the other research questions are connected, and how the interaction can affect the development of a blockchain system.

2
Theory

This section will provide information essential to understand the report. The chapter will include an overview of the most important aspects of blockchain and how it works, the technical challenges and limitations of using blockchain, smart contracts, cryptography, and the steps that are necessary to include in a digital voting process.

2.1 Blockchain Fundamentals

Blockchain is a decentralized, distributed digital ledger that can securely and irreversibly store data [19]. The technology has become more well-known because it can build trust without the aid of a central authority or middleman, making it a desirable solution for a number of industries, including voting [20].

Blockchain technology works by creating a decentralized network of nodes that share a ledger of transactions [21]. A transaction is either an agreement, transfer of data, or exchange of assets between two parties [22]. A transaction can be published to the blockchain for a fee, called the transaction cost. When a new transaction is initiated, it is broadcasted to the network and validated by a group of nodes using a consensus algorithm. Once the transaction is validated, it is added to a block, which is then added to the existing chain of blocks. The data is secured through cryptography, making it impossible to alter or manipulate without the approval of a majority of the network. This ensures the integrity and immutability of the data stored on the blockchain.

The blockchain network is built up of nodes, and any computational device that is linked to the blockchain network and takes part in the transaction processing and validation is referred to as a node in blockchain technology [23]. Examples of a node include computers, smartphones, and tablets. The specific nodes which are capable of expanding the blockchain with a new block are also called miners. The blockchain ledger, which includes all of the transaction records in the network's history, is kept by each node in the network. To make sure that all copies of the ledger are up to date and that new transactions are approved before being added to the blockchain, the nodes communicate with one another.

As described briefly above, the blockchain consists of multiple blocks that are linked together like a chain, hence the name. The transactions are then stored in these blocks. A block is comparable to a container that stores a collection of transactional

information [24]. A block can only store a specific amount of transactions, and the amount depends on the blockchain, for example, Bitcoin has an average of 2000 transactions per block [25]. When new transactions are created and the previous block is full, the process of generating a new block is started. The blocks are put on the blockchain once they are validated by the miners, and are connected by storing a hash of the previous block in the new block.

A blockchain can only handle a specific number of transactions per second (TPS), and developers or users can not modify this without an update to the whole network [26]. For example, the blockchain Ethereum can handle 13-15 transactions per second now, but a future update to the network is estimated to handle 100 000 transactions per second [27]. This update and increase in TPS is dependent on an idea called sharding [28]. Another possibility to increase the TPS for a blockchain is using second-layer solutions, which are dependent on implementations outside the blockchain [28]. The reliability of a blockchain is also something that is not modifiable by a developer or user, but the historical data for blockchains shows that there are minimal instances where the biggest blockchains have had any downtime [29] [30].

2.2 Blockchain Architectures

There are two main types of blockchain architectures that can be used, either permissoned or permissionless [31]. These two types can be used for different purposes and operations, and some areas benefit from using one over the other.

Permissionless blockchains are the most common ones, where the whole network is public and there is no central authority controlling the network [31]. The network is therefore aiming for decentralization and consists of pseudonymous actors. Anyone can be a part of the validation process, and anyone can send transactions. All data in the network is also public, which makes the network fully transparent. The drawback of using a permissionless blockchain is the scalability problems, risk of bad actors, less privacy, and high energy consumption [32].

A permissioned blockchain is the opposite, only whitelisted people are authorized to access the network [31]. Different users can be provided with different types of access, which enables them to do specific actions. A permissioned blockchain is therefore less decentralized and transparent and therefore most beneficial for a private setting. The main drawbacks of a permissioned blockchain is the centralization together with vulnerability to attacks [32].

A blockchain strives to fulfill three properties: scalability, decentralization, and security. Relying on traditional blockchain techniques only makes two out of these three achievable [33]. The vital parts of blockchain are decentralization and security, but in order to use it in bigger projects with a lot of requests scalability is needed as well. Below are the main features which are achieved for each blockchain architecture [33]:

- **Permissionless:** decentralization and security
- **Permissioned:** scalability and security
- **Multichain (multiple blockchains interconnected):** decentralization and scalability

2.3 Consensus protocol

Consensus protocol refers to a set of guidelines that are adhered to throughout transactions using blockchain technology [34]. As long as the consensus process is followed, there is a certain amount of confidence when updating or transferring data between end users. There are two types of consensus protocols that are mostly used, and these are Proof of Work and Proof of Stake.

The consensus method Proof of Work (PoW) is used to confirm transactions and add new blocks to the network. It is the blockchain's original consensus technique and is described thoroughly in the Bitcoin white paper [8].

Nodes (or miners) compete in PoW to find solutions to challenging mathematical puzzles [8]. In order to develop a solution that satisfies the network's requirements, miners undertake several calculations on the puzzle using their computers. The answer is subsequently validated by other network nodes, and if it is accurate, the miner who discovered the answer is rewarded with fresh cryptocurrency tokens.

Mining is a resource-intensive operation that uses a lot of electricity to run the computers that are doing the calculations [35]. As a result, there are worries about the effects of PoW mining on the blockchain ecosystem.

The second protocol is Proof of Stake (PoS), and it is a consensus algorithm that can be used instead of the PoW algorithm. In PoS, the nodes (validators) that confirm transactions are chosen based on the amount of cryptocurrency they hold and are willing to stake as collateral [36], as opposed to miners trying to solve challenging mathematical problems. The likelihood of being chosen to validate the following block of transactions increases with the size of the cryptocurrency staked.

PoS has several advantages to PoW, including being more energy-efficient [36]. PoW uses a lot of processing power to answer difficult mathematical problems, but PoS uses less processing power and therefore less energy. PoS also has the benefit of allowing for greater scalability [37].

2.4 Importance of the blockchain structure in a voting system

The way blockchain is structured is very important to give us the features needed in a digital voting system. A voting system must be decentralized [38] to ensure that

no single person, group, or organization has complete influence over the election process. It is ensured that the outcomes are not manipulated or skewed by any one party by using a distributed network of nodes that all take part in checking and validating votes. This enhances confidence in the outcomes and preserves the integrity of the electoral process.

A voting system must be transparent [38] in order for all participants to see and be able to confirm the election results. A public ledger that keeps track of all votes cast as well as the outcome of the election can serve to boost accountability and lower the possibility of fraud. Because of the way blockchain is set up, every vote is recorded in a clear and verifiable fashion, making it challenging to manipulate or conceal information. By doing so, you may boost voter confidence in the election process and make sure that everyone will accept the results.

A voting system must also be immutable and irreversible [39] to ensure that the outcomes of the election cannot be altered or tampered with. Votes are entered onto the blockchain and stored there as permanent records that cannot be changed or removed [21]. This assures that the results remain unchangeable, definitive, and helps to safeguard the integrity of the electoral process.

2.5 Technical challenges and limitations

The voting process could be revolutionized by blockchain technology, but there are significant technical obstacles and constraints to take into account. The issue of throughput, or the number of transactions that can be performed per second, is one of the biggest obstacles [24]. It can be difficult to scale a blockchain-based voting system to handle high amounts of votes because the present throughput of most blockchain systems is generally lower than that of traditional systems with a central authority.

Another issue that could restrict the scalability of a blockchain-based voting system is size and bandwidth [24]. Each new transaction adds to the size of the blockchain, which can lead to a lot of data that needs to be stored and sent around the network. This can make it challenging to maintain the blockchain's integrity and security while also making sure that the system is still scalable.

Another significant obstacle that must be overcome when implementing blockchain in a voting system is security [24]. Despite the fact that blockchain technology is usually regarded as secure, there are still significant flaws that need to be fixed, such as the possibility of a 51% attack [40], in which one entity seizes control of the network and manipulates the results.

Wasted resources can also be a significant limitation of blockchain in a voting system [24]. It takes a lot of processing power and energy to create new blocks and validate transactions, which can have expensive fees and a big impact on the environment, especially if a blockchain with PoW is used.

In regards to usability, blockchain can also be difficult for a layman to understand [24]. This could cause problems with widespread adoption since there is uncertainty of not only how it works, but if it works.

2.6 Smart contracts

A program that can run on the blockchain is called a smart contract [41] and it consists of functions and states, like any other programming language. The smart contract technology enables developers to build decentralized applications, that can be used for any industry. A variety of programming languages can be used, and one of the most popular is Solidity for Ethereum.

To interact with a smart contract, it has to be deployed to the blockchain [42]. A deployed contract is represented by a decentralized address, which means that users can interact with it and access its functionalities. Specific functions can be executed by posting a transaction to the address related to the contract.

2.7 Cryptography

In computer science, it is a standard to protect information by encryption so that only those the information is intended for can read it. The information is encrypted with cryptographic protocols, and there are different protocols for different areas of use. Encryption functions and hash functions are the two main parts of cryptography, and there are multiple algorithms that can be used for both, where each has its own usage [43]. Encryption functions are used to encrypt a message. For example, hiding a message in a cipher text, that later can be decrypted to read the initial value. A key is needed to both encrypt and decrypt a message, and the key structure is dependent on the encryption algorithms.

Compared to encryption algorithms, a hash function can only be used to encrypt information and the data can never be decrypted, this is used for proofs. Hash functions are designed to ensure that two different inputs can not have the same hashed value. A common example of this is passwords [44]. It would be unsafe to store passwords in plaintext on a server, but also unsafe if it is encrypted since there is a way to decrypt it. Hash functions can therefore be used as proof to ensure that someone knows the given information by comparing a hashed message with a stored hash.

2.7.1 RSA-encryption

RSA, which stands for Rivest–Shamir–Adleman, is an encryption scheme that is used to transmit protected messages online. RSA is an asymmetric-key encryption

scheme, which means that two different keys are needed for encryption and decryption, these keys are called public key and private key [45]. The public key is known by anyone, and the private key is only known by the receiver. This means that anyone can encrypt messages with the public key, but they can only be decrypted with the private key.

2.7.2 Merkle tree

A Merkle tree, also called a hash tree, is a hash-based data structure that is used for data verification. The tree is built up with hashes, where each leaf node is the hash of data and the parent node is a hash of its children [46], see Figure 2.1. The tree structure is the same as a binary tree, which means that each parent node can have up to two child nodes. The tree can be used as proof and is typically employed when the leaves are identities and someone wants to prove its identity is part of the tree. This can be done by providing a Merkle proof, which is a set of node hashes and the plaintext for its own identity, that are used to backtrack the tree and recreate it [47]. If the root hash of the tree generated with the Merkle proof is the same as the root hash stored from creating the tree, the person providing the proof can be verified as a member of the tree.

There are multiple different hashing functions that can be used, and there is no restriction on which to use for a Merkle tree. In the field of blockchain, Keccak256 is one hashing function that is commonly used and has become more popular in the community [48]. Keccak256 accepts inputs of varying lengths and turns the data into a fixed-size output of 256 bits. This hashing function is popularly employed together with the Merkle tree data structure for whitelisting on the blockchain.

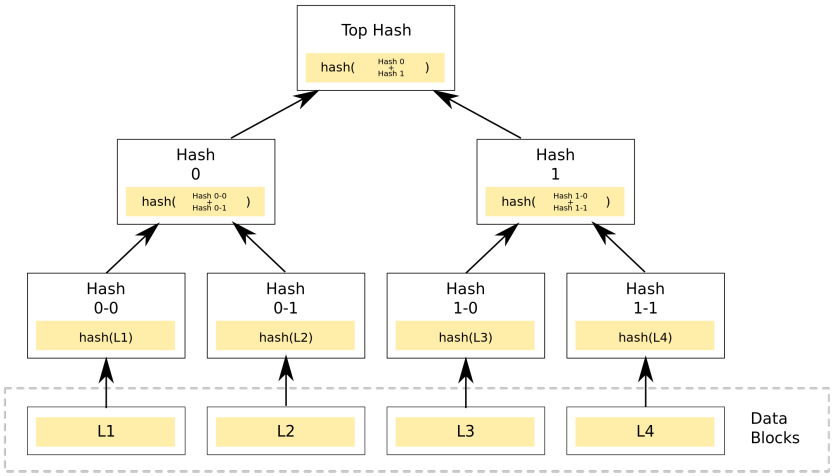

Figure 2.1: Merkle tree structure [49].

2.8 The digital voting process

To enable blockchain voting, there are four steps in the process that needs to be taken into consideration: registration, voting setup, voting, and result [50].

The first phase of the process is the registration phase. Voting is not possible without registration, which also guarantees that only eligible voters can cast ballots [50]. The voter's identification and other eligibility requirements, including age, citizenship, and residency, are gathered and verified as part of the registration process. Only authorized users should be able to reach the voting stage, hence identity verification should be employed to ensure this.

The second phase is the voting setup phase, which involves setting up the required security protocols and establishing the necessary information in the smart contract [50]. This entails developing smart contracts that will oversee the voting process, customizing the voter interface, and setting up the system needed for conducting an election. This process also includes getting information about the parties and candidates that are eligible to vote for.

The third phase is the voting phase, which entails actually casting a ballot, which is then recorded on the blockchain [50]. Voters can use a secure digital interface to cast their ballots, and those ballots are encrypted to protect the voters' privacy and confidentiality. The vote is recorded on the blockchain after it is cast, making it a permanent record that cannot be changed or removed.

The fourth and final phase is the result phase. Election results are stored on the blockchain, where they become a permanent record that cannot be altered or falsified [50]. The outcome is calculated based on the votes cast, and the winner is chosen by performing an off-chain calculation based on the data from the blockchain.

2.9 Requirements engineering

Requirements engineering is used at the beginning of a project to understand the demands and necessities of a software system. Lausen [51] emphasizes in his book "Software Requirements: Styles and Techniques" the importance of focusing on the problem space rather than the solution space. He means that gathering requirements, analyzing user needs, and understanding the context in which the software will be used, is more important than designing and implementing the software solution that addresses the identified requirements.

Requirements engineering is a process in the software engineering field that is used to determine how a system should behave, act and function [51], an example of this is the inputs from the user or what features the solution should have [52]. The process of finding requirements is called elicitation and a lot of different elicitation techniques can be used to identify what requirements a system needs [51]. The

elicited requirements are turned into a requirements specification where all demands and requirements are specified, together with their reasoning and description. The requirements specification consists of everything that a system will include and what the involved parties can expect from the project [53].

Requirements are usually specified into functional requirements and quality requirements [51]. The functional requirements specify how the system behaves and operates, as well as the features and functions of the system. One common technique to define functional requirements is with user stories, which include a feature that someone wants, who wants the feature, and why they want the feature [54]. Quality requirements, usually called non-functional requirements, specify the efficiency of the features. This includes security, efficiency, maintainability, and other quality factors. Most projects have quality factors that are more important than others, and a quality grid can be used to specify their importance [51]. The grid includes quality factors on one axis and importance ratings on the other, see Figure 2.2. For each quality factor, its importance is assessed and the most critical factors can be specified into requirements. The quality factors are categorized into operation, revision, and transition. The important quality factors can also be written using Planguage, which is a way to specify quality factors [51]. Planguage includes information on the factor and what it is about together with three metrics, what the system is required to handle, what the system is planned to handle, and what the system wishes to handle.

	Critical	Important	As usual	Unimportant	Ignore
Operation					
Integrity/security			X		
Correctness			X		
Reliability/availability		1			
Usability		2			
Efficiency			X		
Revision					
Maintainability			X		
Testability			X		
Flexibility			X		
Transition					
Portability					X
Interoperability	3			4	
Reusability					X
Installability		5			

Figure 2.2: Quality grid [51].

3
Method

Runeson et al. [55] describes four steps to be used when conducting a case study and these are the following: case study design, data collection, analysis of data, and reporting. The first step involves preparations such as purpose, what the case is, the method of data collection, and more. The second step is to gather all the data needed to conduct the case study, and the third step involves how the data collected should be analyzed. The fourth and last step is the reporting of the findings which will be placed in Section 4 of this report.

The case investigated in this thesis is a blockchain-based voting system in Sweden, and the case study involves looking into both the problem space and the solution space. This means that the needs, goals, constraints, and challenges of a digital voting system in Sweden will be specified (problem space), and the design and implementation of the voting system will be created (solution space). By looking into both the problem space and the solution space in one project ended up with a research method that is not standardized. The problem space was identified through interviews and literature review, and the solution space was identified through literature review and self-exploring.

RQ1 will be answered during the data collection and analysis phase, where the problem space has been identified. RQ2 can be answered when all data have been analyzed and a solution has been implemented. The last research question can be answered when both the problem space and the solution space have been identified and will be the last part of the case study.

3.1 Research Design

Runeson et al. [55] describes the importance of knowing why research is being conducted and the overall objective of the study. In this study, the reason for the research is to contribute to the software engineering field, and more specifically to the topic of blockchain, with how the development of new applications can be done and what to keep in mind when specifying the problem and solution space. Runeson et al. [55] also describe that the case that will be investigated needs to be a contemporary phenomenon in the real-life context, which in this case is the blockchain-based voting system. The data used in this case is based on the Swedish election and will provide insights and solutions that could be applied in a real-life situation. The case study will analyze the whole process of voting which includes

four phases, registration phase, setup phase, voting phase, and result phase and the problems related to these different phases.

The data needed for the study was collected through three approaches, interviews, literature review, and self-exploration. The interviews were the primary data collection technique, and the data collection from the interviews was complemented by data from the literature review and self-exploration in order to achieve triangulation. The interviews can typically be conducted in three ways, structured, semi-structured, and unstructured. Since we wanted answers to some specific areas and questions, but also wanted the freedom to go off script we decided to use a semi-structured interview style. The interviews were conducted in order to gather requirements for a voting system and to answer RQ1, but also to get ideas for how a blockchain based voting system can be implemented to answer RQ2. Additional interviews were conducted at the end of the project and used as validation, to gain knowledge on how well the other data had been specified and that all parts have been covered. This process brought knowledge to answer RQ3.

The literature review was used to validate the data from the interviews, gather more in-depth knowledge about the interview answers and blockchain implementation specifics, and also to fill any holes that remained after the interviews. Since more knowledge relating to the interview answers, and information relating to blockchain implementation was reviewed, this data collection method was used to answer both RQ1 and RQ2.

The third data collection source, self-exploration, was used to validate the data collected relating to blockchain implementation specifics in practice. This data collection method used information gathered from the interviews and literature review to gain knowledge on how theoretical blockchain development principles work in practice. The data gathered was used in order to answer RQ2.

This case study is dependent on two different fields, political science, and blockchain engineering. The chosen interviewees were active in one of these areas, and two people from each participated. Multiple Swedish authorities and people with different backgrounds in the blockchain field were contacted for an interview, some did not answer, others mentioned a lack of time and resources, and some were apprehensive to answer questions regarding blockchain due to its uncertain and complicated nature. The four selected interviewees were the only ones wanting to participate. The interviewees selected from political science are key stakeholders at the Swedish electoral authority, where one was a part of the development unit & technology, and the other was active in research regarding projects and organization. These interviewees were chosen from their deep knowledge of the Swedish election system and to gather as much in-depth information in regard to the voting process itself. The remaining two have been working in the field of blockchain for a long time, whereas one is a researcher, professor, and author in the field and the other is the CEO of a company whose product is used in multiple blockchain exchange applications.

3.2 Data collection

The data collection phase of the study included three parts, interviews as the primary source and literature review and self-exploration which was used to complement the answers from the interviews and triangulate the answers.

The interviews included preparations, contacting interviewees, and conducting the interviews. The interviews used a semi-structured approach and an interview guide was used during the sessions, see Appendix C. Around 20 people were contacted for an interview but only four participated. The interviewees that agreed to participate were key people both in the area of political science and within blockchain applications and provided necessary and valuable information to keep the quality of the study. The length of the interviews ranged from 60 to 110 minutes, and all interviews were recorded with the interviewees' consent.

The interview sessions started off with a brief explanation of the project and the goal of the session. Then it followed up with questions regarding the four phases, registration phase, setup phase, voting phase, and result phase. The questions were related to what requirements are needed during these phases, how the process works today, what needs to be taken into account when developing an election system, and how the interviewee envisioned a digital voting system to be implemented. In addition to questions related to the phases, some questions about security and the general topic were also asked.

The second data collection source was a literature review, which was used to validate the interview answers, gather more in-depth knowledge on the topics discussed during the interviews, and also to fill any holes that were left open after the interviews. The documents analyzed were therefore related to specific problems within electronic voting, for example, user identification and anonymity, what requirements an electronic voting system needs, relevant regulations and policies within the Swedish government, and blockchain development specifics. During this data collection, it was identified that the solution space should be created with a smart contract in the programming language Solidity [56] for the blockchain Ethereum [10].

Self-exploring was the third and final data collection source, and it was used to validate the collected data related to blockchain implementation and to understand the possibilities of blockchain development. It was used to gather data on how theoretical concepts work in practice, and in order to understand the capabilities of Solidity and smart contract development. This was done by implementing theoretical principles in Solidity, creating functions, deploying smart contracts, and calling smart contracts after deployment. This data collection process was held after the interviews and literature review, and the information gathered from these processes was used to gather practical knowledge within the same field. The implementation, testing, and deployment utilized the integrated development environment (IDE) Remix [57], which is an IDE for Solidity development.

3.3 Analysis of collected data

The analysis of the data was done through thematic analysis, and the answers were categorized into five themes, which were the four phases identified during the design phase and a category general for the whole process. This type of thematic analysis is called a deductive approach and is the process of analyzing and categorizing data into predetermined themes [58]. Since interview sessions were recorded, all interviews could be transcribed in full detail. A text-to-speech transcription application was used to facilitate this process. When the transcription was done, all interviews were listened to again to make sure the transcription program recorded all information correctly and also to get familiar with the answers. The transcribed interviews were then analyzed and the relevant information was categorized into the themes described above. The data collected during the literature review and self-exploration was also categorized into themes by going through the documentation made during those processes.

The problem space was explored with software requirements, and the requirements were created from the thematic analysis of the data. When all interview transcriptions were categorized into themes, requirements were generated from the answers and turned into a requirements specification consisting of functional and non-functional requirements. The functional requirements specified the features and functions of the system and were written in the form of user stories. The non-functional requirements specified the efficiency of the features and specified using a quality grid and Planguage.

When the requirements specification was generated, the solution-seeking process for the requirements started. The solution-seeking process began by finding solutions for every requirement and how they could be implemented into Solidity, finding different alternatives, and rating their pros and cons. This was done by going through the collected data from the literature review and the self-exploration, on how solutions should be developed and what techniques to be used for specific cases. When there was a set of solutions for the requirements, they were merged together to make a full implementation for the entire project. The selected solutions were evaluated on their effectiveness in addressing the identified requirements, as well as their feasibility and scalability.

3.4 Validation

Validation interviews were conducted at the end of the project to ensure that the requirements specification and the implemented system covered all parts and that they were specified correctly. The validation was done by conducting an additional interview with three out of the four interviewees. During these sessions, the requirements specification was reviewed first, where all requirements were described in detail and presented to the interviewee. The implemented system was demon-

strated after the requirements specification, to show the interviewee how we fulfilled the requirements. These sessions were between 40 to 60 minutes and were used to measure the success of the project and how well the spaces had been specified.

4
Results

In this chapter, the results from the study will be presented in three parts. The first part will display the requirements identified for a blockchain-based voting system. The second part presents the solution as well as the implementation of the smart contract code. The third part will delve into if the requirements are fulfilled by the solution presented in part two or not.

4.1 RQ1: Requirements

The requirements identified during the analysis have been divided into two types of requirements: the functional requirements which describe the tasks and functionalities needed to fulfill the users' expectations, and the non-functional requirements which capture the qualities and constraints that define how the system should perform. The full set of requirements will be presented below under the corresponding section.

4.1.1 Functional requirements

The functional requirements elicited are based on the interviewees answers. The interviewees from The Swedish Electoral Authority mentioned that their answers are derived from regulations used in the traditional voting system, which either are based on Swedish law or regulations decided by the Electoral Authority. The answers from the interviewees in the field of blockchain were based on regulations needed for a secure digital election. The requirements cover the whole process from registration to results needed to conduct an election in Sweden. The full set of functional requirements can be seen in Table 4.1.

The requirements are specified with user stories, to specify who the requirement are based for and what the reasoning behind the requirement is. The requirements related to what The Swedish Electoral Authority wants are specified as the product owner since they are the owner of the system, and the requirements related to what the voters wants are specified as the user.

ID	Functional requirement (label and user story)
FR1	**Digital voter registry** As a product owner, I want a digital voter registry, so I can keep track of who has voted and not.

FR2	**Date for registering parties** As a product owner, I want all parties to be registered at least 30 days before the election, so I can follow the current Swedish law.
FR3	**Date for registering candidates** As a product owner, I want all candidates to be registered at least two days before the election, so I can follow the current Swedish law.
FR4	**Voting only for eligible users** As a product owner, I want only eligible users to vote, so that the result represents the Swedish population.
FR5	**Vote from anywhere** As a user, I want to be able to vote from anywhere, so I don't need to travel to a booth.
FR6	**Prevoting** As a user, I want to be able to prevote and vote on the election day, so I can do it when it fits me.
FR7	**Revoting** As a user, I want to be able to revote, so I can change my vote if I regret my decision.
FR8	**Marking invalid votes** As a product owner, I want invalid votes to be marked, so we can identify potential problems and attacks.
FR9	**Connect vote to constituency** As a product owner, I want each vote to be connected to a constituency, so we make sure the candidates are representative for the whole country.
FR10	**Constituency level results** As a product owner, I want to be able to count the result on constituency level, so I can view the results in different areas of the country.
FR11	**Count votes after elections** As a product owner, I only want the votes to be able to be counted after the election is over, so that the election results are not affected by decision-making from real-time data.
FR12	**One vote per person** As a product owner, I only want one vote per person to be counted, so that the election is equal for everyone.
FR13	**Last vote counted** As a user, I want the last vote to be the only one counting, so that I can revote.
FR14	**See personal vote** As a user, I want to be able to see my own vote after, so that I can verify that it is correct.
FR15	**See all votes** As a user, I want to be able to see all votes, so that I can verify that the election results are correct.

FR16	**Untraceable votes** As a product owner, I want the votes to be untraceable, so that no one can see who made which vote.

Table 4.1: Functional requirements for a blockchain-based voting system

The interviewees had similar views on what the system needed, and their answers did not differ drastically. The only areas where the responses differed were regarding the topic of voting from anywhere and seeing personal votes. Which also were the areas where the interviewees gave feedback during the validation sessions.

The feedback on the requirements from the validation interviews was positive, but some requirements were commented on. The functional requirement FR5 (**Vote from anywhere**) was commented on by two of the interviewees, where one of them said:

> *"You have to choose one or the other, there are pros and cons to both. So when you design the system, you just have to choose."*

The other interviewee was more skeptical of it and said:

> *"There is a challenge with that. ... One of the points of having to go to a booth and vote is that you should avoid pressure. You know that you vote alone if you vote in a booth."*

The interviewees had different views on this topic during the elicitation interviews as well, where three out of four said that the system should support vote from anywhere and one said that it would be good, but it's also a risk and that the system should support voting from anywhere as well as from a booth.

Another requirement that one interviewee was skeptical about from the validation interviews was FR14 (**See personal vote**), and said:

> *"That is a bit complicated. ... The question is whether it poses a risk to the secrecy of the election or not."*

Overall the requirements specification showcase a variety of demands including the law and technical aspects. One of the interviewees wrapped up the functional requirements nicely:

> *"It is a mixture of requirements that has to do with legal aspects of election implementation and technical aspects on how you want the system to work."*

The interviewee also added that the requirements look good on a general level from a quick overview.

> *"I think the requirements are generally good, but I can not say that they are complete just from a short review like this."*

A few refinements were made after the validation meetings based on feedback. This included changing FR3 from stating that the candidates need to be registered 30 days before, to two days before. There were also minor rephrasings of the requirements to make them clearer.

4.1.2 Non-functional requirements

The non-functional requirements are used to support the functional requirements with quality factors. The non-functional requirements are represented in three levels: a quality grid, a table with specifications, and Planguage.

The quality grid, see Table 4.2, demonstrates the importance of certain quality factors in the non-functional requirements and includes an importance rating for each factor. Each quality factor in the table is connected to a rating, which is either a number or an x. Since this case is related to democracy, multiple factors need to be set as *Critical* to ensure a safe process. The numbered ratings are described underneath the table since these are of more importance, and the ratings *As usual* do not have a specific effect on the system compared to other quality factors and therefore do not need further explanation.

	Critical	**Important**	**As usual**	**Unimportant**	**Ignore**
Operation					
Security	1				
Correctness	2				
Reliability	3				
Usability					4
Efficiency	5				
Revision					
Maintainability			x		
Testability			x		
Flexibility			x		
Transition					
Portability			x		
Interoperability			x		
Reusability	6				

Table 4.2: Quality factors for the blockchain-based voting system.

Concerns from the quality grid:

1. **Security:** If the system is not secure, it could cause severe problems for the election.

2. **Correctness:** If the system can accept invalid votes, it could change the election results.
3. **Reliability:** If the system allows downtime, it could lead to transactions not being published.
4. **Usability:** The system in this case study does not consider the usability of the platform. Usability would be of greater value if a front-end application also was developed.
5. **Efficiency:** The system must be able to handle the number of transactions needed to conduct the election.
6. **Reusability:** The system needs to be reused in the parliamentary, county council and municipal elections for over 6000 constituencies across the country.

The quality grid was used to specify non-functional requirements within the quality factors set as *Critical*. The non-functional requirements specified for a blockchain-based voting system are presented below in Table 4.3.

ID	Non-functional requirement (label and description)	Category
NFR1	**Transaction speed** The system should be able to handle enough transactions to supply the Swedish election	Efficiency
NFR2	**Encrypt sensitive data** All sensitive data stored on the blockchain needs to be encrypted	Security
NFR3	**Safe from data tampering** The application should be safe from data tampering	Security
NFR4	**Applicable for multiple elections** The solution should be applicable for multiple elections (parliamentary, county council, and municipal elections) as well as multiple constituencies	Reusability
NFR5	**Disallow invalid data** The application should not accept invalid input data	Correctness
NFR6	**High uptime** The system should not have any downtime during the election	Reliability

Table 4.3: Non-functional requirements for a blockchain-based voting system.

A key aspect to point out from the non-functional requirements table above is that security is deeply rooted in blockchain. A project which uses blockchain needs security as perhaps its main focus for it to be used. One of the interviewees also mentioned this briefly after being asked about their thoughts on NFR3 (**Safe from data tampering**).

> *"From what I know about blockchain, the main feature should be that the data is where it is."*

The interviewee also stated that they felt that a shorter list of non-functional requirements is appreciated.

> *"I looked through a document with non-functional requirements yesterday. ... The document never ended because there are endless amounts [non-functional requirements] to relate to. It never ends, it just continues, both categories and what it is. So less is more I believe."*

The set of non-functional requirements, see Table 4.3, was used to operationalize the requirements with Planguage. These could then be used to easily identify if the non-functional requirements have been fulfilled or not. The metrics used are based on the needs of Sweden and cryptographic safety.

Request speed: The system should handle enough transactions to supply the Swedish election	
Scale	Total number of transactions during a day
Meter	Measured in number of transactions per day
Must	12 million transactions (based on e-voting in Estonia [14] accounting for 51% of casted votes, around 8 million people in Sweden [59] being able to vote, and there are three transactions per person).
Plan	20 million transactions (adding votes because there are likely going to be revotes by some people).
Wish	30 million transactions

Table 4.4: Planguage table for Request speed.

For Table 4.5 below, the encryption method is not provided and is instead left as an open metric. This is because encryption techniques can be easily interchangeable for this project and should be adapted to the state of the art. The used encryption was set to RSA, which is explained further in Section 4.2.

Encryption of data: All sensitive data stored on the blockchain needs to be encrypted	
Scale	All sensitive data stored on the blockchain
Meter	Encryption
Must	The sensitive data stored on the blockchain should be encrypted
Plan	The sensitive data stored on the blockchain should be encrypted with standard encryption
Wish	The sensitive data stored on the blockchain should be encrypted with above standard encryption

Table 4.5: Planguage table for encryption of data.

4.2 RQ2: Solutions

The implemented solutions to the requirements will be presented in this section and will include how the solutions work, the labels from the requirements tables will also be included in the text where they are being fulfilled by the solution. The process identified at the beginning of the project, registration phase, setup phase, voting phase, and result phase, will be used to present the results. The overall setup of the voting process will be both in a smart contract on the blockchain (on-chain) and off the blockchain (off-chain) with code in TypeScript. The solution is not a fully working process, but smaller parts that together make a functioning solution. It is built up with one smart contract and three TypeScript functions.

When the elections are active, it is important that it is only possible to call specific functions when the right phase is active, for instance voting only during the voting phase. This was solved by including different phases into the smart contract so that specific functions can only be called during the correct phase. This functionality is implemented with an Enumerator that defines the phases, a modifier that is used within functions to require a specific phase, and a function that can change the phase, see Figure 4.1.

```
1     // Setting up the possible phases
2     enum Phase {Setup, Vote, Result}
3     Phase public state;
4
5     // Modifer to validate phase
6     modifier validPhase(Phase reqPhase) {
7         require(state == reqPhase);
8         _;
9     }
10
11    // Function to increase state (change to next phase)
12    function changeState(Phase x) public onlyChair() {
13        require (x > state);
14
15        state = x;
16    }
```

Figure 4.1: Phase functionality in the smart contract.

The smart contract also includes a modifier called `onlyChair()` which is used for functions only the organizers (chairperson) should be able to call, for example, `changeState()`. This enables the chairperson to decide for example when the voting phase begins (**Prevoting**). The chairperson is the one who deployed the contract, and there will be one chairperson per constituency. The reason for this is that each constituency has its own instance of a deployed smart contract (**Applicable for multiple elections**), which will ensure that each voter is connected to a con-

stituency (**Connect vote to constituency**) and that the votes can be counted on the constituency level (**Constituency level results**).

4.2.1 Registration phase

The first phase in the voting process, the registration phase, will not have anything to do with the features enabled by the blockchain. This phase will look similar to what it is today in the traditional voting system. This process will be handled by Skatteverket, The Swedish Tax Agency since they have records on all citizens and eligible voters. Instead of sending physical vote cards to all voters as in the traditional system, a digital system will be used to mark who has voted and who has not. The solution also requires the voter registry to include the decentralized identities, the address of the blockchain account, of the voters. The voter registry does not necessarily need to know what address is connected to which voter, but they need to know the decentralized identities of a constituency. The reason for this is to prepare the verification mechanism in the smart contract so that only eligible voters can access the voting function. Since the verification mechanism is dependent on the voter registry, the complete list of eligible voters has to be completed before the voting phase starts.

4.2.2 Setup phase

The second phase, the setup phase, will not be utilizing any of the features that are specific to the blockchain either, but it will deploy the smart contracts and add data to them. This phase will handle the registration of parties and candidates, as well as add the verification mechanism to verify the eligible voters to the smart contract.

The registration of parties and candidates is done by Valmyndigheten, The Swedish Election Authority. When a party or candidate is being registered they will be mapped off-chain to a specific integer, that is specific for the party or candidate, see Figure 4.2. This means that if a user wants to vote for Party B and Candidate 3B, the vote would be 2 for the party and 3 for the candidate and if a user wants to vote for Party C and Candidate 1C, the vote would be 3 for the party and 1 for the candidate.

The second part of the voting setup phase is adding the verification mechanism to verify the eligible voters to the smart contract (**Voting only for eligible users**). This mechanism is using a Merkle tree, also known as a hash tree, where each node is labeled with a cryptographic hash of data, in this case, the decentralized identity of a voter. The tree is generated with a function in TypeScript, see Figure 4.3. The `allowList` array in the code includes the addresses of all eligible voters, these addresses are being hashed with the hashing function Keccak256 and then turned into a Merkle tree. From the validation interviews, one interviewee really liked the verification mechanism.

Identifier	Party
1	Party A
2	Party B
3	Party C

Identifier	Candidate Party A
1	Candidate 1A
2	Candidate 2A
3	Candidate 3A

Identifier	Candidate Party B
1	Candidate 1B
2	Candidate 2B
3	Candidate 3B

Identifier	Candidate Party C
1	Candidate 1C
2	Candidate 2C
3	Candidate 3C

Figure 4.2: Registered parties and candidates example.

"Beautiful, beautiful. I think you've used it very nicely"

```
// Generate tree from eligible users (example addresses provided)
let allowList = [
        '0xb794f5ea0ba39494ce839613fffba74279579268',
        '0xb794f5ea0ba39494ce839613fffba74279579267',
        '0xb794f5ea0ba39494ce839613fffba74279579266',
        '0xb794f5ea0ba39494ce839613fffba74279579265'
    ]

const leaves = allowList.map((address) => keccak256(address))
const tree = new MerkleTree(leaves, keccak256, {sortPairs: true})
```

Figure 4.3: Generate Merkle tree in TypeScript.

In order for the verification mechanism to work, the smart contract has to know the root hash of the Merkle tree. When the tree is generated, the root hash can then be gathered and stored in the smart contract. By calling `tree.getHexRoot()` in the TypeScript file, the root of the Merkle tree can be collected. The next step would be to add it to the smart contract, but before that, the smart contract needs to be deployed onto the network. When that is done, the function `setMerkleTreeRootHash(bytes32 rootHash)` can be called to add the root. This function can add or change the root hash in the smart contract and is only callable in the setup phase due to the modifier `validPhase(Setup)`, see Figure 4.4. This function is also exclusively callable by the chairperson and is specified with the modifier `onlyChair()`. How the verification mechanism works will be described during the voting phase.

```
// Function to set Merkle tree root during the setup phase
function setMerkleTreeRootHash(bytes32 rootHash) public onlyChair()
        validPhase(Phase.Setup) {
    merkleRootHash = rootHash;
}
```

Figure 4.4: Set Merkle tree root hash in the smart contract.

4.2.3 Voting phase

The third part of the process is the voting phase, which will take part only on-chain with functionality from a smart contract. The voting part of the smart contract was built with one main function, `Vote(bytes32[] calldata _merkleProof, bytes calldata encryptedVote)`, see Figure 4.5. This function has two responsibilities, call the function `checkValidity(bytes32[] calldata _merkleProof)` to validate the voter and to post the transaction to the chain.

```
// Function to vote during the voting phase
function Vote(bytes32[] calldata _merkleProof,
        bytes calldata encryptedVote) public validPhase(Phase.Vote) {
    checkValidity(_merkleProof);

    Voters[numberOfVotes].Address = msg.sender;
    Voters[numberOfVotes].Vote = encryptedVote;
    numberOfVotes++;
}
```

Figure 4.5: Function to vote in the smart contract.

The parameters that the function needs are the Merkle proof for the verification and the encrypted vote. The function will first handle the verification, where the root hash added in the setup phase will be compared to a newly generated root hash from the Merkle proof in the smart contract. The Merkle proof is gathered off-chain by calling `getHexProof(hashedAddress)` on the generated tree from the setup phase. If the generated hash is the same as the hash added to the smart contract, the user has been verified and is allowed to vote, see Figure 4.6.

```
// Function to validate eligible voters
function checkValidity(bytes32[] calldata _merkleProof) private view {
    bytes32 leafToCheck = keccak256(abi.encodePacked(msg.sender));
    require(MerkleProof.verify(_merkleProof, merkleRootHash,
        leafToCheck), "Incorrect proof");
}
```

Figure 4.6: Function to check validity in the smart contract.

The second parameter that the vote function needs is the encrypted vote. The

encryption will be done off-chain with a public-key encryption scheme and the encrypted vote will be passed to the function. Rivest-Shamir-Adleman (RSA) is the encryption technique used in this solution, and the encryption was calculated by a web-based solution. The voting technique described in the setup phase will be used, where the votes for the party and candidate are represented by an integer in a comma-separated structure. The same public key is used to encrypt all votes, therefore a vote-specific identifier is included in each vote to make sure it is unique. If this is not included, each encrypted vote for the same party and candidate would look the same. The first integer in the vote is the vote-specific identifier, the second is the party and the third is the candidate, see Figure 4.7.

Figure 4.7: Encryption of vote.

The reason for passing an already encrypted vote to the blockchain, and not doing the encryption on-chain is because of transparency. Since blockchain is open to anyone and each transaction can be tracked, the arguments sent to a function are public. By doing the encryption off-chain, the argument will be the encrypted vote and no one will be able to decrypt it until the voting phase is over.

To prevent the encrypted vote to take up a lot of space on the blockchain, which would increase the transaction costs, the encrypted votes are turned into a series of bytes instead of a string before calling the vote function, see Figure 4.8. The conversion is done off-chain, see Figure 4.9, to minimize the computations in the smart contract, which also lowers the transaction costs. This design decision was commented on in one of the validation interviews, where the interviewee mentioned:

> *"Just fantastic, fantastic. I like bytes better than strings, it's expandable. I know that bytes are finite."*

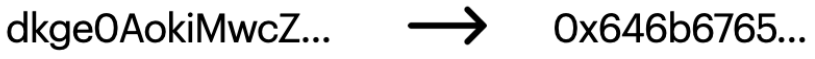

Figure 4.8: Converting vote into hexadecimal bytes.

```
// Generate bytes in hexadecimal form from string
function generateHexFromString(inputStr: string) {
    var result = '';
    for (var i=0; i<inputStr.length; i++) {
      result += inputStr.charCodeAt(i).toString(16);
    }

    return result;
}
```

Figure 4.9: Generate hexadecimal bytes from a string.

When the verification is completed, the encrypted vote can be stored in the smart contract. The vote and the address of the voter are stored in a struct, which can be seen as a class object in object-oriented programming. This struct is mapped to an integer that is the vote number, see Figure 4.10. This structure was implemented so the smart contract can keep track of all votes, and gather them all by looping through all votes in the end to get the result. The voter can do multiple transactions to change their vote (**Revoting**). The reason for storing the address together with the vote is in case of a revote, which enables the latest vote from the same address to be counted off-chain.

```
    // Amount of votes submitted
    uint256 numberOfVotes = 0;

    // Struct that includes the address of the voter and the encrypted vote
    struct Voter {
        address Address;
        bytes Vote;
    }

    // Map to keep track of address and encrypted vote for each vote
    mapping(uint => Voter) Voters;
```

Figure 4.10: Functionality to store votes on-chain.

4.2.4 Result phase

The last and final phase of the voting system is the result phase. This phase starts when the voting phase has been completed, which means that no votes can be placed by this point (**Count votes after election**). The responsibility of this phase is to return the encrypted votes and the corresponding addresses (**See all votes** and **See personal vote**). Since the smart contract stores the number of votes posted, we can loop through all posted votes, store them in arrays, and return the arrays, see Figure 4.11.

```
    // Function to get results during the result phase, returns list
    // of votes and addresses
    function getResult() public validPhase(Phase.Result) view returns
            (bytes[] memory, address[] memory) {
        bytes[] memory byteVotes = new bytes[](numberOfVotes);
        address[] memory voterAddress = new address[](numberOfVotes);

        for(uint256 i=0; i < numberOfVotes; i++) {
            byteVotes[i] = Voters[i].Vote;
            voterAddress[i] = Voters[i].Address;
        }

        return (byteVotes, voterAddress);
    }
```

Figure 4.11: Function to return the result arrays.

As described in the voting phase, all encrypted votes posted to the blockchain have been converted into a series of bytes. This means that the returned values are also a series of bytes, so in order to decrypt them they need to be converted into strings again to get the cipher text, see Figure 4.12. The private key can be used to decrypt the vote when the cipher text has been gathered.

```
// Generate string from bytes in hexadecimal form
function generateStringFromHex(inputHex: string) {
    var bytes: Array<number> = new Array(inputHex.length / 2);
    for (let i = 0; i !== bytes.length; i++) {
        bytes[i] = parseInt(inputHex.substr(i * 2, 2), 16);
    }

    var result = String.fromCharCode.apply(String, bytes);

    return result;
}
```

Figure 4.12: Generate string from bytes.

In order to keep the private key anonymous during the voting phase, it is split among stakeholders, for example, the parties or some unbiased entity, and all parts are needed to decrypt the message, see Figure 4.13. This will ensure that no one can decrypt the votes during the voting phase and that all stakeholders are needed to decrypt them, no single stakeholder can do it by themselves. When the result phase starts, the private key is public to the citizens to open up the possibility count at home.
When the cipher texts have been gathered they can be looped through off-chain, and decrypted to get the actual parties and candidates that the citizens have voted

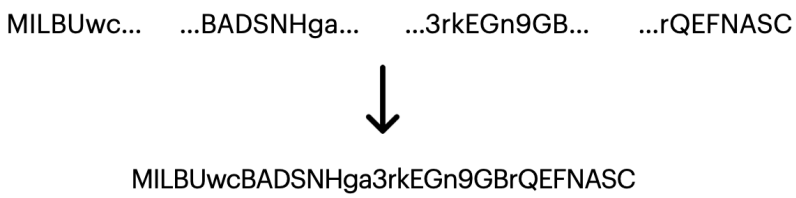

Figure 4.13: Split private key.

for. A key aspect to keep in mind during this process is to remove all votes except the last one for each voter, so multiple votes for each user are not counted.

4.3 RQ3: Requirements fulfilled by solution

The solution above does not fulfill all the requirements for various reasons. Some have been ignored because they are not part of the blockchain solution and instead connected to off-chain code which has not been put emphasis towards. Other requirements can have an effect on each other which means two requirements can be complicated to implement simultaneously. Below is Table 4.6 and Table 4.7 containing each requirement from Section 4.1. They contain one column stating if the requirement was fulfilled or not, and another describing the reasoning for it.

ID	Argument for (non-)fulfillment	Fulfilled
FR1	**Digital voter registry** The digital voter registry is only a prerequisite for the system and was not included in the solution above since it is not blockchain related. Such a system would be supplied by The Swedish Tax Agency.	
FR2	**Date for registering parties** The registration of parties would be handled by a different system. This is not something that was looked into, since the blockchain code will not be affected by it.	
FR3	**Date for registering candidates** The registration of candidates would be handled by a different system. This is not something that was looked into, since the blockchain code will not be affected by it.	
FR4	**Voting only for eligible users** Users will need to prove they are eligible voters by confirming that they are part of the Merkle tree.	Yes
FR5	**Vote from anywhere** The system developed can be used from anywhere. A downloadable front-end application from any client can be built to support this.	Yes

FR6	**Prevoting** The system developed can be available for as long as needed. The smart contract owner can decide when the voting and result phase begins.	Yes
FR7	**Revoting** Revoting is possible since there is no limit to how many times a user can vote.	Yes
FR8	**Marking invalid votes** Invalid vote markers have not been implemented in the smart contract. This will be handled off-chain.	
FR9	**Connect vote to constituency** The smart contract is duplicable and can therefore be divided into any amount of constituencies.	Yes
FR10	**Constituency level results** The smart contract will only return the votes it received. This allows for the results to be presented for each constituency.	Yes
FR11	**Count votes after election** The votes are only decrypted after the election is over and the results phase has begun. Nobody can read the results in real time during the voting phase.	Yes
FR12	**One vote per person** The smart contract will not consider if people have voted once or multiple times. This check will be handled off-chain by counting the number of times an address appears and then taking the latest transaction from this address.	
FR13	**Last vote counted** The smart contract will not check if the users' last vote is counted. This check will be handled off-chain as stated in FR12.	
FR14	**See personal vote** The users will be able to see their own votes after the voting phase is over by decrypting the vote with the private key and checking the vote coupled with their address.	Yes
FR15	**See all votes** The users will be able to see all the votes by decrypting the blockchain data after the voting phase when the private key is accessible to the public.	Yes
FR16	**Untraceable votes** Votes are still traceable since it is possible to view the addresses and the votes coupled with them.	

Table 4.6: Description and fulfillment of the functional requirements.

ID	Argument for (non-)fulfillment	Fulfilled
NFR1	**Transaction speed** The Ethereum blockchain does not currently support the number of transactions per day needed for the system.	
NFR2	**Encrypt sensitive data** The votes stored on the blockchain are encrypted. However, the addresses are not encrypted.	
NFR3	**Safe from data tampering** When the data is added to the blockchain, it can not be tampered with as long as no entity gets control of 50% percent of the network. The smart contract code does not contain any update or delete functions, meaning it is only possible to add information.	Yes
NFR4	**Applicable for multiple elections** The smart contract is duplicable and can be used for multiple elections.	Yes
NFR5	**Disallow invalid data** The smart contract accepts any message and the off-chain code should have restrictions to not allow invalid input data.	
NFR6	**High uptime** This is fulfilled by looking at historical data for Ethereum	Yes

Table 4.7: Description and fulfillment of the non-functional requirements.

5
Discussion

This section aims to answer the research questions RQ1-3. These questions will be answered individually with a description of what the goal with each question was together with an answer to the question based on the case study. Additionally, threats to the validity and limitations of the research are brought up in this section. Lastly, future research in regard to blockchain development and our results are discussed.

5.1 RQ1: How can the problem space for a blockchain system be identified?

The first part of the project focused on identifying the problem space for a blockchain-based project, and we decided to answer this question by using requirements engineering. This question was chosen to understand if requirements engineering can be used in the field of blockchain.

The process of requirements engineering was not changed compared to how it is used in traditional software engineering and the techniques used for elicitation, specification, and verification were standard in the field. By the use of requirements elicitation and requirements specification, we found that the identification of functional requirements could easily be specified. A set of requirements were created from the interviews, and those need to be fulfilled in order for the system to function properly. Although, since blockchain is a highly secure technology to use, the functional requirements had a lot of security built into them. For example, the result can only be gathered after the election. This is of course related to the case we decided to use, but the pattern of having security built into the functional requirements should be the case for most blockchain applications since that is the area where the technology is most useful. The decision to use blockchain as the technology therefore had an impact on how the requirements were conducted. Lausen [51] mentioned the importance of focusing on the problem space rather than the solution space. From this thesis, we understand the importance of separating them, but we have also identified that when working with blockchain there is a connection that has to be addressed in the requirement specification.

The biggest problem relating to identifying the problem space was the non-functional requirements. There are some quality factors that can be specified and fulfilled for a blockchain solution, such as maintainability and correctness since these are depen-

dent on how the smart contract was developed. But there are some quality factors that the developer can not control, and these are reliability and efficiency. Since a blockchain is a decentralized network of nodes there is a very limited possibility to increase the reliability or efficiency, which is something that is easier to affect in traditional SE. The reliability attribute is related to uptime, which in practice is phenomenal for a blockchain system. As described in Section 2.1, the biggest blockchain networks have had minimal instances of downtime [29] [30]. Although, this is not something that is ensured and there is nothing a developer can do to prevent a blockchain to have downtime because of its decentralized nature. When it comes to efficiency, it is also nothing that can be ensured. The smart contract can be developed in an efficient way, which reduces the transaction costs and resources needed to publish transactions to the blockchain, but the actual transaction time and the number of transactions per second can not be modified unless switching to another blockchain or by using a second layer solution [28].

When specifying the quality grid for the non-functional requirements, we also realized that many of the quality factors are tilted towards the extreme points of *Critical* or *Ignore*. Under the category operation, usability is set as *Ignore* in our case. The reason is that our case only looked at the blockchain part, where usability is related to user interface and user experience. If the case would have specified the whole process usability would have been categorized as *Critical* as well. That would mean that all attributes under the category operation would be set as *Critical*. This is dependent on the case we decide to use since it is related to democracy and it would be a tragedy if something would go wrong from a societal perspective, which is the reason for many attributes being set as *Critical*.

As stated above, the quality factors are specified with high importance since it is related to democracy. Additionally, decentralization, transparency, and data immutability are needed for a digital voting system [38, 39, 21], but this is not exclusive for voting. There are other applications with the same characteristics as voting that need these factors as well, such as government-owned applications, currencies, and ownership of assets since if these applications fail, it could lead to tremendous outcomes. This shows that blockchain applications are useful when almost no margin of error is allowed. The attributes could be set at less important than *Critical* as well, but then the question related to if blockchain is the technology to be used comes in. If there are no critical reasons for security and correctness, then a centralized database is more than enough, the same goes for reliability as well. An application that does not necessarily need the functionality of the blockchain actually benefits from not using it.

The quality factors under revision and transition were decided to be set to *As usual*, with the exception of reusability in this project, and the main reason for this is that they are not particularly important compared to the other factors. Each constituency should have its own deployed instance of the smart contract and there are over 6000 constituencies in Sweden, all consisting of different eligible voters to be verified. All of these constituencies should also be able to count their own votes, it

was therefore set as *Critical*. For blockchain projects striving for high security and transparency, the factors under revision and transition will differ depending on the project and application. Some projects might need more interoperability and others more flexibility, so non of these factors will particularly have an impact by using blockchain.

In summary, the answer to RQ1 is that the problem space can be identified with requirements engineering. The functional requirements can easily be specified if the right elicitation techniques are being used, although the requirements have a connection to blockchain and are specified with blockchain in mind, with security being built in. The non-functional requirements on the other hand generate problems. Some of the quality factors can not be modified or controlled, which leads to the non-functional requirements being hard to specify.

5.2 RQ2: How can the solution space be specified with blockchain technology?

The second part of the project focused on identifying the solution space by implementing a working solution from the problem space. This process was done with the programming language Solidity and the blockchain Ethereum.

The solution created in this case is a fully working process and could in practice be used if the correct integration between the parts is implemented. Although, the solution space is not fully covering the entire problem space. There are some requirements that are not fulfilled, such as FR16 (**Untraceable votes**). The reason for this is related to public addresses since anyone can see what transactions a user has published if the address can be viewed by anyone. This is specifically a problem if a voter is a well-known identity on the blockchain doing other transactions such as currency transfers or calling other smart contracts. If the receiver for those transactions knows the identity behind the sender's address, then the receiver can easily see what they voted for. This is not a problem specific to our case, but a problem for blockchain overall, although it can also be seen as an advantage of blockchain in other areas. The problem arises for applications where private information is transferred that should not be open to the public. This problem has a deep connection to the problem space, which also leads to issues with identifying the problem and solution space separately.

During the implementation phase, we also realized that it is impossible to develop the whole process on the blockchain, and regular software engineering is needed for support. All parts except for a user interface could in practice be built on the blockchain, but that would convolute the end product. Avoiding blockchain in scenarios where it is not vital for the product is necessary, and it should not be implemented only for the buzzword aspect. It is also important to understand that the blockchain should not be used as a database where it does not have to. It should only be used in cases where data immutability, decentralization, and security are critical require-

ments, like our case study. With this being said, information and functionalities that can be handled off-chain should be handled off-chain.

One example where blockchain should not be used on its own and is better with support from off-chain code would be sending classified information, in our case that would be the votes. In theory, one might think that sending information to a smart contract that encrypts the data would be safe, but it's not. Since all transactions have to be verified by the nodes, those nodes can access the data sent to the smart contract. Also, the arguments sent to a smart contract can be seen in the transaction history. The problem with integrating off-chain functionality is the security of the system, even though the security on the blockchain is great, that is not necessarily the case for off-chain systems. To ensure that the blockchain system is secure, we therefore also have to ensure that the off-chain system is secure, which stated in the introduction is a wicked problem.

From the findings in Section 4.2, the product we developed included parts both on-chain and off-chain, which was a necessary decision for the quality, but also to only use the blockchain when it is needed. By analyzing the scenario, we identified that the only parts benefiting from using blockchain are validating the voters, recording votes, and gathering the results.

An issue described in Section 5.1 was if the blockchain is efficient enough to handle the workload. This is an issue today and could be solved by a future update for Ethereum [60]. Although, when developing a blockchain application we can not only consider our own usage since it is a public network open for anyone and can only handle a specific load. Imagine a lot of our daily tasks start using the blockchain instead of centralized databases. We would then need to make sure that all of those tasks together with the new application can handle the load together, otherwise, our society could suffer from applications not working. Another take on this could also be if someone wants to sabotage the system. Then they could put the network under a lot of stress and slow the processing of other transactions which also would increase the transaction costs. In this case, that could lead to not all votes being processed until multiple days after the election.

Due to the concerns above, some of the requirements are therefore more complex to solve and find a solution for, although this is not unique to blockchain development. There could be requirements for a regular software engineering project that is equally difficult to solve, the difference is the process of finding the solution and what aspects to have in mind when solving the problem. The most important thing to have in mind when solution-seeking for a blockchain system is that all data is public, which is an extra complex task when classified information is transmitted.

In relation to democracy, it is crucial that the correct source code is being used and not a modified one that could impact the election. If a centralized system is used, there is no possibility to ensure that the right source code is being used, the system might be open source but the code running could be modified. In Section 1

we introduced this as a wicked problem in SE, which is still the case for blockchain. When a smart contract is deployed, the code running is not visible to anyone but the posted transactions from the smart contract are. So anyone can see what happened to the input data when a function was executed.

In summary, the answer to RQ2 is that the identification of the solution space is difficult when defining which parts should be utilizing the functionality provided by the blockchain, how to solve non-functional requirements, and how to keep the data safe until it reaches the blockchain. The identification of what parts should use the blockchain should be done by specifying the whole scope of the project, and pinpointing what processes need security, immutability, and decentralization. Although, some parts that should utilize these attributes, might not be able to due to the transparency of the blockchain.

5.3 RQ3: How do the problem space and the solution space interact when specifying a blockchain system?

The final research question is related to how the problem space and solution space interact with each other, and what effects they have on each other. This question was chosen to understand if any of the two spaces could change depending on the other, or if there is an easy transition from problem to solution space.

The analysis for answering this question started when the problem space was identified and the development began. We quickly identified that our initial plan for the development changed, and we had to use off-chain functionality more than expected. As answered in Section 5.2, blockchain should only be used when necessary, and our initial idea was to send the vote details to the smart contract and do all of the encryption on-chain. The idea behind it was to minimize the risk of security failure, for instance, malware on the local device recording the votes. It would be a good idea to do it on-chain to prevent this, but because of blockchain's transparency, it is not possible. The votes which have not been encrypted would then be visible in each transaction as the argument. Therefore it had to be done off-chain, which led to new requirements being specified.

As mentioned in Section 1, security in SE is a wicked problem. By connecting this to the issue above we can see that the wicked problem in this case is not related to the blockchain part of the application, but rather the off-chain code. This is also something that was identified during the thesis, the wicked problem of security is not really a problem for blockchain, but rather all other integrations that need to be implemented to make the application function as planned.

The connection between problem space and solution space is quite close, and they depend on each other, and specifically when solving one requirement becomes a blocker for another. The system supports revoting, which is good for preventing

extortion, but it makes the addresses public, as described in Section 5.2, and could lead to leaked votes if it's known who owns which address. This could have been solved by using addresses owned by the government to make the transactions, in other words, all transactions are made from the same address. This would however remove the possibility of revoting since there would not be a way to know which votes have been posted by the same voter and which to remove during the counting. Therefore, a change in the problem space would have been necessary to make sure the solution space covers the entire problem space.

The problem above shows how the problem and solution space interact, where solving a problem creates a new one. In this case, we kept both requirements, even if both are not solved to show how they relate to one another. If this would have been the actual development of the system, a change in the problem space would have been needed after exploration in the solution space. Another example where solution space has an impact on the problem space would be the non-functional requirement NFR1 (**Transaction speed**). This requirement impacts the solution space since a blockchain that supports the required load needs to be chosen, on the other hand, choosing a blockchain with a larger TPS could be a case of the scalability trilemma. Using a blockchain with a larger TPS will sacrifice either security or decentralization, in some cases even both. A blockchain that supports a large TPS might also not have the same development support and functionality as one with less TPS, which could lead to the product not being able to be developed on that specific blockchain. In a case like this, a change in the problem space might need to be done to keep the security, decentralization, and functionality of the end product.

Blockchain is a new technology with the development of new features and performance fixes constantly, so the issues relating to finding solutions for problems get less and less challenging with every new update. The problem with a system with constant updates and improvements is that it is hard to specify standards. There are not a lot of standards for how to solve different problems which resulted in the finding of solutions for problems that in practice should have a standard implementation. We noticed this with the user verification, there was no generalized standard which forced us to find a solution. In hindsight, it was a good thing since we pioneered a way of verifying users for blockchain-based voting using the Merkle Tree. But on the other hand, it requires the electoral authorities to store all voter addresses which was not planned initially. In this case, the solution space impacted the problem space and requires the off-chain modules of the project to be more secure but also to store the decentralized identities of all citizens, which is quite a large and difficult requirement to fulfill.

The problem space identified in this project relates to a highly digital society, with requirements such as a digital voter registry. These requirements are not only related to the electoral authority and need support from other authorities, this case is dependent on The Swedish Tax Agency. From one of the validation interviews, one interviewee mentioned a lecture quote from Liisa Past, National Cyber Director in Estonia, where she stated that a country needs a highly technical IT infrastructure

to conduct an electronic election. This statement relates well to the case study in this thesis as well. The connection between the problem and solution space gets stronger when looking into a case dependent on other systems. Furthermore, the problem space gets even bigger since more systems other than the actual voting system need to be specified, and the solution space is dependent on other systems.

In summary, the answer RQ3 is that the market needs more functionality than the technology can provide with today's standards. Therefore, the problem and solution space are tightly coupled, and a small issue in the solution space could have a major impact on the problem space. The problem space is then in need of modification in order for the solution space to fully cover the problem space. By modifying the problem space we can also identify that blockchain can not be used in isolation, and is in need of support from traditional systems.

5.4 Threats to validity

This section discusses the threats to validity of the thesis. When conducting research of this size there are always some threats to the validity, and this section describes the most significant ones. The threats are categorized into construct validity, internal validity, external validity and reliability which has been established by Runeson et al. [55].

5.4.1 Construct validity

The thesis had a relatively low number of conducted interviews in the data collection phase, where only four people were interviewed. Two of the interviewees were key stakeholders from The Swedish Election Authority and knows all details relating to an election. The other two interviewees are well-established in the field of blockchain and know the technology as well as its possibilities. While this gave us the information needed to specify the problem space, as well as ideas for potential solutions, more interviews could have provided more insights and requirements for such a system.

5.4.2 Internal validity

Bias towards our own solution is also a threat to validity since we are the ones who researched how the system should be built and implemented it. Both the problem space and the solution space could potentially have been criticized more if it was produced by someone else.

5.4.3 External validity

The thesis focused on a specific case, blockchain-based voting, with the use of Ethereum as the blockchain. The findings could therefore be less generalizable for other blockchain systems.

5.4.4 Reliability

The system developed focused only on the blockchain parts, and not as much on traditional software engineering. The results could potentially have been different if the whole process with user interface, centralized backend, blockchain, and the bridges between these were developed.

5.5 Limitations

This section discusses the limitations of the study. The thesis focused on providing valuable insights and contributions to the field of blockchain development, but the limitations of the study have to be taken into account.

The thesis focused only on permissionless blockchains, and only touch upon some cases where permissioned blockchains could be used. Although, permissioned blockchains and their capabilities of solving specific problems were not investigated thoroughly.

The permissionless blockchain used for the project was Ethereum and no other chains were researched in detail. Using Ethereum as the blockchain influenced the programming language used for smart contract development. Therefore no other languages or smart contract techniques were investigated.

The thesis did not consider interviews with end-user and the reason is related to the thesis only looking into the backend and blockchain part of the system. End-users would have been valuable if the whole system was taken into account with a user interface, but since that was outside the scope it was decided to not include them. The thesis would also not have benefited from including them, since it is a complicated technology that many people do not know about, and the comments from end-users would have been related to what they want and not what the system need.

5.6 Future research

There is a lot of future research that can be done in the field of blockchain, specifically in the same field as this study. We think it would be interesting for future research to look into frameworks for how non-functional requirements can be fulfilled by researching the hindering parameters of blockchain, for example, consensus protocols, and how they can be changed to increase flexibility and modifiability for application developers.
It would also be interesting for future research to conduct larger research with more interviewees involved, where the whole process from idea to finalized product is taken into account and compare it to our results focusing only on the blockchain part. It would also be interesting to do a similar study where a different case is chosen, for example, a case where the quality attributes under revision and transition are more important, to see if the results are comparable.

Another idea for future research would be to look into standardizations for blockchain development, and how solutions for generic problems could be solved.

6
Conclusion

Specifying and developing a blockchain application is more common now than it has ever been, but there is not a lot of previous work on how to go about such a challenge. The research aimed to identify the problem space and solution space and how they interact within a blockchain system. Interviews and literature reviews along with self-exploration were conducted in this case study on a blockchain-based voting system. The research questions for this thesis are the following:

- How can the problem space for a blockchain system be identified?
- How can the solution space be specified with blockchain technology?
- How do the problem space and the solution space interact when specifying a blockchain system?

The first research question is related to identifying the problem space and it was discovered that requirements engineering, a common technique in software engineering, can be used for blockchain development. The process of identifying the problem space was found to be similar to traditional software engineering. Although, considerations in regard to security had to be incorporated into the specification phase, which was also emphasized in the requirements. The most challenging part to identify was the non-functional requirements because some quality factors such as reliability and efficiency are difficult to manage in a decentralized network. Another aspect of the non-functional requirements which posed challenges is transparency since blockchain inherently consists of transparent records of data. This causes a dilemma in identifying how the privacy requirements should protect sensitive information.

Secondly, the specification of the solution space was implemented using the programming language Solidity on the blockchain Ethereum. A functioning blockchain-based voting system was developed. It was observed however that the solution did not fully cover the entire problem space. An example of this is untraceable votes, and the rationale for this was related to the transparency of blockchain transactions. There were also some non-functional requirements that could not be fulfilled due to the limitations of blockchain in relation to transaction speed and transparency. In addition to this, it was discovered that a blockchain application can not operate by itself and needs off-chain functionally for support. Blockchain should only be used when immutability, decentralization, and security are critical, the application should therefore handle off-chain functionality to procedures where these factors do not need to be satisfied.

The third research question investigated the interaction between the problem and solution space. The thesis identified that there is a connection between the two, and a small change in either space has an impact on the other. As the understanding of the solution space improved, adjustments had to be made to the problem space, and functionality off-chain had to be implemented. Solving problems in the solution space therefore had an impact on the problem space, restricting other problems from being solved. The selection of blockchain also had an impact on both spaces, fulfilling some problems but restricting others. In addition to this, the decision to use blockchain as the technology had an impact on the problem space specification. It was also identified that solving the security problems individually was not impossible, but problems started to arise when a lot of security measures had to be considered at the same time, which impacted the problem space.

In conclusion, this thesis highlights the difficulties of identifying the problem space and specifying the solution space for a highly secure and transparent blockchain-based system. It brings up the relevance of using requirements engineering techniques for blockchain, while at the same time thinking of the challenges and possibilities of blockchain. The thesis also identified the importance of evaluating the use of blockchain and its connection to critical requirements, specifically the requirements relating to security, reliability, efficiency, and decentralization. The need for using off-chain functionality was discovered, but also the trade-off effects introduced of integrating on-chain and off-chain code. A highly secure and transparent blockchain application is therefore a challenge to implement with the state of the art, but the future could bring more straightforward guidance for such systems.

Bibliography

[1] P. Schuh, *Integrating agile development in the real world*. Charles River Media, Inc., 2004.

[2] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan, "On the usage, co-usage and migration of ci/cd tools: A qualitative analysis," *Empirical Software Engineering*, vol. 28, no. 2, p. 52, 2023.

[3] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.

[4] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version control system: A review," *Procedia Computer Science*, vol. 135, pp. 408–415, 2018.

[5] J. Farrelly, "High-profile company data breaches 2023," Online, May 2023, accessed on May 3, 2023. [Online]. Available: https://www.electric.ai/blog/recent-big-company-data-breaches

[6] H. W. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy sciences*, vol. 4, no. 2, pp. 155–169, 1973.

[7] S. Maheshwari and A. Holpuch, "Why countries are trying to ban tiktok," Online, April 2023, accessed on May 3, 2023. [Online]. Available: https://www.nytimes.com/article/tiktok-ban.html

[8] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] (2022, October) The important features of blockchain technology. Accessed on June 15, 2023. [Online]. Available: https://dashtechinc.com/the-important-features-of-blockchain-technology/

[10] "Ethereum whitepaper," Online, Ethereum, accessed on March 22, 2023. [Online]. Available: https://ethereum.org/en/whitepaper/

[11] R. Sheldon. A timeline and history of blockchain technology. Accessed on May 24, 2023. [Online]. Available: https://www.techtarget.com/whatis/feature/A-timeline-and-history-of-blockchain-technology

[12] J. Speakman and M. Washburn. Can blockchain voting strengthen democracy? Accessed on May 24, 2023. [Online]. Available: https://beincrypto.com/blockchain-voting-securing-democracy/

[13] E. van Winkoop, "Intellectual property rights and electronic voting based on blockchain," Online, January 2023. [Online]. Available: https://northsearegion.eu/bling/news/intellectual-property-rights-and-electronic-voting-based-on-blockchain/

[14] E. Piirmets, "How did estonia carry out the world's first mostly online national elections," Online, March 2023, accessed

on May 20, 2023. [Online]. Available: https://e-estonia.com/how-did-estonia-carry-out-the-worlds-first-mostly-online-national-elections/

[15] D. Palmer, "Moscow blockchain voting system 'completely insecure,' says researcher," August 2019, accessed on January 11, 2023. [Online]. Available: https://www.coindesk.com/markets/2019/08/16/moscow-blockchain-voting-system-completely-insecure-says-researcher/

[16] L. Moore and N. Sawhney, "Under the hood: The west virginia mobile voting pilot," Boston, MA. 02109, 2019.

[17] T. Haarseim. Blockchain voting: Decentralised, transparent elections? Accessed on May 24, 2023. [Online]. Available: https://democracy-technologies.org/voting/blockchain-voting-decentralised-transparent-elections/

[18] S. Schumacker and A. Connaughton, "From voter registration to mail-in ballots, how do countries around the world run their elections?" October 2020. [Online]. Available: https://www.pewresearch.org/short-reads/2020/10/30/from-voter-registration-to-mail-in-ballots-how-do-countries-around-the-world-run-their-election

[19] K. Moreland. (2019, October) What is blockchain? Accessed on April 21, 2023. [Online]. Available: https://www.ledger.com/academy/blockchain/what-is-blockchain

[20] G. Habib, S. Sharma, S. Ibrahim, I. Ahmad, S. Qureshi, and M. Ishfaq, "Blockchain technology: Benefits, challenges, applications, and integration of blockchain technology with cloud computing," *Future Internet*, vol. 14, no. 11, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/11/341

[21] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.", 2014.

[22] P. Vadapalli. (2022, September) What is blockchain transaction? how does it work? Accessed on May 22, 2023. [Online]. Available: https://www.upgrad.com/blog/what-is-blockchain-transaction/

[23] B. Becher. (2022, September) What are blockchain nodes and how do they work? Accessed on April 23, 2023. [Online]. Available: https://builtin.com/blockchain/blockchain-node

[24] A. S. Rajasekaran, M. Azees, and F. Al-Turjman, "A comprehensive survey on blockchain technology," *Sustainable Energy Technologies and Assessments*, vol. 52, p. 102039, 2022.

[25] OriginStamp. How many blocks are in a blockchain? Accessed on May 26, 2023. [Online]. Available: https://originstamp.com/blog/how-many-blocks-are-in-a-blockchain/

[26] B. Academy. Transactions per second (tps). Accessed on May 31, 2023. [Online]. Available: https://academy.binance.com/en/glossary/transactions-per-second-tps

[27] W. Masa. (2022, August) How many transactions per second ethereum: What is the tps for eth? Accessed on May 31, 2023. [Online]. Available: https://bitkan.com/learn/how-many-transactions-per-second-ethereum-what-is-the-tps-for-eth-2698

[28] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE Access*, vol. 8, pp. 125 244–125 262, 2020.

[29] Ethereum. What is ethereum?: The foundation for our digital future. Accessed on May 28, 2023. [Online]. Available: https://ethereum.org/en/what-is-ethereum/

[30] Bitcoin uptime tracker. Accessed on June 15, 2023. [Online]. Available: https://buybitcoinworldwide.com/bitcoin-uptime/

[31] T. K. Sharma. (2022, November) Permissioned and permissionless blockchains: A comprehensive guide. Accessed on May 27, 2023. [Online]. Available: https://www.blockchain-council.org/blockchain/permissioned-and-permissionless-blockchains-a-comprehensive-guide/

[32] B. Academy. (2023, March) What are permissioned and permissionless blockchains? Accessed on May 27, 2023. [Online]. Available: https://academy.binance.com/en/articles/what-are-permissioned-and-permissionless-blockchains

[33] V. Buterin. (2021, April) Why sharding is great: demystifying the technical properties. Accessed on May 22, 2023. [Online]. Available: https://vitalik.ca/general/2021/04/07/sharding.html

[34] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[35] R. Zhang and W. K. V. Chan, "Evaluation of energy consumption in blockchains with proof of work and proof of stake," in *Journal of Physics: Conference Series*, vol. 1584, no. 1. IOP Publishing, 2020, p. 012023.

[36] P. R. Nair and D. R. Dorai, "Evaluation of performance and security of proof of work and proof of stake using blockchain," in *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*. IEEE, 2021, pp. 279–283.

[37] Y. Gao and H. Nobuhara, "A proof of stake sharding protocol for scalable blockchains," *Proceedings of the Asia-Pacific Advanced Network*, vol. 44, no. 1, pp. 13–16, 2017.

[38] K. Curran, "E-voting on the blockchain," *The Journal of the British Blockchain Association*, vol. 1, no. 2, 2018.

[39] T. Moura and A. Gomes, "Blockchain voting and its effects on election transparency and voter confidence," in *Proceedings of the 18th annual international conference on digital government research*, 2017, pp. 574–575.

[40] S. Shanaev, A. Shuraeva, M. Vasenin, and M. Kuznetsov, "Cryptocurrency value and 51% attacks: evidence from event studies," *The Journal of Alternative Investments*, vol. 22, no. 3, pp. 65–77, 2019.

[41] Ethereum. (2023, May) Introduction to smart contracts. Accessed on May 31, 2023. [Online]. Available: https://ethereum.org/en/smart-contracts/

[42] P. Wackerow. (2022, September) Introduction to smart contracts. Accessed on May 31, 2023. [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/

[43] K. Richards. (2021, September) Cryptography. Accessed on May 23, 2023. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/cryptography

[44] IBM. (2021, March) Hashing functions. Accessed on May 23, 2023. [Online]. Available: https://www.ibm.com/docs/en/psfa/7.2.1?topic=toolkit-hashing-functions

[45] A. Katz, A. Ng, P. Bourg, C. Williams, E. Ross, J. Khim, and A. Kau. Rsa encryption. Accessed on May 23, 2023. [Online]. Available: https://brilliant.org/wiki/rsa-encryption/

[46] A. Chumbley, K. Moore, and J. Khim. Merkle tree. Accessed on May 23, 2023. [Online]. Available: https://brilliant.org/wiki/merkle-tree/#citation-1

[47] D. Apostolou. (2021, December) Merkle proofs for offline data integrity. Accessed on May 23, 2023. [Online]. Available: https://ethereum.org/en/developers/tutorials/merkle-proofs-for-offline-data-integrity/

[48] F. Kochan. (2023, March) How to use keccak256 hash function with solidity. Accessed on May 23, 2023. [Online]. Available: https://www.quicknode.com/guides/ethereum-development/smart-contracts/how-to-use-keccak256-with-solidity/

[49] Wikipedia. Merkle tree. Accessed on May 23, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Merkle_tree

[50] S. T. Alvi, M. N. Uddin, L. Islam, and S. Ahamed, "Dvtchain: A blockchain-based decentralized mechanism to ensure the security of digital voting system," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 6855–6871, 2022.

[51] S. Lauesen, *Software requirements: Styles and teechniques*, 1st ed. Pearson Education Limited, 2002.

[52] L. A. Macaulay, *Requirements Engineering*, 1st ed. Springer-Verlag London Limited, 1996.

[53] G. Krüger and C. Lane. (2023, January) How to write a software requirements specification (srs document). Accessed on April 23, 2023. [Online]. Available: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document

[54] M. Kassab, "The changing landscape of requirements engineering practices over the past decade," 08 2015.

[55] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research In Software Engineering: Guidelines and Examples*, 1st ed. John Wiley Sons, Inc., 2012.

[56] Solidity. Accessed on April 24, 2023. [Online]. Available: https://soliditylang.org/

[57] Remix project: Jump into web3. Accessed on April 24, 2023. [Online]. Available: https://remix-project.org/

[58] J. Caulfield, "How to Do Thematic Analysis | Step-by-Step Guide 038; Examples," *Scribbr*, 11 2022. [Online]. Available: https://www.scribbr.com/methodology/thematic-analysis/

[59] SCB, "Valdeltagande i sverige," Online, February 2023, accessed on May 20, 2023. [Online]. Available: https://www.scb.se/hitta-statistik/sverige-i-siffror/manniskorna-i-sverige/valdeltagande-i-sverige/

[60] Ethereum. Ethereum roadmap. Accessed on May 28, 2023. [Online]. Available: https://ethereum.org/en/roadmap/

A

Appendix 1 - Solidity smart contract

```
1
2  //SPDX-License-Identifier: UNLICENSED
3
4  pragma solidity 0.8.19;
5
6  import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";
7
8  contract VotingPhase {
9
10     // Address of the smart contract deployer
11     address chairperson;
12
13     // Setting up the possible phases
14     enum Phase {Setup, Vote, Result}
15     Phase public state;
16
17     // Merkle root hash for verification
18     bytes32 public merkleRootHash;
19
20     // Amount of votes submitted
21     uint256 numberOfVotes = 0;
22
23     // Struct that includes the address of the voter and the encrypted vote
24     struct Voter {
25         address Address;
26         bytes Vote;
27     }
28
29     // Map to keep track of address and encrypted vote for each number of vote
30     mapping(uint => Voter) Voters;
31
32     // Modifer to validate phase
33     modifier validPhase(Phase reqPhase) {
34         require(state == reqPhase);
35         _;
```

```
36      }
37
38      // Modifier to validate chairperson
39      modifier onlyChair() {
40          require(msg.sender == chairperson);
41          _;
42      }
43
44      // Constructor to set chairperson and the initial state
45      constructor () {
46          chairperson = msg.sender;
47          state = Phase.Setup;
48      }
49
50      // Function to increase state (change to next phase)
51      function changeState(Phase x) public onlyChair() {
52          require (x > state);
53
54          state = x;
55      }
56
57      // Function to set Merkle tree root during setup phase
58      function setMerkleTreeRootHash(bytes32 rootHash) public onlyChair()
59              validPhase(Phase.Setup) {
60          merkleRootHash = rootHash;
61      }
62
63      // Function to vote during the voting phase
64      function Vote(bytes32[] calldata _merkleProof, bytes calldata encryptedVote)
65              public validPhase(Phase.Vote) {
66          checkValidity(_merkleProof);
67
68          Voters[numberOfVotes].Address = msg.sender;
69          Voters[numberOfVotes].Vote = encryptedVote;
70          numberOfVotes++;
71      }
72
73      // Function to validate eligible voters
74      function checkValidity(bytes32[] calldata _merkleProof) private view {
75          bytes32 leafToCheck = keccak256(abi.encodePacked(msg.sender));
76          require(MerkleProof.verify(_merkleProof, merkleRootHash, leafToCheck),
77              "Incorrect proof");
78      }
79
80      // Function to get results during the result phase, returns list of
81      // votes and addresses
```

```
82      function getResult() public validPhase(Phase.Result) view returns
83              (bytes[] memory, address[] memory) {
84          bytes[] memory byteVotes = new bytes[](numberOfVotes);
85          address[] memory voterAddress = new address[](numberOfVotes);
86
87          for(uint256 i=0; i < numberOfVotes; i++) {
88              byteVotes[i] = Voters[i].Vote;
89              voterAddress[i] = Voters[i].Address;
90          }
91
92          return (byteVotes, voterAddress);
93      }
94  }
95
```

B

Appendix 2 - TypeScript code

```typescript
// Generate tree from eligible users (example addresses provided)
async function generateTree(): Promise<MerkleTree> {
    let allowList = [
        '0xb794f5ea0ba39494ce839613fffba74279579268',
        '0xb794f5ea0ba39494ce839613fffba74279579267',
        '0xb794f5ea0ba39494ce839613fffba74279579266',
        '0xb794f5ea0ba39494ce839613fffba74279579265'
    ]

    const leaves = allowList.map((address) => keccak256(address))
    const tree = new MerkleTree(leaves, keccak256, {sortPairs: true})

    return tree
}

// Generate bytes in hexadecimal form from string
function generateHexFromString(inputStr: string) {
    var result = '';
    for (var i=0; i<inputStr.length; i++) {
      result += inputStr.charCodeAt(i).toString(16);
    }

    return result;
}

// Generate string from bytes in hexadecimal form
function generateStringFromHex(inputHex: string) {
    var bytes: Array<number> = new Array(inputHex.length / 2);
    for (let i = 0; i !== bytes.length; i++) {
        bytes[i] = parseInt(inputHex.substr(i * 2, 2), 16);
    }

    var result = String.fromCharCode.apply(String, bytes);

    return result;
}
```

C

Appendix 3 - Interview questions

Introduction

1. Give a quick summary of our work
2. Describe the phases, ask if that sounds good, and if there are any other phases that should be discussed that we have missed.
3. Do you think these phases are valid or do you see it from another point of view?

Phase 1 - Registration phase

1. How does the registration process work today?
2. What are the current requirements for registering a voter?
    1. Are there any requirements for personal information?
    2. How can the current system ensure that only eligible voters are registered?
3. How does the current system prevent duplicate registrations?
4. Are these requirements based on a law?
5. How do you think the registration phase could look for a digital election?

Phase 2 - Setup phase

1. What are the preparations for an election today? (printing ballots etc)
2. How is the list of possible parties to vote for created? And where can it be found?
3. How do you think the voting setup phase could look for a digital election?
4. How should all the information regarding the parties and candidates be stored?

Phase 3 - Voting phase

1. Do you believe that we should be able to vote from home or should we still be needed to go to a voting booth?
2. How do you ensure that one person hasn't voted multiple times?
3. What security measures should be taken to ensure authenticity and accuracy?
    1. How does the current system prevent voter impersonation?
4. How can a blockchain-based voting system prevent vote manipulation, e.g. coercion, or vote buying?

5. How should votes be recorded to the public sector?
    1. Should there be any references to the person?
    2. Should there be any references to the area that the person voted in?
        1. If it should be able to check the result on an area level, should each area then be managed as its own election that is later combined with all the others?
6. How should the system record a vote and the voter without connecting them with each other?
7. How do you think the voting phase could look for a digital election?

Phase 4 - Result phase

1. What is the process for counting the votes?
2. What is the process of revealing the result today?
    1. Should we be able to see the result in real time? (e.g. after each counted vote)
    2. In what resolution should the results be revealed?
3. What are the legal requirements and regulations for reporting and certifying election results?
4. How can the system ensure that each vote is counted accurately and that the results are trustworthy?
5. How do you think the result phase could look for a digital election?

General questions

1. From your point of view, what are some of the potential issues and challenges that could arise with a digital voting system in terms of:
    1. Accessibility?
    2. Voter turnout?
    3. Social trust?
2. What kind of improvements do you think could be made to the current election process?
    1. How could these improvements be implemented by a digital solution?
3. What kind of technology is currently used in the election process?
4. What are the key requirements that the blockchain-based voting system needs to meet to be considered a viable alternative to the traditional paper-based system?
5. Have we missed any crucial parts?

Security questions

1. How can the system prevent or mitigate against potential attacks, such as 51% attacks or denial-of-service attacks?
2. What testing and validation methodologies are best suited for ensuring the reliability and security of a blockchain-based voting system?