UNIVERSIDAD DEL NORTE

MAESTRIA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

THESIS

Continuous and Secure Integration Framework for Smart Contracts

Alvaro Jose Reyes Yepes

Barranquilla, 2022

UNIVERSIDAD DEL NORTE

THESIS

Continuous and Secure Integration Framework for Smart Contracts

Author: Alvaro Jose Reyes Yepes

Supervisor: Miguel Angel Jimeno Paba

Co-supervisor: Ricardo Villanueva Polanco

Barranquilla, 2022

Por todos aquellos que me impulsaron a seguir adelante.
Para los que me veran terminnar y para los que ya no podran.
Cada uno fue simplemente indispensable.

CHAPTER 1
Introduction

In software development projects, organizations work to implement development practices using mostly agile methodologies [1]. One of these practices is DevOps. This software development practice focuses on improving the cooperation between the different teams working on a project, especially the development and operation team, from which the name is derived[2, 3]. The result is an improvement in the productivity of all involved teams by effectively using their time, which leads to faster software development cycles and higher product quality [1][4]. However, there are issues in implementing such practices, as presented in works by [5], [6], and others. Typical issues include manual and time-consuming tasks (which makes project management not scalable, especially for small teams), inaccurate results in testing phases, subjective evaluations, and authority constraints when phases are divided across teams. The work presented by Nogueria, Ana et al. [6] uses machine learning techniques to formulate strategies that lead to even higher product quality.

Although the adoption of DevOps has shown indisputable benefits, its implementation is challenging [7]. This challenge is significantly more complex when the implementation considers ongoing projects with developed practices and habits. The first issue faced is dealing with a currently established development routine [8], as having to re-establish current procedures and having the teams follow it can be met with resistance. The second issue is cost, as there are infrastructure considerations involving an initial cost in budget, skill development, and time taken for integration and maintenance of the tools required [9] [10]. Finally, the third issue is the communication between the developer and operational teams, as continuous deployment can demand additional awareness of the different systems [11].

In recent years, the development of smart contracts has become popular thanks to the increasing interest in cryptocurrencies. Researchers have found interesting uses for smart contracts beyond the new currencies thanks to this revived interest. Smart contracts are protocols that enable and enforce contracts made between several parties on a blockchain. For this reason, the distributed fashion of blockchain creates particular requirements for elaborating the contracts. [12, 13]. As shown in [14], constant changes in the DevOps process can cause unexpected delays, which can be a big issue in the case of smart contracts, given that the language is changing fast [15] and with it, the need to implement new and more complete tools

and development strategies. However, the viability of implementing a few steps of DevOps on an Ethereum blockchain has been proven by Wöhrer, and Zdun [16]. Implementing all the DevOps steps is important to guarantee a project's success. Al-Mazrouai and Sudevan [17], Marchesi, Marchesi and Tonelli [18], Lenarduzzi, Lunesu and Tonelli [19] have all also suggested other processes using agile development as a base. These authors further validate that it is possible to establish a framework to work with blockchains and smart contracts.

CHAPTER 2
Problem & Objectives

2.1. PROBLEM STATEMENT

Blockchain and smart contracts, as mentioned in 1, the implementation of agile methodologies has been combined with the blockchains and smart contracts. Literature on this topic has been showing an increased rate in the past years however, this is still a new field when compared to more traditional software. To gain a better understanding on why DevOps should be properly implemented and the benefits it tries to cover on smart contracts, a research is required.

For this we had the next questions:

1. ¿Which problems are being faced when implementing DevOps on smart contracts?
2. ¿Can traditional DevOps be applied without any change to smart contracts technology?
3. ¿What issues can be prevented with DevOps being applied to smart contracts?

2.2. OBJECTIVES

The thesis aims to propose a DevOps framework to work specifically with smart contracts. This proposal would be independent of blockchain and language as long as the tools mentioned are available. To achieve this, the objectives in Table 2.1 has been specified.

Table 2.1: Objective's Description and Results

Objective	Results
General: Propose a DevOps framework aimed at smart contracts.	Creation of the framework proposed in this document.
Specific: Specify the DevOps steps and how it has been handled in traditional software.	An state of art was elaborated including the information for DevOps steps and software available to aid this process, in both on-premise and cloud scenarios.

Continues in the next page.

Objective	Results
Specific: Specify the different blockchain types that exist, attacks and prevention mechanisms.	An state of art was elaborated including the information for blockchain, smart contracts, vulnerabilities and software to aid on vulnerability identification.
Specific: Design a framework to be used in a DevOps process with smart contracts in consideration.	A framework was built, taking into account the different factors faced when developing smart contracts.
Specific: Create a study case based on Ethereum and Solidity for document identification.	A test application for document identification was created using Solidity in a single smart contract.
Specific: Validate the proposed framework using the developed study case.	A pipeline was created in Azure DevOps to run the proposed framework on he study case application.

CHAPTER 3
State of Art

3.1. DEVOPS

DevOps is a software development practice focused on improving the cooperation between the different teams working on a project, especially the development and operation team, from which the name came [2]. Modern development organizations require entire teams of DevOps to automate and reduce the gap between the development team and the operation team. At the same time, this process needs to acknowledge the interdependence of the different teams to produce faster results for software products and services in an organization [20]. DevOps can be defined in five major phases: Continuous Planning (CP), Continuous Integration (CI), Continuous Testing (CT), Continuous Deployment (CD), and Continuous Feedback and Monitoring (CFM) [21] [22].

CP is the phase that handles dynamic planning, where events are constantly assessed to determine the best action to take according to the specific event. This phase allows plans to be agile and adapt to moving conditions [22]. The process consists of a cycle where managers plan small steps, execute them and get feedback. These steps allow the team to react and adjust the current plan to match the feedback. As a result, the phase generates a prioritized product backlog, and participants can change this prioritization where adjustments can be made at any point [23]. CI allows the integration of multiple activities on a partial automation process going from source control to project building [24][25]. One of the CI process's most significant advantages is that a clean code build is always the result. The result from the CI phase can vary from a compiled executable, an image, a container, or a library to be used on another project [26].

CT is the next phase of the DevOps process, where multiple tests are done over the source code and the compiled build during the CI phase. These practices have evolved from functional testing to include static code analysis and dynamic code analysis for security and performance issues. These analyses allow the detection of possible defects that developers must address as soon as possible. Systems' users access the results of this activity in real-time without depending on each other [27], leading to more software features and finding bugs much faster. After the successful completion of the CT phase, the CD phase happens. The artifact's deployment goes

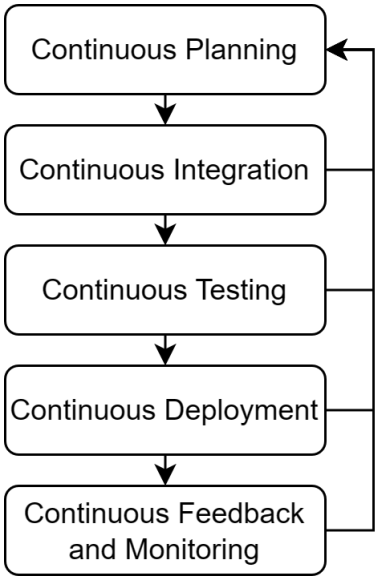

Figure 3.1: DevOps phases

into the different environments for the product. The activities inside the CD phase do not handle the security of the generated artifacts, confidentiality, or infrastructure issues [24]. They do, however, facilitate updating a system component, allowing feedback from the different teams to be made available to the developer more frequently, given the automation of the process. [28]. CD, however, can be divided into two types: Continuous Deployment and Continuous Delivery. The ending activity defines the difference. Continuous Delivery is not fully automated, as it finishes delivering an artifact capable of deployment. Continuous deployment, however, ends with the final artifact being pushed to a system without any decision-making.

Finally, CFM starts once the build has been deployed on a system. As Fayollas, Camille et al. [29] suggested, this could be included in the CD phase. However, to differentiate the most, a new phase is defined. Here, there exists a valuable chance to observe the behavior and usage of the system through different qualitative parameters to react to any bugs or improvements that can be done over the source at an early stage of the deployment [21].

There is, however, a significant point to consider when implementing the tools required by a DevOps process called security. As mentioned before, not only is it a part of the DevOps process to guarantee the security of the artifact being deployed, but the security of the application itself needs to be considered. As mentioned by Düllmann, Thomas et al. [30], security is also a concern in the tools themselves. There must be mechanisms to prevent identity spoofing, data tampering, repudiation, information disclosure, denial of service, and elevation of privilege.

3.2. DEVOPS TOOLS

This section describes software tools for each phase in the DevOps cycle. DevOps tools are the software in charge of helping with the different parts of software development, such as

version control, building, testing, and deployment. Tools are tailored for specific activities of the process.

It is essential first to discuss Source Control Management (SCM) tools. Working on a shared project with a local directory is satisfactory at the start of the project. However, as it starts growing, i.e., other participants start joining and contributing to it, sharing the code changes becomes convoluted, and hence a way to do it is a need [31]. That does not imply that no benefits can be obtained from an SCM if working alone since a sole member may benefit if one wishes to recover a previous code version in case something goes wrong. An SCM tool stores the code in repositories, versions the code, and helps distribute the work between project members. Some SCM technologies are Git, Subversion (SVN), Concurrent Versions System (CVS), Vesta, and Mercurial.

The second set of tools handles the orchestration between the multiple services. These tools coordinate the processes required to compile a project. These ensure that the predefined steps are done for both success and error cases. Sometimes the orchestrator can be built-in in another tool, such as Gitlab (a Git repository manager) with its own CI/CD [11]. However, there are tools dedicated to this function, such as Jenkins, Tekton, Travis-CI, Circle-CI, and AppVeyor [14] [16].

The third tool is not mandatory but is highly encouraged to have and handles the testing of the application. As mentioned before, testing should now include static code analysis and dynamic code analysis without forgetting about unit testing. The used code framework should cover unit testing. However, the other testing tools require other tools. A static code analysis tool is responsible for flagging preconfigured programming and style errors in the source code. Meanwhile, a dynamic code analysis tool flags potential vulnerabilities in a running artifact.

3.3. DEVOPS CLOUD-BASED TOOLS

Emerging cloud technologies influence the paradigm for applications, which has changed as the capability to develop and deploy on the cloud has increased the scalability and reliability of the systems [32]. Cloud providers have had to adapt to these changes causing the support of DevOps tools on the cloud to increase and allowing support for the DevOps process while minimizing the deployment downtime. The architecture that handles all of this is the microservice architecture [14]. One key factor that allows the implementation of the microservice architecture in the cloud is virtualization, which can be done through virtual machines and containerization. Some of the cloud providers that offer DevOps tools services are:

- **Microsoft Azure Pipelines**: Pipelines simplify hardware and VM management using Microsoft's agents and allow users to automate code builds and deployments with Pipelines. Azure offers support for every major platform and tool through container jobs. Important to note is that Azure allows the user to do a deployment to multiple cloud providers and on-premise machines [33].
- **Google Cloud Build**: It is a fully managed platform to build, test, and deploy code from multiple source code repositories. Cloud Build permits the user to deploy the build to

the platform of choice. Important to note is that Google offers "Binary Authorization" to perform deep security scans, enforce standardized container release practices and verify images to ensure no tampering is done on deployment [34].

- **Alibaba Cloud DevOps Pipeline (Flow)**: An automated delivery pipeline service that offers tools to go from continuous integration to continuous deployment. Flow benefits include integrating multiple cloud repositories, code and security scanning, and deployment to public clouds or self-hosted environments [35].
- **IBM Cloud Continuous Delivery (CCD)**: IBM CCD platform allows user to handle their pipeline entirely through tools integration. It offers issue tracking, source code repository, and web IDE in addition to the tools required to build, test and deploy. IBM CCD offers deployment exclusively to the IBM Cloud [36].
- **Amazon AWS CodePipeline**: It is a platform to build, test, and deploy code based on a defined release process model. It is based on three services: AWS CodeBuild (code building and testing), AWS CodeDeploy (deployment to AWS servers or on-premise servers), and AWS CodeStar (user interface for configuration) [37].
- **Redhat OpenShift Pipelines**: OpenShift Pipelines is a Kubernetes-native CI/CD solution based on Tekton. Therefore, this allows each pipeline step to run on its container. Openshift Pipelines can build, test, and deploy applications to public cloud platforms and on-premise. [38]

Table 3.1 compares the analyzed cloud providers that a developer should consider when starting a smart contract development project.

Table 3.1: Comparison of cloud providers.

Cloud Provider	On-Premise Deployment	Multi-Cloud Deployment	Additionals
Microsoft	Yes	Yes	Extension marketplace
Google	No	Yes	Binary Authorization
Alibaba	Yes	Yes	
IBM	No	No	Issue tracking, Web IDE
Amazon	Yes	No	
Redhat	Yes	Yes	

3.4. BLOCKCHAINS AND SMART CONTRACTS

Blockchains worldwide have become the base for digital currencies containing blocks protected from manipulation and alteration. Any block has information regarding the previous block and timestamp. Blockchains implementations are free from alteration as it is impossible to change

the data in a block [39]. First-generation blockchains, like Bitcoin, introduced cryptocurrency transactions as their only function. Meanwhile, second-generation blockchains, like Ethereum, have introduced the possibility of building and deploying software executed by the members of the blockchain [40].

The programs on the blockchains are called smart contracts, which are the core of a blockchain service. A smart contract is, as previously stated, an immutable software code that runs on top of the blockchain. Currently, Ethereum is the most popular blockchain for smart contracts [15], but other blockchains with smart contract support are shown in recent studies [41].

Bugs and security concerns are issues that smart contracts need to solve to have a bigger adoption rate. Security concerns regarding smart contracts have appeared in recent years, as shown by different studies [42] [43] [44]. Bugs are an issue on both cost, as running smart contracts on a blockchain has a transaction cost [45], and security, as unsafe programming can allow an attacker to run undesired code [46]. Since smart contracts convert software programming instructions and due to requirements occurring in multiple scenarios, many smart contract platforms have become available to solve business requirements [43]. Some of these blockchains are [41]:

- **Ethereum**: One of the most popular open-source public blockchain platforms with a cryptocurrency called ETH or Ether. Ethereum is the oldest smart contract platform, allowing developers to build decentralized apps through its Ether. Ethereum Virtual Machine (EVM) software stores and executes all smart contracts, with Solidity as the relevant programming language. However, this blockchain suffers from concurrency issues which developers are working to reduce [47].
- **Hyperledger Fabric**: A private blockchain platform with smart contract features that first became available as an enterprise blockchain platform. It is an open-source Linux project which supports the collaborative development of blockchain-based distributed ledgers. This blockchain architecture bases itself on a microservice architecture for an appropriate deployment. Hyperledger developers have access to tools that allow them to develop smart contracts more efficiently and quickly [48].
- **New Economy Movement (NEM)**: A platform for private blockchains focused on building solutions for requirements using a cryptocurrency called XEM, which can be traded but not used as payment. Created initially in 2015 based on another blockchain called NXT [49, 50]. Unlike other blockchains, NEM uses a proof-of-importance (POI) mechanism instead of a proof-of-work (POW) mechanism. NEM offers several advantages to the users, such as ease of deployment, deep customization, performance, and complete security.
- **Stellar**: It is a blockchain-based platform that facilitates economic transactions over boundaries. For the Stellar blockchain, smart contracts manifest as Stellar Smart Contracts (SSC) [51]. The crypto coin used in Stellar is Lumen [52]. A Stellar Smart Contract (SSC) is a composition of connected and executed transactions using various constraints with a processing time between three and five seconds. Stellar allows users to create, trade, and

send digital representations of all forms of money (i.e., bitcoin, dollars, and pesos) while securing this with the Stellar Consensus Protocol (SCP).

- **EOS**: It is a blockchain-based platform designed to develop scalable and secure applications with smart contract capability [53]. EOS provides decentralized storage of enterprise solutions to solve the scalability issues faced by Bitcoin and Ethereum. A difference between the EOS platform and others is that it eliminates all users' fees and uses a Proof-of-Stake (PoS) algorithm.
- **Corda**: An open-source blockchain project, designed for business from the start, allows users to transact directly with smart contracts [54]. This practice streamlines business processes to reduce transaction costs and record-keeping. The R3 Corda platform represents the smart contract corresponding to real-world contracts. It is an agile and flexible platform that can scale to meet business requirements. Applications built on Corda, CorDapps are designed and developed to transform businesses across various sectors, including insurance, healthcare, finance, and energy.

Table 3.3 has an abbreviated comparison of the blockchains discussed before.

Table 3.3: Comparison of most used blockchains.

Blockchain	Open source	Supports Cryptocurrency	Miner participation	Smart Contract Language	Consensus Mechanism	Scalable
Ethereum	Yes	Yes	Public	Solidity	Proof of Work	No
Hyperledger Fabric	Yes	No	Public, Private	Java, Solidity, Golang	Proof of Work	Yes
NEM	No	Yes	Private	Java	Proof of Importance	Yes
Stellar	Yes	Yes	Public	Solidity, Javascript, Java, Go	Stellar Consensus Protocol	Yes
EOS	Yes	Yes	Public	C++	Proof of Stake	Yes
Corda	Yes	No	Private	DAML, Kotlin, Java	Validity and Uniqueness	Yes

3.5. SMART CONTRACTS SECURITY

In addition to the security risks posed against distributed ledgers, such as 51% attack [**9567686**] against blockchain based on mining [**Classification**, **math10142504**], as well as emerging threats as cryptojacking [**aponte2022detecting**], there are other risks directly against smart contracts. The distributed and immutable characteristics of a smart contract in a blockchain had consequences when faults in them caused economic impacts in multiple cases [46][55]. Such risks pressed developers to hasten the creation of multiple tools to mitigate these impacts. However, not all tools have been kept up to date or made accessible to a larger public, as shown by Lopez, Antonio et al. [56]. There is also the difference in programming language between the blockchains that only makes them available for some blockchains [41]. It is also worth mentioning the analysis of the tools developed are built for static code analysis instead of a dynamic code analysis [45].

There are doubts about the current tools available for smart contract programmers, which has caused proposals such as SolAnalyzer [57], SolAudit [55] and SuMo. However, the biggest concern is the need for articles concerning employing DevOps for a smart contract pipeline.

3.5.1. Smart Contract Vulnerabilities

Information on vulnerabilities of Solidity Smart Contracts has been collected in multiple studies such as Villar Lopez et al. [56], Badruddoja et al. l [44], Dika et al. [45], Akca et al. [55]. Most studies focus on and explain the most critical vulnerabilities presented in this section.

- **Reentrancy**: This vulnerability is considered one of the most severe. The reentrancy vulnerability relies on the interaction between two smart contracts. If, through smart contract dependencies, a contract hands over the control to another contract, it allows this second contract to call back into the first contract before the first initiated interaction between them is completed. Therefore, the second contract could have an action to refund gas and do this operation multiple times to empty the balance of the smart contract of the target. The correct order of operations during a balance transfer is essential to prevent this attack.
- **tx.origin**: tx.origin is a global variable used in Solidity smart contracts containing the account's address that sends the transaction. The vulnerability, named after this variable, aims to identify the user who initiated the chain of interactions between contracts. This action is done during the authorization method to spoof unauthorized users as authorized and obtain leveraged privileges in the contract.
- **Callstack depth exception**: Contracts have a call stack limit of 1024. Therefore it is possible to make a smart contract execution fail by making external calls and exceeding this maximum stack call size. An attacker can use this call as an advantage to produce an output that suits them the best during a contract execution if the call stack exception is not handled correctly by the contract.
- **Timestamp dependence**: A contract that depends on the timestamp can be vulnerable if used in a vital contract call. An attacker can manipulate the timestamp to produce an

output that suits them best.

- **Transaction-ordering dependence**: This vulnerability is a prevalent security bug in the smart contract that consists of relying on the order of transaction execution. This vulnerability can have the attacker be the smart contract owner or the miner. It consists of making the gas price of the transaction change during the execution of the smart contract because the attacker sends a transaction that modifies the price before the transaction being executed finishes.
- **Gasless send**: This situation happens when, under certain conditions, the gas sent to execute a contract call is not enough to cover the call. This situation generates a "gas exhaustion" exception and, therefore, a transaction failure. The exception happens if the recipient's contract has a fallback function with a large code base. It is crucial to throw an exception if a failure based on gas consumption happens and to ensure that the gas requirements for contracts' calls are not too high.
- **Call to the unknown**: It happens when a Call, Send or DelegateCall primitive of the Solidity language is called inside a contract. However, this one internally uses a function defined inside all contracts (as part of the Solidity language) but is not found in the miner's environment during execution.
- **Overflow and underflow**: This vulnerability can occur when transactions do not check the input data to verify it is an authorized input. Smart contract overflow occurs when the value provided exceeds the maximum value defined for its data type. In the case of Solidity, it is a 256-bit number. In the case of underflow, it is trying to achieve the same by using numbers under the limit permitted by Solidity.
- **Short address attack**: This vulnerability occurs in Ethereum Virtual Machine (EVM). It permits padded arguments that allow an attacker to send a crafted address that leads to a contract exploit. The EVM will add zeros to the end of the encoded arguments to make up for an expected length of 20 bytes (applied only when the argument is less than 20 bytes). This attack is not explicitly made on the Solidity contracts themselves but on the third-party applications that interact with them. It is usually an issue when third-party applications interacting with smart contracts do not validate the inputs.

3.5.2. Analysis Methods

For developing secure software, especially in the case of smart contracts, it is essential to make the code error-free. To achieve this level, developers need to involve a wide array of security testing [58]. Security testing for vulnerability detection methods divides into two significant testing methodologies: static vulnerability detection and dynamic vulnerability detection [59]. These analyses can be specialized to follow specific standards according to the sector where the program's execution will occur in [60].

Static Vulnerability Detection

Static software analysis is a way of studying a program from its compiled binary code without executing it. The basic concept common to all static analysis tools is checking the source code

to identify specific coding patterns that often lead to vulnerabilities [61]. Manually checking for these patterns is an option, but automated tools are more effective than manual options. There are several analyses done when doing static analysis [56]

- **Control analysis**: Focuses on the flow of the different calls in a structure. The analysis revises the calls done in a process, function, method, or subroutine.
- **Data analysis**: Concerns using defined data to ensure data objects are correctly operating.
- **Fault/failure analysis**: Analyzes faults and failures in the data components used inside the code.
- **Interface analysis**: Verifies the interfaces between solutions and applications to determine that the components interact appropriately.
- **Pattern recognition analysis**: Searches for portions of code known to contain potentially vulnerable code.

Dynamic Vulnerability Detection

Dynamic software analysis studies a program during service execution to find vulnerabilities that can only be detected upon the execution of a service [62]. This process consists of running multiple inputs in the program to crash it [63]. There are several analyses done when doing static analysis:

- **Fault localization**: Searches for faults in the code by giving multiple random values to tests to see if the tests fail with a specific value.
- **Memory errors and concurrency errors analysis**: Searches for resource and memory leaks during race conditions on multi-threaded programs.
- **Performance analysis**: Traces software applications at run-time and captures data to identify the causes of poor performance.

3.5.3. Blockchain development tools

A tool can be defined as a concept, technology, software system, template, framework, or library that aims to help design, development, and maintenance of a software product [64]. With this in mind, multiple tools used during the development of smart contracts are shown in Tables 3.5 and 3.6, with information regarding IDEs and Security Tools.

A tool can be defined as a concept, technology, software system, template, framework, or library that aims to help design, development, and maintenance of a software product [64]. With this in mind, multiple tools used during the development of smart contracts are shown in Tables 3.5 and 3.6, with information regarding IDEs and Security Tools.

After, on Table 3.6, we organized the tools not classified as IDE for security analysis. Here, we can find tools executed for static and dynamic analysis of the code created for smart contracts. In both cases for Table 3.5 and 3.6, we added information regarding the available blockchain and the last time they were updated.

Table 3.5: Blockchain development tools for IDE purposes.

Description	Tool Name	Classification	Blockchain	Last Updated
Web-based IDE with built-in static analysis, and a test blockchain virtual machine	Remix [65]	Web-IDE	Ethereum	Apr 2022
Web-based IDE that lets you write, compile, and debug your smart contract, powered by Loom Network	EthFiddle [66]	Web-IDE	Ethereum	Jun 2021
A Cloud-Based Multi-Chain IDE that provides debugging, testing and deployment one-stop services, developers don't need to install extra tools while working on smart contracts	ChainIDE [67]	Web-IDE	Ethereum	Mar 2022
A customizable development environment for Ethereum with hot reloading, error checking, and first-class testnet support	Replit [68]	Web-IDE	Ethereum	Apr 2022
Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux.	Visual Studio Code [69]	Local-IDE	Ethereum, Hyperledger Fabric, Corda	Apr 2022
IntelliJ IDEA is an integrated development environment written in Java for developing computer software.	JetBrains IDEs [70]	Local-IDE	Ethereum, Corda	Apr 2022
Remix Desktop is an Electron version of Remix IDE. It works on Linux, Windows, & Macs.	Remix Desktop [71]	Local-IDE	Ethereum	Dec 2021
Truffle is a development environment, testing framework and asset pipeline for multiple blockchains.	Truffle [72]	Local-IDE	Ethereum, Corda	Apr 2022

Continues in the next page.

Description	Tool Name	Classification	Blockchain	Last Updated
An application development framework which simplifies and expedites the creation of Hyperledger fabric blockchain applications.	Hyperledger Composer [73]	Local-IDE	Hyperledger Fabric	Aug 2019
Stellar uses no specific IDE to work with. You install the stellar-sdk for the language you want to program with which means you can interact with the network through the API	Stellar-SDK [74]	Local-IDE	Stellar	Apr 2022
A web IDE integrated with various tools required for EOSIO in a unified graphical application.	EOS Studio [75]	Web-IDE	EOS	Feb 2020
Desktop version for EOS studio for Windows, Linux and Mac.	EOS Studio Desktop [76]	Local-IDE	EOS	Sep 2019
IDE providing developers with a personal single-node EOSIO blockchain for development and testing.	EOSIO Quickstart Web IDE [77]	Web-IDE	EOS	Jun 2020
A source code editor for Windows, Linux and macOS.	Zeus IDE [78]	Local-IDE	EOS	May 2021

Table 3.6: Blockchain development tools for IDE purposes.

Blockchain	Tool Name	Classification	Description	Last Updated
Ethereum	Oyente [79]	Static Analysis	An Analysis Tool for Smart Contracts	Nov 2020
Ethereum	Solgraph [80]	Static Analysis	Visualize Solidity control flow for smart contract security analysis	Jan 2019

Continues in the next page.

Blockchain	Tool Name	Classification	Description	Last Updated
Ethereum	MadMax [81]	Dynamic Analysis	Ethereum Static Vulnerability Detector for Gas-Focussed Vulnerabilities	Jun 2021
Ethereum	Manticore [82]	Dynamic Analysis	Manticore is a symbolic execution tool for analysis of smart contracts and binaries.	Mar 2022
Ethereum	Mythril [83]	Dynamic Analysis	Security analysis tool for EVM bytecode.	Apr 2022
Ethereum	ContractLarva [84]	Dynamic Analysis	Runtime verification tool for Solidity smart contracts.	Mar 2022
Ethereum	SolMet [85]	Static Analysis	A static analysis tool for calculating OO-style source code metrics for Solidity smart contracts.	Nov 2020
Ethereum	Vandal [86]	Static Analysis	Static program analysis framework for Ethereum smart contract bytecode.	Jul 2020
Ethereum	Securify [87]	Static Analysis	A security scanner for Ethereum smart contracts.	Sep 2021
Ethereum	Slither [88]	Static Analysis	Static Analyzer for Solidity	Apr 2022
Ethereum	Ethlint [89]	Static Analysis	Code quality & Security Linter for Solidity	Sep 2019
Hyperledger Fabric	Revive-CC [90]	Static Analysis	Static analysis tool for Hyperledger Fabric smart contracts written in Go.	Jul 2020
Hyperledger Fabric	Blockchain Analyzer [91]	Data Analysis	Analyze ledger data stored within a Hyperledger Fabric peer.	Feb 2020
Hyperledger Fabric	Chaincode Analyzer [92]	Static Analysis	CLI tool to detect the codes which can be risks potentially such as nondeterminism in smart contract in Hyperledger Fabric.	Feb 2020

3.6. SELF-SOVEREIGN IDENTITY

For the study case, we opted to work on a system based on self-sovereign identity for document validation. This section describes the self-sovereign identity (SSI) for clarity.

The majority of current existing identity systems are built on centralized storage architectures. In this type of architecture, the identity provider is considered Trusted Third Party (TTP) [93]. Both the service provider and the user need to trust a the TTP to allow for authentication while also trusting them with the required users' personal information. The data stored at either cloud storage or centralized servers is considered by many to be a privacy concern because of the various types of attacks or data breaches that can happen during operation. [94]. Recently, however, a shift in identity management solutions has happened with the growth of blockchain. Blockchains, acting as a decentralized ledger, provide an answer thanks to the immutability factor, where integrity of the transactions can be checked by anyone. With a blockchain-based identity solution, the user is given control of their identity and is the sole user in charge of allowing his or her visibility [95].

3.6.1. Principles

The SSI systems are based on multiple principles that separate them from traditional identity mechanism [94] [96] [97]. These principles are:

- **Existence:** The identity must always reflect a human user that allows it to access a service.
- **Control:** The user must be able to exert control over its digital identity and attributes.
- **Access:** The user must be able to always access the data associated with its identity.
- **Transparency:** Any service providers must be transparent in the information that is being used from the user.
- **Persistence:** The user's identity must endure as long as a user deems it appropriate.
- **Portability:** The user must be able to transfer its identity from one provider to another.
- **Interoperability:** The user's identity should work with any service provider.
- **Consent:** The user is the only party that permits information sharing.
- **Minimalization:** If a user permits information to be During the usage of the identity, especially when disclosing attributes, only a minimum amount of data must be disclosed to third parties. The principle of data economy should be adhered to.
- **Protection:** The axiom of protection implies the precedence of user rights. In case of a conflict between the identity holder and the network, the decision should be in favor of the identity holder.

3.6.2. Four Privacy Layers of SSI

With the principles mentioned before, four privacy layers of SSI have also been defined which allow users to share as much as they desire about their identities. The layers defined are the following [96]:

- **Level 0:** No information is shared with others.
- **Level 1:** Only the identity defined by public/private key pairs is shared.
- **Level 2:** Identity is allowed to trusted third parties/organizations
- **Level 3:** Identity is allowed to be publicly shared.

3.6.3. Blockchain-based SSI

After the introduction of Blockchain technology and smart contracts, the implementation of a decentralized identifier that realize the SSI paradigm was finally be abled to be introduced [93]. Blockchain use cryptography to authenticate users and sign messages that are sent to the network, this allows to have mechanism be applied for user credentials for authentication and identity control. Moreover, a decentralized identifier ensures the uniqueness of the identity in the system. With this in mind, the removal of a central authenticator is not needed to guarantee individuality. The relation between all the parts can then be established as following [93] [98] [**sis8**]:

- **Issuer** The agents in charge of defining a credential and issuing or revoking those credentials to other users.
- **User** The identity owner. The holder of an identifier, in charge of presenting credentials when required. Is able to control the privacy of their information.
- **Verifier** The agent requiring to a ascertain information from a user to be able to authorize movements.

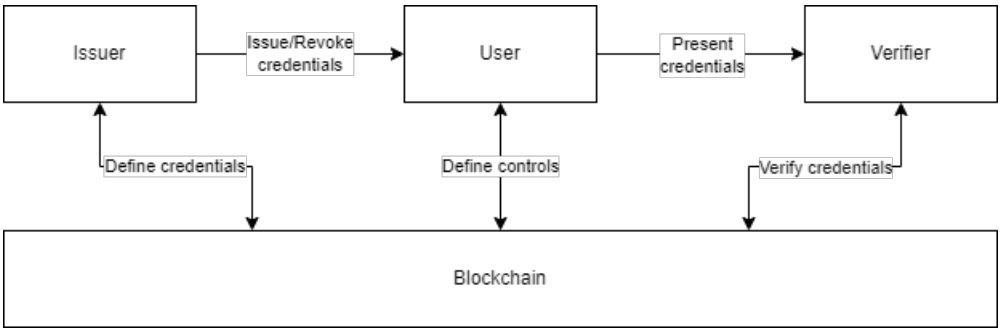

Figure 3.2: Blockchain-based SSI

3.6.4. Proposed Blockchain SSI Models

On the literature, multiple models have been proposed to work with blockchains and SSI, however on the commercial side many more have been established [99]. These aim to establish mechanisms to identify a user based on decentralized systems while giving control of their information to the user for data safety and privacy. Some of these models are:

- **Casper** built as an Android/iOS based mobile identity wallet application as use case for an academic proposal. It provides the user ability to save their identities in their own mobile wallet application wit the proof of these in the blockchain. Casper platforms'

provides a Zero Knowledge Proof mechanism to other users that allows them to verify the identity information. [94].

- **ATIB** stands for Attribute Trust-enhancing Identity Broker. The academic proposal includes an abstract of a dedicated SSI solution with plans to offer standard protocols for multi party consumption in order to facilitate adoption by different service providers. The aim of this model is to comply with the SSI principles for the user [93].
- **MQTT-based** The Message Queuing Telemetry Transport (MQTT) protocol is often cited to present security challenges due to their authentication mechanisms. The academic proposal, however, presents an approach involving an authentication scheme using decentralized identity system to manage the users' identities. By using this mechanism, the proposal aims to facilitate both the queue subscribers and publishers their authentication by utilizing a smart contract in Ethereum blockchain to guarantee trust while preseving their privacy [100].
- **Sovrin Network** is an open-source framework for managing digital identity of users. The Sovrin Network is managed by a non-profit organization Sovrin Foundation. This framework is built on the Hyperledger blockchain. In this framework, a digital identity is created that permits the user to be identified privately without sharing any information using Zero Knowledge Proof mechanism. [101] [102].
- **uPort**, like Sovrin Network, is also an open-source framework for decentralized identity management. This framework however operates on Ethereum, where users can use their identity to sign transactions and control their own data by storing it in a personal wallet instead of a third-party. [101] [103].
- **Jolocom** is also a framework that stores the identifiers on the public Ethereum blockchain. The information here includes how certain attributes may be used where this is under user control [101].

CHAPTER 4

Framework Proposal

This section presents the proposal for the application of DevOps to smart contracts. When used with both traditional software and blockchain smart contracts, the underlying DevOps concepts and techniques are strikingly similar. However, some phases are different while working with blockchains because of the limitations. In more detail, we outline the suggested DevOps phases as they apply to blockchains and smart contracts, along with the tasks involved in each phase and useful indicators that provide ongoing feedback. When it comes to tasks, we describe each one in full, including the input and output that are anticipated. In addition, important indicators that provide feedback for the different teams throughout the entire process will be taken into account and described in their own area.

4.0.1. Preparation

While the preparation itself is not a component of the DevOps approach, some of the actions conducted prior to beginning with DevOps are. Also, we consider this phase important to assure the quality and security of the following phases. It's important to remember that a specialized team with the capabilities to take decisions should handle this step alone. At this beginning stage of the phase, a number of tasks should be finished, including:

Analysis of current systems

The first phase should be an analysis of the existing resources, and the person or group in charge of it should provide all of the input. Repurposing the current resources requires a thorough analysis of the development team's existing systems. This action can result in less training time because the team is already familiar with the present procedures, as well as a means to save money because no new system will be needed. The analysis of the current artifact and container repositories, the development tools (both for version control and automated testing), and the tools available for general work management are the results of this analysis, which is presented as a report to determine whether these tools will be compatible with the new blockchain project. A report containing no information would also be a valid result because it would indicate the need to buy or make plans not to use the unavailable tools (not recommended).

Defining the quality strategy

As development consists of both functional and non-functional requirements, the recommendation is to work with a development and a quality environment before releasing the new smart contract onto the blockchain (if possible, as this depends on the blockchain). The input for these tasks consists of the identified quality metrics defined for the project (if they exist), with the output consisting of the testing strategy and the release strategy while ensuring the use of the distinct environments.

Securing the development process with the respective tooling

With the input taken from the two previous tasks, this task validates the security practices required for the defined strategy. Security not only implies the integration of security tooling for the code but also for the tooling systems used by the team. If the team has no tools available, as an empty report is considered valid, the recommendation for the whole secured environment must be given (version control, artifact management tool, automated testing, and building).

Validation of existing code base and dependencies

For this task, the information of previous source control and the repositories of existing systems are taken to validate the code base for compliance and inspect the dependencies for security flaws. The output of this step should be the code requiring manual validation and dependencies requiring updates due to security failures.

Setup of tools environment

Considering all input taken from the previous steps, now a decision must be taken on the minimum necessities for the project and allow growth according to the given budget for tools that would allow saving time due to simplifying tasks. With all decisions made, this line of tasks ends up with an environment set up to start DevOps.

4.0.2. Continuous Planning

The DevOps first phase and the one that generates will generate the input for the following phases is the CP phase. As smart contracts should focus on a single task, developers should prepare to plan according to this ideology, where one prefers multiple small smart contracts [16]. This planning saves on the number of transactions done and thus reduces the execution costs.

Task Discovery

The first task of the whole DevOps strategy will be to collect the information regarding the requirement needed to deploy. This result is considered a gap, and the multiple small tasks required to complete the gap are defined. This step is done by a single senior developer or a group of developers.

Task Planification

The second task of the whole DevOps strategy will be management work. The project manager is informed of the tasks required to complete the expected functionality. These tasks must be registered in a work management flow tool to keep track of the activities in progress (e.g., Jira, Gitlab, ClickUp), which will serve as output for this task.

Task Assignment

This final task will now take the activities registered on the previous task as input. Again, a senior developer or team will assign the responsibility based on the available human resources. They consider the task's difficulty to assign to a developer according to their skills. This task then leaves an output of assigned activities with the requirement to be done by the developer.

Indicators

These indicators will help provide a better understanding of the capacity of a team to fulfill the tasks created.

1. Resources Available: The total amount of resources assigned to a project. This doesn't necessarily mean only human resources but other types of resources too (computational, tools, third-party supports).
2. Assignment Time: The time taken between task creation and task assignment.
3. Lead Time: Time taken between task assignment and task completion. This can only be calculated once the full development cycle is completed.
4. Feature Prioritization: Reinforces the value of DevOps, where constant iterations are being done to meet user demands. This will determine if the tasks being assigned meet the needs of those features being utilized the most.

4.0.3. Continuous Integration

As explained previously, CI allows the integration of multiple activities on a partial automation process going from source control to project building. The input for this step is the activity list previously defined during the CP phase to solve the gap. The general flow, as previously mentioned, involves developing and building the code.

Code

Developing new smart contracts involves taking the activities planned during the CP phase to solve the requirements needed. Various programming languages (e.g., Solidity, Java, C) can be used depending on the blockchain. Depending on the blockchain, the IDE developers will work with also varies (e.g., Remix, VSCode, HyperledgerComposer). The output of this step will be the code that solves the requirement given.

Unit test creation

After being done with the code, or in parallel, the unit tests should get coded too. This step will guarantee that the code is ready for production and behaving as the developer expected. The input for this task is the current developed code, and the output is the unit tests created to assert that the code is behaving as was initially expected.

Commit

Different Git repositories (e.g., Github, Gitlab, Bitbucket) and SVN repositories (e.g., SourceForge, CloudForge) can store the developed code. The code, the input for this case that is published, will then be submitted for a merge request. This process involves giving human approval to take the submitted changes and apply them to the main code. This step works as a peer review to at least get a second pair of eyes to validate the changes. No automatic acceptance is recommended here as the chance for a peer review before testing is lost. In the end, the step output will be the merge requested code.

Build

The build step includes all the steps required to generate the artifacts needed for execution from the source code. The compiler and instructions to execute the source code change depend on the blockchain and programming language used. The input taken for this step is the merged code that is now in the code repository. An established CI software will then do the build (e.g., Jenkins CI, Travis CI, Gitlab CI/CD), which will involve automating the actions required to generate the final output of the CI phase: the compiled smart contract.

Indicators

These indicators will help provide a better understanding of task fulfillment by developers.

1. Unplanned Work: Represents on-demand changes done while work is in progress due to unidentified situations during task discovery. This could also be due to unexpected situations occurring due to ongoing changes. No matter the reason, however, this is a metric that should be accounted for. Aimed at being a low value, as it represents proper discovery work being done and no changes being accepted once work has started.
2. Change Volume: Determines the extent to which code was changed when fulfilling a given task. Ideally, the change volume should remain low implying only the task objectives are being changed.
3. Test Coverage: Constitutes the percent of functionality in a smart contract that is covered by unit tests. A low value implies the unit test creation step is being skipped, which then will lead to unexpected bugs.

4.0.4. Continuous Testing

The CT phase has steps that can be executed only after the CI output. However, some of the analyses done during this phase can be executed even during the CI phase. To separate responsibilities between phases, it is assumed that all testing will get done after the CI phase. In this phase, a series of automated tests review the output of the smart contract. As in traditional software, multiple layers are available for testing in smart contracts. Layers can dictate a subdivision, i.e., the contract, the data, or the blockchain's consensus [104]. When a test involves functional testing, there should be a simulated local temporary blockchain where the smart contract can be deployed and tested.

Static Analysis

This step involves taking the source code as input and using a subset of tools to test it for different issues (e.g., checking for errors, checking structural problems) without executing the smart contract. For example, Splinter is one tool that can be run against the source code to test the smart contract. Other examples of tools are in Table 3.6. The test metric should be as error-free as possible, and the output of this test is either a pass or no pass.

Dynamic Analysis

Different tools are available to work on dynamic analysis. Some take the source code and work as a static analysis tool, while others take the generated output and work on this. These tools examine the code for potential bugs, undesirable patterns, and patterns that could lead to possible errors during the smart contract execution. Some of these tools are Mythril, Manticore, Vertigo, and Echidna. This process naturally comes with the possibility of false positives, so a minimum required metric for the output of pass or no-pass should be planned.

Private Blockchain Deployment

Before unit testing or integration testing can happen, there is a need for deployment on a blockchain. The suggested approach due to cost and control is to have a private blockchain already created which has been done in the setup of tools environment mentioned in the Preparation phase. This task will take as input the build previously generated and deploy it into the test blockchain where the following steps can connect. The output of this task will be a success or failure of the deployment.

Unit Testing

The third type of testing to be executed is unit tests. According to the survey by Chakraborty et al. [105], this step should not be skipped as it was the testing methodology that most commonly found issues on blockchain software. The ideal scenario for unit tests includes: all methods covered, all inputs validated, transactions reverts checked, and access privilege verified [16]. This task takes as input the unit tests previously developed. The output for this step includes a

metric informing the tests passed. The advantage of unit testing is the complete automation and parallel testing execution using frameworks such as OpenZeppelin or Truffle.

Integration Testing

This type of testing validates the interaction of various components [106]. In the context of smart contracts, this involves setting up multiple smart contracts in specific states to validate the proper behavior of the smart contract. This activity can be time-consuming, mainly due to the manual setup needed for each case. The input for this activity consists of the smart contract functionality to test, and the output consists of a metric determining the tests passed.

Indicators

These indicators give insight into the testing being executed over smart contracts. Low or high values, depending on the indicator, can reflect future issues that will be faced if deployment is done instead of revisiting the scenarios.

1. Test Pass Rate: If code releases consistently fail unit tests, this suggests that teams are ignoring secure practices and work must be done to correct those issues. This value should always remain high, as it implies functionality works as expected according to the tests created.
2. Error Detection Rate: Represents the errors and warnings detected by the static and dynamic analyses. The detection rate value aim is to be low as possible, which will imply the common errors found in code are accounted for.
3. Escape Rate: Represents the value of changes creating changes in functionality that were not caught in testing. A high value will imply the testing will need to be reviewed to see where the issue lies.

4.0.5. Continuous Deployment

As described by Górski, the CD aims to enable on-demand software release [107]. However, the smart contracts' complexity and interaction with the blockchain strain the phase. In this phase, the input obtained from the CT phase will be the metrics that will allow the process to decide if the compiled smart contract will be released and deployed.

Release

The release step is a decision step, where all the inputs from the testing get evaluated against previously decided acceptance metrics. If the step decision is "pass", the smart contract is considered releasable. This decision is followed by generating all the files needed to release the smart contract in a blockchain. The output for this step will be the files prepared for deployment.

Deploy

The deployment step can be manual or automatic, working on a pull (deploy when you need) or push (deploy when you release) configuration. The release decision on how to operate the deployment depends solely on the operation team, as the smart contract would then be immutable in the blockchain [108]. The input for this step is the released contract generated on the Release step, and the output is whether the smart contract has been successfully deployed.

Indicators

These indicators provide information regarding deployment and the team's ability to respond to scenarios regarding this.

1. Deployment Time: The time between release and deployment. Deployments can occur with high frequency if tasks are simplified enough, but these times should remain relatively constant. Any kind of dramatic increases in deployment time will warrant further investigation,
2. Recovery Time: The time indicating the team's ability to respond appropriately to issues during deployment. If issues are found promptly but not followed by an equally rapid recovery time it would mean little.
3. Failed Deployment Rate: Related to the previous indicator, this value aims to represent the percent of deployment failures. The value should be kept as low as possible to indicate a low amount of issues are being generated.

4.0.6. Continuous Feedback and Monitoring

Assuring smart contracts are secure before deployment is preferable when possible [109]. This step in the DevOps process verifies proper behavior and generates feedback for the CP phase. This phase still has work to be done in the current state as the available tools do not permit extensive monitoring. The input for this phase is the deployment notification from the deployment step, which will allow the monitoring to start and generate feedback.

Monitoring Smart Contract

Monitoring smart contracts, as previously stated, is a challenging task as it depends on the capabilities of each blockchain to give the information required to allow monitoring. Assuming data is available, possible metrics expected are transactions versus actual transactions, the overall cost per transaction, and transaction speed. Also, even when blockchains can resist data loss or unintended data manipulation to a certain extent, there is a need to monitor the data integrity. This monitoring could be done automatically by generating automated reports with the compiled data.

Monitoring Permissions

In cases where authorizing accessing data is done, via methods, features, or permissions, an attacker or a misconfiguration can lead to a breach of privacy. Therefore, since it is required to ensure that confidential data remains as so, the team must be able to check on data access. It is possible to monitor data access manually or automatically by using state comparison tools to raise warnings in case of differences.

Monitoring Client

Having access to a client node for the smart contract allows the monitors to validate the proper behavior of the smart contract on that specific client node. Multiple validations on distinct nodes would allow the generation of reports. However, this implies the work has to be done manually and, therefore, will have a high resource cost.

Feedback

This step involves taking the information from the monitoring steps and generating a feedback report. This report will then be sent to a team lead to evaluate utilizing the same work management flow tool defined in the CP phase as a new task for the team lead.

Indicators

Not the most important, but highly valuable information is contained in these indicators. This determines whether a good job was executed by the different teams and allows for feedback to be given appropriately.

1. Compliance: This highlights the difference between the deployed work and the planned work. A high compliance value ensures that expectations are being met.
2. Ticket Volume: This concept reflects alerts generated by a monitoring user to indicate bugs or unexpected functionality. An increased ticket volume suggests issues in the deployed code and/or issues not caught in testing.
3. Performance: A key indicator for any application. Code deployed should not affect the performance of previous functionality and should not generate unaccounted slow performance times.

4.1. GENERAL USE CASE

This section describes a framework summary using a use case to guide the steps required to run DevOps on smart contracts. The guidance image is in Figure 4.1. This figure explains the overall process and steps to take in each.

1. The next is the continuous planning phase, which consists of two tasks: planning and assignment. Issue tracker software covers these functionalities. Examples of software

dedicated to issuing tracking are Atlassian Jira, HubSpot, ClickUp, and Backlog. However, other applications like Gitlab or Microsoft Teams have issue-tracking capabilities.

2. The next phase is the continuous integration phase, which consists of coding tasks. For the smart contract coding, it is necessary to create unit tests for the smart contract and commit to the repository hosting the smart contract or groups of smart contracts. Once completed, a build process using an automation server should verify that the recent commit will allow the code to build successfully. The repository, with technologies mentioned in Section **??**, can be hosted by software such as Github, Gitlab, VisualSVN Server, or Apache Subversion, among others. Meanwhile, the automation server, as mentioned in **??**, can be done by software such as Jenkins, Tekton, Travis-CI, Circle-CI, and AppVeyor.
3. The next phase is continuous testing, which tests the code directly using static analysis tools. Dynamic analysis tools are also run on the compiled code. Some tools are referenced in Table 3.6. The automation server should do these tests once the build successfully generates a report on possible bugs. Finally, manual and automatic integration tests are available using blockchain simulators such as Ganache or Geth once all tool testing finishes.
4. The release of the smart contract is generated, which involves the creation of the needed files to publish the smart contract on a blockchain. After this, the files are deployed onto the blockchain. Everything in the continuous deployment phase should be left to the automation server, or at least as much as possible.
5. The continuous monitoring of the released smart contract starts operating. The proposal is to do this by monitoring statistics related to the smart contract and the data generated by it and its users. Also, monitoring related permissions to the smart contract must constantly, so there is a verification regarding undesired permission changes. A feedback report is filed from the operations team to send back to the development team.

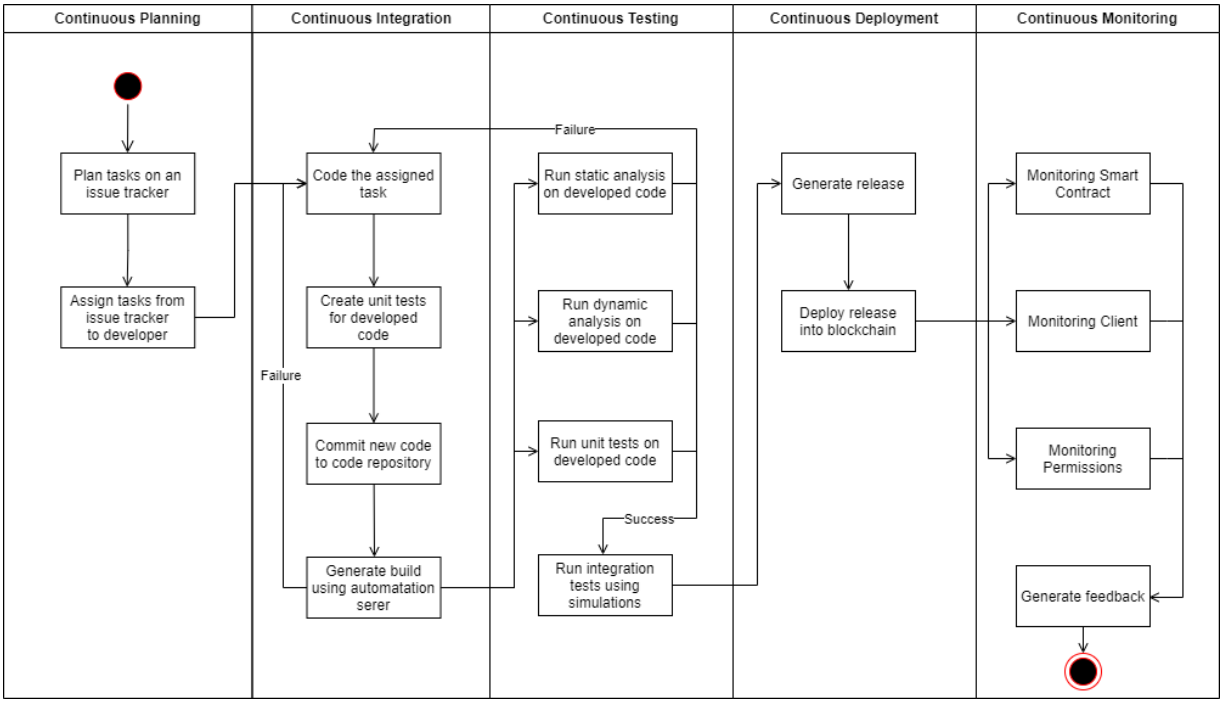

Figure 4.1: Framework example

CHAPTER 5

Study Case

For framework proposal to be implemented, we required a study case we could control. For this, we created a smart contract with Solidity for Ethereum. The proposed system for this smart contract is based on model behind SSI, but instead aiming for this to be used for general document identification where the user has complete control on the data that is being shared to the document verifier.

5.1. REQUIREMENTS

Based on the information provided in Chapter 3 regarding SSI, we set the next requirement objectives for the system:

- **Authentication** A user must be able to log in to the system using their own account.
- **Authorization** After proper authentication, the system must allow the user must be able to only do what their permissions state.
- **Verification** A user must be able to ascertain the information provided by a user regarding their document.
- **Flexible Attributes** The documents creation must be generalized to allow for any type of data to be added into it.
- **SSI principles** The application must follow the SSI principles and allow the user control of its data.

5.2. SYSTEM USERS AND OPERATIONS

Once the requirements were established, the users and the operations required to comply with this were identified. The users that interact with the system are the following:

- **Owner** This account would be the initial owner of the contract and general system admin which would be in charge of authorizing specific accounts to access certain functions.
- **Issuer** The accounts that would be allowed define their own document schema and issue a document to an specific document holder.

- **Document Holder** User accounts with access to multiple assigned documents, should be allowed control of their information and when to share it.
- **Verifier** The accounts that require verification that a document presented is valid, for this they would require to know the schema beforehand. A verifier can also be an issuer.

With this system users in consideration, the operations in the smart contract that would allow the system to work were defined as following:

- **Owner change** The system must allow the owner to be changed, but only if the owner executes the function.
- **Role assignment** The system must allow the owner assign issuer permission and verifier permissions to other accounts, this accounts don't necessarily have to previously be registered. Once the owner defines a permission to give to an specific account, it will then be automatically registered to the system registries. Once the account then log in, the system will verify the credentials the account has and show the proper interface for each permission the account user has.
- **Document creation** The system must allow issuers to create a document and assign it to a document holder. Later, on the system must also allow only the issuer who created the document to assign attributes to the document. No modification to the assigned attributes can be done, if this case must be filled, then a new document must be created to reflect this.
- **Visualize documents** The system must allow the document holders to check their assigned documents and read the information attributes in it.
- **Generate token** The system must allow the document holder to create a token that will allow temporary exemption for a verifier to ascertain the document data.
- **Verification** The system must allow a verifier ascertain document claims which are presented by the document holder. For this, the verifier is required to present the token generated by the document holder.

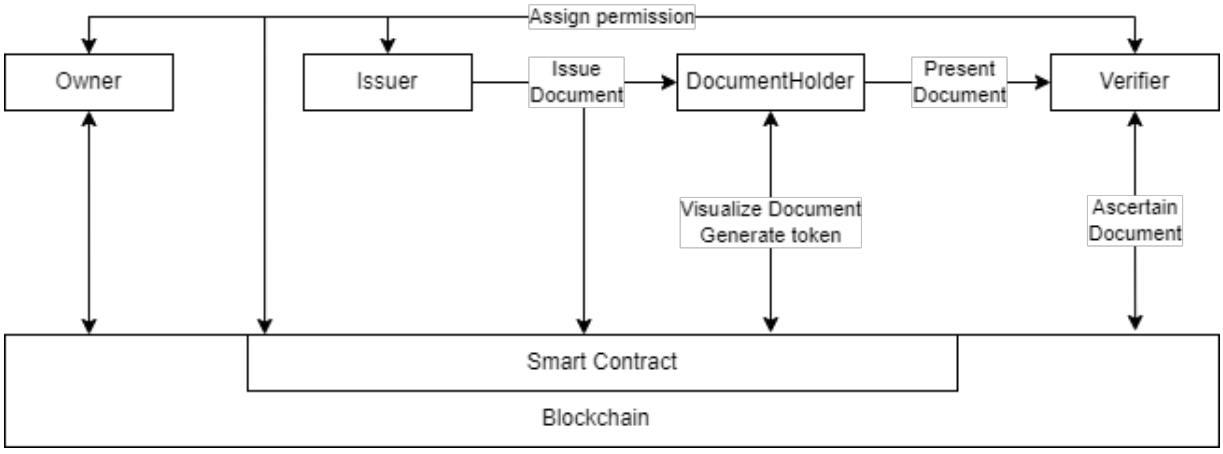

Figure 5.1: Users and Operations

5.3. ARCHITECTURE

Next, the defined architecture and how components interact with each other will be defined. For the study case implementation we opted do have a simplified scenario to have a better control on the pipeline once we get to the validation of it. The components forming the architecture are the following, as seen in Figure 5.2:

- **Ethereum** The blockchain in charge of holding the smart contract and storing the information. However, for the study case, a test Ethereum blockchain was used. This particular case used Ganache (which is part of Truffle which was mentioned in Table 3.5) to deploy a personal Ethereum blockchain that would allow for easier testing.
- **Document Identifier Smart Contract** The smart contract holding the operations logic. The system users would interact with the blockchain through specific public functions exposed in the smart contract.
- **MetaMask** A software wallet used to interact with the Ethereum blockchain. This allows the users to access a Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications [110].
- **Document Identifier Front End** A web application creating using react. This application is in charge of connecting to the MetaMask wallet to identify the user to the smart contract, which then is used to authorize the user to the available functions. Also in charge of the visualization logic for all the operations in the smart contract.
- **Browser** The medium through which the system users will be able to access the Document Identifier front end.

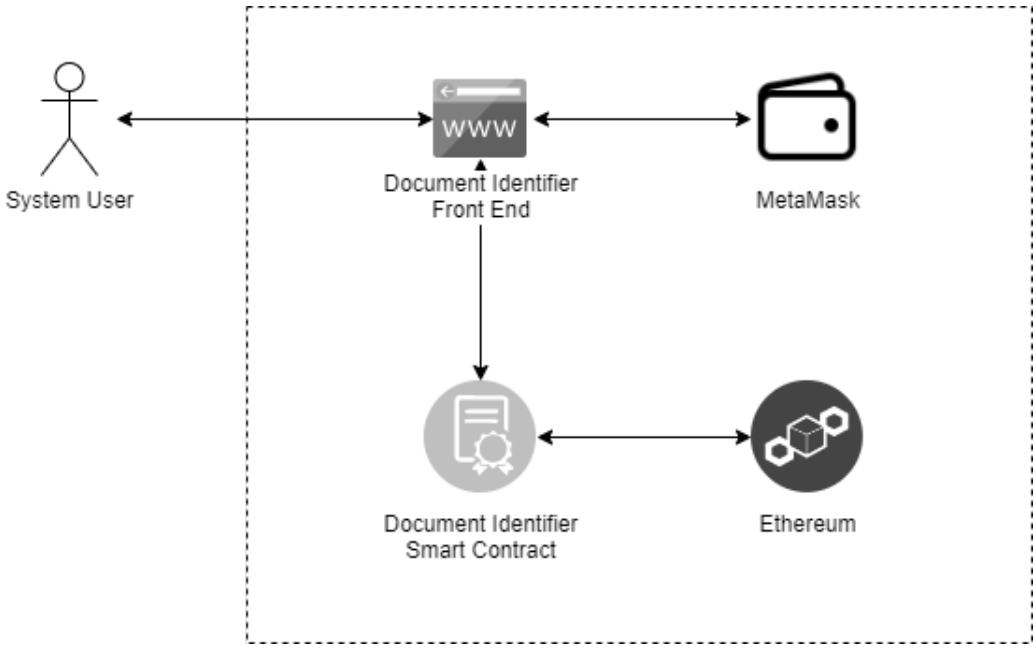

Figure 5.2: Implemented Architecture

5.3.1. Document Identifier Smart Contract

The most important component for the application is the Document Identifier Smart Contract (DISC). This smart contract is a contract that acts as a storage for the the system. It is involved in the creation, reading and updating operations of the identities or structures defined for the operation. This smart contracts takes the information from MetaMask to obtain the identity, and therefore their authorizations. This smart contract also provides the verification mechanism with which a verifier can ascertain the document provided by a document holder.

Structures

The structures or structs are the identities defined in the contract, set for a general use case scenarios. This could be simplified if applied to an specific subset of documents instead of a generalized case. The structures required in the system were defined as followed and their relationship can be seen in Figure 5.3 (actual implementation can be found in Figure A.1):

- **Issuer** The issuer struct contains the information required to identify an issuer. The struct holds the information regarding an ID for which the issuer can be identified and a name to identify the issuer.
- **DocumentHolder** This struct holds the information for the document holder. It includes a unique identifier, a name, the token information with which the document holder allows a verifier to check on the information and the documents information. The token information includes the actual token, an expiration for the token as this token is just a temporary token and a boolean for token used as the token was defined as one time use. Meanwhile, the documents are just a dictionary Document structs, with two extra properties a dictionary of document ids and a value for document size that were required for visualization.
- **Document** This struct holds the information assigned by an issuer to a document holder. The document information includes the issuer and the attributes in the document. The attributes defined in the document consist of a key and a value which allows for information safe guarding as the key needs to be known to be able to access the value.

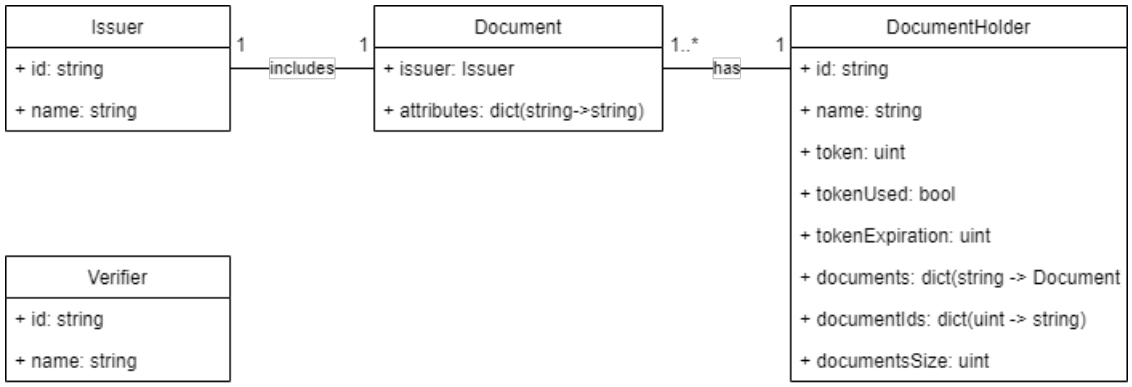

Figure 5.3: DISC Structures

- **Verifier** The verifier struct contains the information required to identify an verifier. The struct holds the information regarding an ID for which the verifier can be identified and a name to identify the verifier.

Functions

The functions are the endpoints available for the front end to connect to the smart contract. All the functions validate for proper authorization to allow for only the previously assigned accounts to execute them. The available functions in the smart contract to obtain the desired functionality are defined as following:

- **createIssuer** Function executed only by an account set as owner. The function receives the address of the issuer account, with the id and name that will be associated with this account. This will register the issuer inside the issuer list and emit an event to the front end once the operation has successfully finalized to be able to refresh the web application according to the information received.
- **createDocumentHolder** Function executed only by a issuer registered with the previous function. The function receives the address of the document holder account, with the id and name that will be associated with this account. The account is generated with a expired and marked as used token as a default. The document holder entity will also have an empty document dictionary. The account is finally added to the dictionary of document holders for the whole system.
- **createDocument** This function is only executed by an issuer. The functions received the address for the document holder, which has to have been previously been registered using the function above. It will then create the base document struct and add it to the document list for the document holder.
- **createDocumentAttribute** This function can only be executed by an issuer. The functions receives the address of the document holder and the document id. This is then used easily identify the document inside associated document holder list. A verification will be done on the issuer, as only the issuer who created the document will be able to add attributes to the document.
- **createVerifier** Function executed only by an account set as owner. The function receives the address of the verifier account, with the id and name that will be associated with this account. This will register the verifier inside the verifier list and emit an event to the front end once the operation has successfully finalized to be able to refresh the web application according to the information received.
- **refreshToken** This function can only be executed by a previously registered document holder. This will allow the document holder to generate a new token that the verifier can input as permission to allow document validation. This function will generate a new token, change the token used flag to false and give a expiration date for the token. The expiration date is set for 5 minutes in current implementation.
- **verifyDocument** The function holding the logic for verifiers to ascertain the attributes shown by a document holder from one of documents issued. It receives the document holder address, an array of attributes with keys, values and comparison sign, and token.

The function will then first validate the token and later take the input received to calculate whether the document is valid with the given information. The only value returned to the verifier is whether the result is valid or not. This is to ensure no information is leaked from the document holder.

5.3.2. MetaMask

As mentioned before, MetaMask is an established wallet that allows users to connect to their Ethereum accounts through a browser extension. This is exactly what was required for the web application to allow for authentication process. For this wallet, the user would connect to the wallet through their browser using a set of words as password. Whenever the application would require connection to the smart contract, the plugin would then automatically signs all transactions and pop up a confirmation window when you make a payment to get user approval. The following diagram shows the flow when using MetaMask:

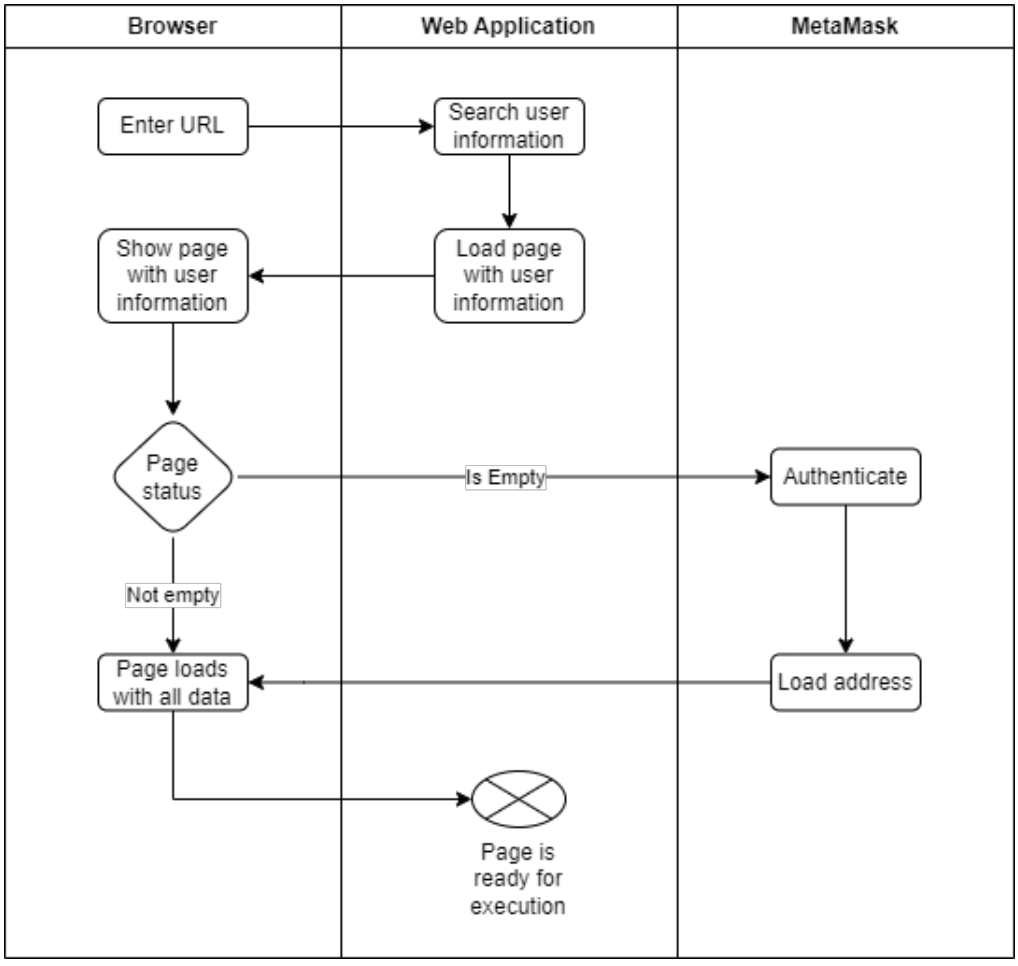

Figure 5.4: MetaMask interaction flow

CHAPTER 6

Framework Validation

This chapter describes the tools and steps taken to execute the framework proposal. These covers the steps from Continuous Integration up to Continuous Deployment. The objective here was to make sure that the proposed framework was able to be executed utilizing commercial and/or open-source tools available.

6.1. TOOLS

Multiple tools were utilized to generate the CI/CD pipeline. There are many available as seen in Table 3.5 and 3.6, however the following tools were the ones selected:

- **GitHub** is the application selected to host the private Git repository. DevOps connects directly to GitHub account using the GitHub app for authentication and to later be authorized to perform the required actions by the pipeline. The repository holds the react application, the solidity smart contract and pipeline configuration for CI/CD in it.
- **Azure DevOps** or DevOps is the cloud automation tool selected to work for pipeline orchestration. A pipeline will then be a list of tasks, each of which comprises one or more steps. Each execution that occurs in the pipeline is called a job. The tasks done in the pipeline are configured in a .yml file with predefined tasks and personalized tasks. These steps are then triggered to build whenever a change is pushed to the repository using the connection made to GitHub. There is also the possibility to manually trigger the execution of the steps. It also holds the dashboard for visualization of the process.
- **Agent** is the container or the machine where the steps are executed. The agent for the scenario is a local machine that hosts the Ganache installation for simplicity in the connection. The other requirements for the pipeline to execute properly will be downloaded and prepared by the agent.
- **SolHint** is an open source project for linting Solidity code. The linter provides both security and style guide validations for the contract, making it ideal for static code analysis of the smart contracts.
- **Ganache** is the private Ethereum blockchain deployed in the agent machine for testing. The blockchain had 10 addresses with 100 ETH for testing. The repository files had the configuration for the connection to Ganache.

6.2. TASKS

The as mentioned above in Section 6.1, the *.yml file used by the repository holds all the configuration for the process. These steps are prepared to go into any agent and be able to compile and build the project without issues. Therefore, we can conclude there are 2 types of steps: preparation steps and execution steps. The preparation steps involve preparing the machine for steps after, these involve installation and configuration.

1. **Checkout Code** This task is not defined in the *.yml file, it instead is an automatic step created by the DevOps pipeline where it connects to GitHub and pulls the code to the agent.
2. **Install NodeJs** Run with predefined task: NodeTool@0, this task receives as input the Node.js version and installs it in the agent in a temporary folder, exclusive to the current job execution.
3. **Install NPM Dependencies** Runs as a personalized task, this task only includes one step. It is installing all the dependencies required by the project to compile and build.
4. **Configure Linter** Runs as a personalized task, it involves 2 steps: installation of SolHint and creating the basic rule set for SolHint to run.
5. **Run Linter** This is a personalized task in charge of executing SolHint on the smart contract. The information resulting from SolHint execution is saved as a variable in the pipeline to be used in a future task. The task is configured to keep going even when an error occurs.
6. **Add GitHub comment** This is a predefined task that takes as input the output from the previous step. As long as the variable with the linter information is not null, then the task GitHubComment@0 would execute. This task adds a commit to GitHub adding a comment to the last pull where it inputs what SolHint detected during its run.
7. **Run Unit Tests** This runs as a personalized task with only one step. This step is running the prepared unit tests for functionality check. The smart contract is deployed in Ganache to be able to access its functions. The unit tests are executed using Javascript. If all unit tests assertions are correct, then it moves to the next step.
8. **Compile smart contract** This runs as a personalized tasks with a single step. It involves compiling the smart contract and leaving the files ready for deployment into the blokchain.
9. **Archiving contract** Final step of the pipeline, this takes the files resulting from the previous task and creating a zip with everything. This file is archived in the agent in an specified archiving route.

The actual definition of the pipeline can be seen in the figure located in Figure A.2.

CHAPTER 7
Discussion

This section describes some implementation issues that require attention and consideration when migrating from traditional software to a smart contract based application, which would lead to the application of this framework.

7.0.1. Secure deployment and integration

This segment describes implementation issues to consider to make a secure deployment and integration of smart contracts.

1. Define, design, and implement a mechanism to allow existing software to request/execute a smart contract method. Given that a complete overhaul of a software architecture is expensive, and not accounting for the cases where specific technologies do not allow for this to be done, the goal would be to reutilize as much as possible from current software. A mechanism that would connect current software with a smart contract with minimal change is however out of scope for the proposed framework.
2. Define, design, and implement a mechanism to authenticate and authorize requests to access resources in the smart contract from existing software. Such a mechanism should correspond to identities created in the existing software to accounts from the blockchain. This part would help extend the authorization model to control what identities may or may not access resources deployed in the implementation or update of smart contracts. Authentications or authorizations should not be lost under any circumstance for a system. If such case exists where a migration should occur, the deployment step of the framework should consider any kind of migration on the authentication and authorization model to guarantee data is kept.
3. Define, design, and implement a mechanism to properly handle/integrate the different versions of a smart contract deployed into the blockchain. Since a contract version is unavailable for modification after deployment, such a mechanism should help direct existing software requests to the smart contract's correct version. This mechanism would act as proxy, which mediates requests from existing software to the smart contract. When deployment of the smart contract being updated is executed, the deployment of this proxy should also be updated. Whether this mechanism is another smart contract, or another

type of tool is left to the implementer, however it is recommended to find an automated way according the implementation.

7.0.2. Secure monitoring

This segment describes implementation issues regarding the monitoring of smart contracts, as the transactions occurring on execution time increase the need for proper monitoring and analysis. Sometimes the information required to be monitored in a smart contract is not available depending on the implementation. One way to accomplish monitoring is to look at all transactions of the contract, however that may be insufficient, as message calls between contracts are not recorded in the blockchain. A monitoring mechanism should:

1. Define, design, and implement metrics to measure events related to the operation of a smart contract. An event is a convenient tool given by smart contracts to record executions in the contract. Events that were emitted stay in the blockchain along with the other contract data and are available for future audits. A mechanism should be available to be constantly seeking this data to transform it into visual form. This would work as feedback for the different team working on implementation.
2. Define, design, and implement a mechanism to utilize the data from the smart-contract operation. Such a mechanism should access the data extracted mentioned in the previous point. This data can then be given as machine models which can be fed and deployed to measure the previously defined metrics. The objective of this mechanism is to make this data available in visual form for the different teams involved in the implementation.

CHAPTER 8

Conclusion and Future Work

DevOps is a proven development strategy applied to traditional software development that shows high potential for the blockchain smart contracts development cycle. As shown in our framework, applying these same principles is considered with extra precautions to handle the nature of smart contracts. The strategy is created with overall security in mind while associating tools with specific activities to make it easier to identify the step. It also has the benefit of contributing to both repeatability and portability of the framework, a capability we consider will be desired by many organizations for security and business purposes.

In the case of our implementation for the test case, it was done entirely with specific tools in mind. overcome this limitation. Further work is required to demonstrate the mentioned repeatability and portability of our framework using more tools to prove these characteristics. However, there is confidence the changes required for the application in other tools will be exclusively on the tool implementation rather than the steps required. With the goal of security of both the framework and the smart contracts, we expect the integrated vulnerability checks to create a much higher awareness of security issues during development. This, in theory, will force developers to fix vulnerabilities as soon as possible.

As blockchain technology continues to move forward, so will the requirement to build software around them. However, testing the released smart contracts can not be reduced as we consider it a critical practice that can not and should not be disregarded. In this regard, after stating the best practices for smart contract development and the overall steps, we have found some critical steps that require further research. This future research should evaluate two main points: first, the strategy to identify the points used to evaluate the output of the further analysis, depending on the blockchain, to specify whether we can consider the smart contract steps of the CT phase as successful. Second, to automate the monitoring of the smart contracts as the current proposal is a task to be done manually. Ultimately, the teams integrating DevOps into their practice will face challenges to ensure reliable and secure blockchain smart contracts, to which we place our contribution.

Bibliography

[1] M. Rizky and D. Sulistiyo, "Implementation of continuous integration and continuous delivery (ci/cd) on automatic performance testing," *9th International Conference on Information and Communication Technology*, 2021. DOI: 10.1109/ICoICT52021.2021.9527496.

[2] L. de Aguiar Monteiro, "A proposal to systematize introducing devops into the software development process," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 269–271. DOI: 10.1109/ICSE-Companion52605.2021.00124.

[3] W. De Kort, *DevOps on the Microsoft Stack*, 1st ed. Berkeley, CA, USA: Apress Berkley, 2016, vol. 1, ISBN: 978-1-4842-1446-6. [Online]. Available: https://doi.org/10.1007/978-1-4842-1446-6 (visited on 06/21/2022).

[4] Q. Liao, "Modelling ci/cd pipeline through agent-based simulation," *IEEE International Symposium on Software Reliability Engineering Workshops*, 2020. DOI: 10.1109/ISSREW51248.2020.00059.

[5] N. Chen, S. C. H. Hoi, and X. Xiao, "Software process evaluation: A machine learning framework with application to defect management process," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1531–1564, Dec. 2014, ISSN: 1573-7616. DOI: 10.1007/s10664-013-9254-z. [Online]. Available: https://doi.org/10.1007/s10664-013-9254-z (visited on 06/21/2022).

[6] A. Nogueira *et al.*, "Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques," *International Conference on the Quality of Information and Communications Technology*, 2018. DOI: 10.1109/QUATIC.2018.00050.

[7] M. Aldeen *et al.*, "Adopting continuous integeration and continuous delivery for small teams," *International Conference on Computer, Control, Electrical, and Electronics Engineering*, 2019.

[8] T. Tegeler, F. Gossen, and B. Steffen, "A model-driven approach to continuous practices for modern cloud-based web applications," *IEEE Access*, 2019.

[9] M. Virmani, "Understanding devops & bridging the gap from continuous integration to continuous delivery," in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015, pp. 78–82. DOI: 10.1109/INTECH.2015.7173368.

[10] F. Erich, "Devops is simply interaction between development and operations," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software*

Production and Deployment, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds., Cham: Springer International Publishing, 2019, pp. 89–99, ISBN: 978-3-030-06019-0.

[11] S. Throner *et al.*, "An advanced devops environment for microservice-based applications," *IEEE International Conference on Service-Oriented System Engineering*, 2021. DOI: 10.1109/SOSE52839.2021.00020.

[12] S. Wang *et al.*, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, Nov. 2019, ISSN: 2168-2232. DOI: 10.1109/TSMC.2019.2895123.

[13] N. Szabo, *Nick Szabo – Smart Contracts: Building Blocks for Digital Markets*, 1996. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (visited on 06/21/2022).

[14] F. Zampetti *et al.*, "Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study," *IEEE International Conference on Software Maintenance and Evolution*, 2021. DOI: DOI:10.1109/ICSME52107.2021.00048.

[15] K. B. KIM and J. LEE, "Automated generation of test cases for smart contract security analyzers," *IEEE Access*, 2020. DOI: 10.1109/ACCESS.2020.3039990.

[16] M. Wöhrer and U. Zdun, "Devops for ethereum blockchain smart contracts," in *2021 IEEE International Conference on Blockchain (Blockchain)*, 2021, pp. 244–251. DOI: 10.1109/Blockchain53845.2021.00040.

[17] V. Lenarduzzi *et al.*, "Blockchain applications for agile methodologies," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, ser. XP '18, Porto, Portugal: Association for Computing Machinery, 2018, ISBN: 9781450364225. DOI: 10.1145/3234152.3234155. [Online]. Available: https://doi.org/10.1145/3234152.3234155.

[18] L. Marchesi, M. Marchesi, and R. Tonelli, "Abcde—agile block chain dapp engineering," *Blockchain: Research and Applications*, vol. 1, no. 1, p. 100 002, 2020, ISSN: 2096-7209. DOI: https://doi.org/10.1016/j.bcra.2020.100002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096720920300026.

[19] G. Al-Mazrouai and S. Sudevan, "Managing blockchain projects with agile methodology," in *Proceedings of 6th International Conference on Big Data and Cloud Computing Challenges: ICBCC 2019, UMKC, Kansas City, USA*, V. Vijayakumar *et al.*, Eds. Singapore: Springer Singapore, 2020, pp. 179–187, ISBN: 978-981-32-9889-7. DOI: 10.1007/978-981-32-9889-7_14. [Online]. Available: https://doi.org/10.1007/978-981-32-9889-7_14.

[20] J. Shah, D. Dubaria, and J. Widhalm, "A survey of devops tools for networking," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2018, pp. 185–188. DOI: 10.1109/UEMCON.2018.8796814.

[21] P. Agrawal and N. Rawat, "Devops, a new approach to cloud development & testing," in *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, vol. 1, 2019, pp. 1–4. DOI: 10.1109/ICICT46931.2019.8977662.

[22] C. Pang, A. Hindle, and D. Barbosa, "Understanding devops education with grounded theory," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2020, pp. 260–261.

[23] A. Wahaballa *et al.*, "Toward unified devops model," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2015, pp. 211–214. DOI: 10.1109/ICSESS.2015.7339039.

[24] J. Mahboob and J. Coffman, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE 11th Annual Computing and Communication Workshop and Conference*, 2021. DOI: 10.1109/CCWC51732.2021.9376148.

[25] N. Railić and M. Savić, "Architecting continuous integration and continuous deployment for microservice architecture," *20th International Symposium INFOTEH-JAHORINA*, 2021. DOI: 10.1109/INFOTEH51037.2021.9400696.

[26] J. Shah, D. Dubaria, and J. Widhalm, "A survey of devops tools for networking," *IEEE Access*, 2018.

[27] J. Shah, D. Dubaria, and J. Widhalm, "Distributing parallel virtual image application using continuous integrity/continuous delivery based on cloud infrastructure," *The 8th International Conference on Cyber and IT Service Management*, 2020.

[28] A. Agarwal, S. Gupta, and T. Choudhury, "Continuous and integrated software development using devops," *International Conference on Advances in Computing and Communication Engineering*, 2018.

[29] C. Fayollas, H. Bonnin, and O. Flebus, "Safeops: A concept of continuous safety," *16th European Dependable Computing Conference*, 2020. DOI: 10.1109/EDCC51268.2020.00020.

[30] T. Düllmann, C. Paule, and A. van Hoorn, "Exploiting devops practices for dependable and secure continuous delivery pipelines," *ACM/IEEE 4th International Workshop on Rapid Continuous Software Engineering*, 2018. DOI: 10.1145/3194760.3194763.

[31] N. A. A. Khleel and N. Károly, "Comparison of version control system tools," *Multidiszciplináris Tudományok*, vol. 10, no. 3, pp. 61–69, Jun. 2020, ISSN: 2786-1465. DOI: 10.35925/j.multi.2020.3.7. [Online]. Available: https://ojs.uni-miskolc.hu/index.php/multi/article/view/441 (visited on 06/28/2022).

[32] C. Singh *et al.*, "Comparison of different ci/cd tools integrated with cloud platform," *9th International Conference on Cloud Computing, Data Science and Engineering*, 2019.

[33] *Azure pipelines*, https://azure.microsoft.com/en-us/services/devops/pipelines/, 2022.

[34] *Devops and ci/cd on google cloud explained*, https://cloud.google.com/blog/topics/developers-practitioners/devops-and-cicd-google-cloud-explained, 2021.

[35] *Alibaba cloud devops pipeline (flow)*, https://www.alibabacloud.com/product/apsara-deveops/flow, 2022.

[36] *Ibm cloud continuous delivery*, https://www.ibm.com/cloud/continuous-delivery, 2022.

[37] *Devops and aws*, https://aws.amazon.com/devops/, 2022.

[38] *Cloud-native ci/cd on red hat openshift*, https://cloud.redhat.com/learn/topics/ci-cd, 2022.

[39] G. Chen *et al.*, "Blockchain-based cyber security and advanced distribution in smart grid," *IEEE 4th International Conference on Electronics Technology*, 2021. DOI: 10.1109/ICET51757.2021.9451130.

[40] T. Brandstatter *et al.*, "Characterizing efficiency optimizations in solidity smart contracts," *IEEE International Conference on Blockchain*, 2020. DOI: 10.1109/Blockchain50366.2020.00042.

[41] S. Murugan and S. Kris, "A survey on smart contract platforms and features," *7th International Conference on Advanced Computing and Communication Systems*, 2021. DOI: 10.1109/ICACCS51430.2021.9441970.

[42] Y. Li, "Finding concurrency exploits on smart contracts," *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings*, 2019. DOI: 10.1109/ICSE-Companion.2019.00061.

[43] J. Chen, "Finding ethereum smart contracts security issues by comparing history versions," *35th IEEE/ACM International Conference on Automated Software Engineering*, 2020. DOI: 10.1145/3324884.3418923.

[44] A. Dika and M. Nowostawski, "Security vulnerabilities in ethereum smart contracts," *IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics*, 2018. DOI: 10.1109/Cybermatics_2018.2018.00182.

[45] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: Attacks and protections," *IEEE Access*, 2020. DOI: 10.1109/ACCESS.2020.2970495.

[46] G. Destefanis *et al.*, "Smart contracts vulnerabilities: A call for blockchain software engineering?" *1st International Workshop on Blockchain Oriented Software Engineering*, 2018.

[47] S. Richards, *Scaling*, https://ethereum.org/en/developers/docs/scaling/, 2022.

[48] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, Porto, Portugal: Association for Computing Machinery, 2018, ISBN: 9781450355841. DOI: 10.1145/3190508.3190538. [Online]. Available: https://doi.org/10.1145/3190508.3190538.

[49] H. Pervez *et al.*, "A comparative analysis of dag-based blockchain architectures," in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, 2018, pp. 27–34. DOI: 10.1109/ICOSST.2018.8632193.

[50] N. Kraus, K. Kraus, and O. Manzhura, "Newest Digital Technology in Management of National Economic System," Atlantis Press, Sep. 2019, pp. 1–5, ISBN: 978-94-6252-790-4. DOI: 10.2991/smtesm-19.2019.1. [Online]. Available: https://www.atlantis-press.com/proceedings/smtesm-19/125917609 (visited on 06/28/2022).

[51] M. Lokhava *et al.*, "Fast and secure global payments with stellar," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19, Huntsville, Ontario, Canada: Association for Computing Machinery, 2019, pp. 80–96, ISBN: 9781450368735. DOI: 10.1145/3341301.3359636. [Online]. Available: https://doi.org/10.1145/3341301.3359636.

[52] P. Katsiampa, "An empirical investigation of volatility dynamics in the cryptocurrency market," *Research in International Business and Finance*, vol. 50, pp. 322–335, 2019, ISSN: 0275-5319. DOI: https://doi.org/10.1016/j.ribaf.2019.06.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0275531919300637.

[53] W. Song *et al.*, "EOS.IO blockchain data analysis," *The Journal of Supercomputing*, vol. 78, no. 4, pp. 5974–6005, Mar. 2022, ISSN: 1573-0484. DOI: 10.1007/s11227-021-04090-y. [Online]. Available: https://doi.org/10.1007/s11227-021-04090-y (visited on 06/28/2022).

[54] M. Benji and M. Sindhu, "A Study on the Corda and Ripple Blockchain Platforms," in *Advances in Big Data and Cloud Computing*, J. D. Peter, A. H. Alavi, and B. Javadi, Eds., Singapore: Springer Singapore, 2019, pp. 179–187, ISBN: 978-981-13-1882-5.

[55] J.-W. Liao *et al.*, "Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing," *Sixth International Conference on Internet of Things: Systems, Management and Security*, 2019.

[56] A. López *et al.*, "An analysis of smart contracts security threats alongside existing solutions," *Entropy*, 2020. DOI: 10.3390/e22020203.

[57] S. Akca, A. Rajan, and C. Peng, "Solanalyser: A framework for analysing and testing smart contracts," *26th Asia-Pacific Software Engineering Conference*, 2019.

[58] Q. Ashfaq, R. Khan, and S. Farooq, "A comparative analysis of static code analysis tools that check java code adherence to java coding standards," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, 2019, pp. 98–103. DOI: 10.1109/C-CODE.2019.8681007.

[59] S. Kim, Y. Chul, and Y. Park, "Static code analysis in continuous integration - agile and rule-compliant development," *Wireless Personal Communications*, 2016.

[60] A. Paul, "More software safety a static analysis tools perspective," *ATZelectronics worldwide*, 2017.

[61] M. Hermeling, "Static code analysis in continuous integration - agile and rule-compliant development," *ATZelectronics worldwide*, 2019.

[62] R. Kumar, K. Indraveni, and A. K. Goel, "Automation of detection of security vulnerabilities in web services using dynamic analysis," in *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*, 2014, pp. 334–336. DOI: 10.1109/ICITST.2014.7038832.

[63] O. Zaazaa and H. El Bakkali, "Dynamic vulnerability detection approaches and tools: State of the art," in *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*, 2020, pp. 1–6. DOI: 10.1109/ICDS50568.2020.9268686.

[64] T. Theunissen., S. Hoppenbrouwers., and S. Overbeek., "In continuous software development, tools are the message for documentation," in *Proceedings of the 23rd International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, INSTICC, SciTePress, 2021, pp. 153–164, ISBN: 978-989-758-509-8. DOI: 10.5220/0010367901530164.

[65] *Remix*, https://github.com/ethereum/remix-project, 2022.

[66] *Loom network*, https://github.com/loomnetwork, 2022.

[67] *Chainide*, https://chainide.gitbook.io/chainide-english-1/, 2022.

[68] *Replit*, https://github.com/replit, 2022.

[69] *Visual studio code*, https://github.com/microsoft/vscode, 2022.

[70] *What's new in intellij idea 2022.1*, https://www.jetbrains.com/idea/whatsnew/, 2022.

[71] *Remix desktop*, https://github.com/ethereum/remix-desktop, 2022.

[72] *Truffle*, https://github.com/trufflesuite/truffle, 2022.

[73] *Hyperledger composer*, https://github.com/hyperledger-archives/composer, 2019.

[74] *Software and sdks*, https://developers.stellar.org/docs/software-and-sdks/, 2022.

[75] *Eos studio releases*, https://github.com/ObsidianLabs/EOS-Studio-Releases, 2020.

[76] *Eos studio desktop*, https://github.com/ObsidianLabs/EOS-Studio-Desktop, 2019.

[77] *Eosio web ide*, https://github.com/EOSIO/eosio-web-ide, 2020.
[78] *Zeus ide*, https://github.com/liquidapps-io/zeus-ide, 2020.
[79] *Oyente*, https://github.com/enzymefinance/oyente, 2020.
[80] *Solgraph*, https://github.com/raineorshine/solgraph, 2019.
[81] *Madmax*, https://github.com/nevillegrech/MadMax, 2021.
[82] *Manticore*, https://github.com/trailofbits/manticore, 2022.
[83] *Mythril*, https://github.com/ConsenSys/mythril, 2022.
[84] *Contractlarva*, https://github.com/gordonpace/contractLarva, 2022.
[85] *Solmet solidity parser*, https://github.com/chicxurug/SolMet-Solidity-parser, 2020.
[86] *Vandal*, https://github.com/usyd-blockchain/vandal, 2020.
[87] *Securify v2.0*, https://github.com/eth-sri/securify2, 2021.
[88] *Slither*, https://github.com/crytic/slither, 2021.
[89] *Ethlint*, https://github.com/duaraghav8/Ethlint, 2021.
[90] *Revive-cc*, https://github.com/sivachokkapu/revive-cc, 2020.
[91] *Blockchain analyzer*, https://github.com/hyperledger-labs/blockchain-analyzer, 2020.
[92] *Chaincode analyzer*, https://github.com/FujitsuLaboratories/ChaincodeAnalyzer, 2020.
[93] A. Grüner, A. Mühle, and C. Meinel, "Atib: Design and evaluation of an architecture for brokered self-sovereign identity integration and trust-enhancing attribute aggregation for service provider," *IEEE Access*, vol. 9, pp. 138 553–138 570, 2021. DOI: 10.1109/ACCESS.2021.3116095.
[94] E. Bandara *et al.*, "A blockchain and self-sovereign identity empowered digital identity platform," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1–7. DOI: 10.1109/ICCCN52240.2021.9522184.
[95] R. Rana, R. N. Zaeem, and K. S. Barber, "An assessment of blockchain identity solutions: Minimizing risk and liability of authentication," in *IEEE/WIC/ACM International Conference on Web Intelligence*, ser. WI '19, Thessaloniki, Greece: Association for Computing Machinery, 2019, pp. 26–33, ISBN: 9781450369343. DOI: 10.1145/3350546.3352497. [Online]. Available: https://doi.org/10.1145/3350546.3352497.
[96] W. Song *et al.*, "Self-sovereign identity and user control for privacy-preserving contact tracing," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, ser. WI-IAT '21, Melbourne, VIC, Australia: Association for Computing Machinery, 2022, pp. 438–445, ISBN: 9781450391153. DOI: 10.1145/3486622.3493914. [Online]. Available: https://doi.org/10.1145/3486622.3493914.
[97] S. Mahula, E. Tan, and J. Crompvoets, "With blockchain or not? opportunities and challenges of self-sovereign identity implementation in public administration: Lessons from the belgian case," in *DG.O2021: The 22nd Annual International Conference on Digital Government Research*, ser. DG.O'21, Omaha, NE, USA: Association for Computing Machinery, 2021, pp. 495–504, ISBN: 9781450384926. DOI: 10.1145/3463677.3463705. [Online]. Available: https://doi.org/10.1145/3463677.3463705.
[98] Y. Liu *et al.*, "Design patterns for blockchain-based self-sovereign identity," *CoRR*, vol. abs/2005.12112, 2020. arXiv: 2005.12112. [Online]. Available: https://arxiv.org/abs/2005.12112.

[99] R. Nokhbeh Zaeem *et al.*, "Blockchain-based self-sovereign identity: Survey, requirements, use-cases, and comparative study," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, ser. WI-IAT '21, Melbourne, VIC, Australia: Association for Computing Machinery, 2022, pp. 128–135, ISBN: 9781450391153. DOI: 10.1145/3486622.3493917. [Online]. Available: https://doi.org/10.1145/3486622.3493917.

[100] M. ABDELRAZIG ABUBAKAR *et al.*, "Blockchain-based identity and authentication scheme for mqtt protocol," in *2021 The 3rd International Conference on Blockchain Technology*, ser. ICBCT '21, Shanghai, China: Association for Computing Machinery, 2021, pp. 73–81, ISBN: 9781450389624. DOI: 10.1145/3460537.3460549. [Online]. Available: https://doi.org/10.1145/3460537.3460549.

[101] G. Kondova and J. Erbguth, "Self-sovereign identity on public blockchains and the gdpr," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20, Brno, Czech Republic: Association for Computing Machinery, 2020, pp. 342–345, ISBN: 9781450368667. DOI: 10.1145/3341105.3374066. [Online]. Available: https://doi.org/10.1145/3341105.3374066.

[102] N. Naik and P. Jenkins, "Sovrin network for decentralized digital identity: Analysing a self-sovereign identity system based on distributed ledger technology," in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, 2021, pp. 1–7. DOI: 10.1109/ISSE51541.2021.9582551.

[103] N. Naik and P. Jenkins, "Uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain," in *2020 IEEE International Symposium on Systems Engineering (ISSE)*, 2020, pp. 1–7. DOI: 10.1109/ISSE49799.2020.9272223.

[104] S. Porru *et al.*, "Blockchain-oriented software engineering: Challenges and new directions," in *Proceedings of the 39th International Conference on Software Engineering Companion*, ser. ICSE-C '17, Buenos Aires, Argentina: IEEE Press, 2017, pp. 169–171, ISBN: 9781538615898. DOI: 10.1109/ICSE-C.2017.142. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.142.

[105] P. Chakraborty *et al.*, "Understanding the software development practices of blockchain projects: A survey," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18, Oulu, Finland: Association for Computing Machinery, 2018, ISBN: 9781450358231. DOI: 10.1145/3239235.3240298. [Online]. Available: https://doi.org/10.1145/3239235.3240298.

[106] H. K. Brar and P. J. Kaur, "Differentiating integration testing and unit testing," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 796–798.

[107] T. Górski, "Continuous delivery of blockchain distributed applications," *Sensors*, vol. 22, no. 1, 2022, ISSN: 1424-8220. DOI: 10.3390/s22010128. [Online]. Available: https://www.mdpi.com/1424-8220/22/1/128.

[108] S. N. Khan *et al.*, "Blockchain smart contracts: Applications, challenges, and future trends," *Peer-to-peer networking and applications*, vol. 14, no. 5, pp. 2901–2925, 2021, ISSN:

1936-6442. DOI: 10.1007/s12083-021-01127-0. [Online]. Available: https://doi.org/10.1007/s12083-021-01127-0.

[109] S. Azzopardi, J. Ellul, and G. J. Pace, "Monitoring smart contracts: Contractlarva and open challenges beyond," in *Runtime Verification*, C. Colombo and M. Leucker, Eds., Cham: Springer International Publishing, 2018, pp. 113–137, ISBN: 978-3-030-03769-7.

[110] *Metamask*, https://metamask.io/, 2022.

APPENDIX A

Code

```
contract DocumentIdentifier {

    struct Issuer {
        string id;
        string name;
    }

    struct DocumentHolder {
        string id;
        string name;
        uint token;
        bool tokenUsed;
        uint tokenExpiration;
        mapping(string => Document) documents;
        mapping(uint => string) documentIds;
        uint documentsSize;
    }

    struct Document {
        Issuer issuer;
        mapping(string=>string) attributes;
    }

    struct Verifier {
        string id;
        string name;
    }
}
```

Figure A.1: Smart Contract Structures

```
steps:
Settings
- task: NodeTool@0
  inputs:
    versionSpec: '8.15.0'
  displayName: 'Install Node.js'

- script: |
    npm install
    npm run build
  displayName: 'Install NPM Dependencies'

- script: |
    npm install solhint
    npx solhint --init
  displayName: 'Configure Linter'

- script: |
    npx solhint contracts/$(contractName).sol > linter-log.txt 2>&1
    if [ $? -ne 0 ]; then
      echo "##vso[task.setvariable variable=linterLog]$(cat linter-log.txt | sed '1,/> solhint ./d;/npm ERR!/,$
    fi
  continueOnError: true
  displayName: 'Run Linter'
Settings
- task: GitHubComment@0
  condition: and(eq(variables['Build.Reason'], 'PullRequest'), failed())
  inputs:
    gitHubConnection: 'alvarojry'
    repositoryName: '$(Build.Repository.Name)'
    comment: |
      **Linter Error Details:**
      ```
      $(linterLog)
      ```
  displayName: 'Add GitHub comment'

- script: |
    npx truffle test
  displayName: 'Run Unit Tests'

- script: |
    cd src
    npx truffle compile contracts/$(contractName).sol
  displayName: 'Compile smart contract'

Settings
- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: '$(System.DefaultWorkingDirectory)/src/abis/$(contractName).json'
    includeRootFolder: false
    archiveType: 'zip'
    archiveFile: '$(Build.ArtifactStagingDirectory)/$(contractName)-$(Build.BuildId).zip'
  displayName: 'Archiving contract $(contractName)'
```

Figure A.2: DevOps Pipeline Definition