**RIGA TECHNICAL UNIVERSITY**

Faculty of Computer Science, Information Technology and Energy

Riga Technical University

**Pavlo Nikolaiev**

Academic Bachelor Level Study Program “Computer Systems”

Student ID No 201ADB090

**Comparative analysis of blockchain automated functional testing tools**

**bachelor thesis: 5th semester report**

Scientific adviser Dr.sc.ing, associate professor

Egons Lavendelis

RIGA 2024

ABSTRACT

This thesis examines the role of automated testing in blockchain technology. It begins with an introduction to blockchain and its importance in various industries, emphasizing the need for robust testing methods to ensure reliability, security, and efficiency. The purpose of this thesis is to analyze the principles and methods of automated testing of blockchain-based systems. This research aims to use automated testing tools to address challenges related to smart contract testing, connectivity testing, data integrity, performance, security, and interoperability. This thesis provides case studies and practical information on the implementation of functional testing on real blockchain projects, demonstrating its effectiveness in identifying issues and improving blockchain performance. The obtained results highlight the importance of introducing automated testing in blockchain system development to reduce risks and increase overall effectiveness.

table of contents

[1 INTRODUCTION 3](#_Toc181604623)

[2. Automated Functional Testing Principles in Blockchain Technology 4](#_Toc181604624)

[2.1 Comprehensive Testing Approaches 4](#_Toc181604625)

[2.2 Blockchain automation testing tools 5](#_Toc181604626)

[2.3 Test Blockchain Applications principles 12](#_Toc181604627)

[3 Testing Blockchain automation testing tools 14](#_Toc181604628)

[3.1 Truffle 17](#_Toc181604629)

[3.2 Hardhat 24](#_Toc181604630)

[3.3 Foundry 28](#_Toc181604631)

LIST OF REFERENCES

1 INTRODUCTION

The statement of the problem: Blockchain solutions are gaining their popularity because in principle this technology can solve security and transparency issues. At the same time testing of distributed solutions is challanging both for manual and automated testing(IBM). The thesis will address the problem of choosing the best automated functional testing tool for blockchain. The developed recommendations/guidelines will help companies to choose a suitable tool for functional testing purposes in particular blockchain based projects.

The goal of the bachelor thesis: To compare existing automated functional testing tools widely used in industry for testing blockchain based solutions.

**The tasks of the bachelor thesis:**

* To analyze blockchain automated functional testing principles,
* To overview blockchain technologies in the context of testing,
* To analytical compare automated functional testing tools for blockchain technologies and to overview existing comparisons of these tools,
* To define the methodology for comparison of automated functional testing tools for blockchain technologies,
* To do experimental comparison of automated functional testing tools for blockchain technologies based on particular test scenarios,
* To provide guidelines or recommendations for selecting automated functional testing tools for blockchain technologies and demonstrate the use of recommendations/guidelines on realistic scenarios of tool selection.

**The justification of topicality:**

Blockchain technology has become the central space of technical landscape capturing the attention of industries, researchers, and global advance events (Dong & Khushnood & Li & Kamruzzaman , 2023).As organizations use blockchain technology for safe testing methodologies more and more becomes paramount to make sure reliability is guaranteed and functionality of this decentralized systems. The focus of this thesis on the "Comparative Analysis of Blockchain Automated Functional Testing Tools".

Rapid Evolution of Blockchain Technology: The blockchain technology is evolving fast way, as new technologies and emergence of new technologies. In this dynamic environment the selection automated function is become critical to implement in blockchain technologies (Andoni & Robu & Flynn & Abram & Geach & Jenkins & McCallum & Peacock , 2018).

2. Automated Functional Testing Principles in Blockchain Technology

Blockchain automation functional testing principles are critical for insuring the reliability , security and efficiency of blockchain-based systems(Lal & Marijan 2018). Automation testing in the context of blockchain refers to use software tools and scripts to verify the functionality different components within blockchain networks.

When author should test a blockchain solution he can use some testing tactics or types(Parasu & Reddy , 2019).

2.1 Comprehensive Testing Approaches

Integration Testing:Since blockchain is an ecosystem made up of different components, all these components need to be connected(Lal & Marijan , 2021). It is also important to connect to various APIs Test compatibility between components(Scheid & Hegnauer & Rodrigues & Stiller, 2020), (Parasu & Reddy , 2019).

Performance testing of blockchain is important from a digital perspective Transactions and transaction sizes are evaluated against block performance, or within the program is ready to be deployed to production. Other important and dependent parameters include network, transaction sequence of each node, transaction processing speed, user and system interfaces, and responsiveness required by smart contracts (Parasu & Reddy , 2019) , (Lal & Marijan , 2021).

Security testing(Algarni & Attaallah & Eassa, Khemakhem, 2023) **-** Determine whether they are vulnerable to attacks and whether authorization and authentication systems are functioning properly. Security testing also considers other important aspects such as confidentiality, integrity, non-denial of service, availability, etc. Security testing is especially important when hackers attack the identity layer of a blockchain application. Transactions executed when an identity layer hack is detected can't stop immediately. Therefore, security testing should be conducted to reveal all these potential identity layer hacking. Blockchain (cryptocurrency-based) application security testing also includes challenges or tests of wallet signature methods, private keys, consensus and other components algorithm and application platform dependencies.Another essential element of security testing is that it handles fraudulent transactions irreversible because transaction reversal is almost impossible in blockchain technology (Parasu & Reddy , 2019) , ( Lal & Marijan , 2021).

Last test it is Functional Testing**,** (Parasu & Reddy , 2019)itis useful in assessing the business environment and the effectiveness of application scenarios. The following are key implementation considerations functional testing in blockchain applications: Block size: Each block in the blockchain has a storage size in megabytes, which is the case Reduced from 36 MB to 1 MB for security reasons. Testers need to pay attention to the following scenarios. Which encryption technology should be used if more than 1MB of transaction data is stored per block (Ali and Nelson, 2016)?

Data transmission:

* Need to test data loss during transmission between blocks.
  1. Blockchain automation testing tools

Blockchain (Zheng & Xie & Dai & Chen and Wang , 2018),( Yli-Huumo & Ko & Choi & Park & Smolander , 2016) technology is a revolutionary technology that helps store data securely and transparently. Unlike centralized systems, blockchain operates on a peer-to-peer network, allowing multiple computers or nodes to maintain the same ledger. Transactions are divided into blocks, creating blocks, hence the name blockchain. To ensure the integrity and security of recorded data, encryption methods are used, making it difficult for anyone to intercept or manipulate the information contained in the forum. Automated blockchain testing tools (Lal & Marijan 2018) are critical to ensuring the quality and security of blockchain services. These tools allow developers to run tests, identify, and fix bugs, and ensure their software complies with industry standards.

Blockchain system exhibit (Staroletov & Galkin , 2022a) unique characteristics such as consensus mechanism, smart contracts, decentralized architecture. Traditional testing tools may not solve problems which can be started in blockchain applications. A specific comparative analysis of automated tools created for blockchain is needed for identity tools that can effectively navigate the complexities of distributed ledger technologies (Parasu & Reddy , 2019).

Testing of blockchain solutions creates some problems which are related to decentralized nature of the technology. Traditional testing methodologies may fall in issues related to security, consensus mechanism and smart contract functions. A comparative analysis will provide the weakness sides of different automated testing tools for solving specific challenges(Staroletov & Galkin , 2022a).

In the context of technology, blockchain has come an innovative force, translating industries, and reshaping the way people view and do business(Kimani a & Adams & Attah-Boakye & Ullah & Frecknall-Hughes & Kim , 2020). As this technology is adoptive at scale, the need for reliable testing methods to ensure the reliability, security, and robustness of blockchain-based systems will increase. Blockchain very important technology for example for security(Staroletov & Galkin , 2022b).

The reliability of blockchain is also not limited to its unique nature of being both decentralized and immutable, instead it utilizes additional mechanisms to increase its resilience and to protect itself against any form of security threats and attacks.The essence of blockchain, which is decentralization, installs the data on various nodes and so loss of one or more nodes does not introduce a single point of failure and blockage of any single entity to manipulate or control it. Therefore, this distributed structure avoids the possibility of the blockchain network’s being affected due to censorship and/or tamper.Besides, the blockchain has imperturbability characteristic as no single person or organization is reliable for data update, which results in the integrity and permanence of recorded data. Transactions are made confirmed and recorded on the blockchain only once a majority of the network's participants agree. Hence, the tampering or alteration of a transaction is almost impossible as long as the majority of the network participants stay in agreement. Being verifiable makes blockchain data very useful, since it enables the verification of transactions in an open and comprehensive way. (Parasu & Reddy , 2019). Hashing and digital signatures reduce the exposure of blockchain transactions to hacking and fraud thus keeping blockchain technology even more secure(Leng & Zhou & Zhao & Huang, 2020). Cryptohashes are the unique cryptographic digests generated afresh for blocks following specific hash functions. This aids in quick and accurate identification of any tampering of data(Leng & Zhou & Zhao & Huang, 2020). Digital signatures, made by applying public-key cryptography, can be used to identify the participants of the processes and also prove that conditionalities and transactions aren’t repudiated.Add on, most blockchain networks implement consensus mechanisms as either Proof of Work (PoW), or as the more popular of them, Proof of Stake (PoS) to allow for transaction validation and confirmation(Leng & Zhou & Zhao & Huang, 2020). The pools and businesses use some of the computing power for mining the blockchain of the network or the cryptocurrency they own or the one whose value is expected to increase. The consensus algorithms make sure that the valid transactions are only added to the blockchain(Leng & Zhou & Zhao & Huang, 2020). This reduces the risk of double spending and any other fraudulent activities.Beyond this, the development of cryptographic technology, such as the one by zero-knowledge proofs and others by multi-signature schemes, brings in the question of privacy and security within blockchain ecosystems. These strategies broaden the approaches for secure and verifiable transactions that do not reveal the information(Leng & Zhou & Zhao & Huang, 2020). Multisignature schemes on the other hand, enhance this security even further since the signatures are done by more than one party that need to authorize the transactions.Finally, the engagement of decentralization, immutability, cryptographic primitives, consensus mechanisms, and advanced cryptographic techniques as an installment of bodyguards make blockchain's safety more secure. Through dignifying and deepening these built-in and appendages security measures, the blockchain technology still stands out as a better platform which is trustworthy and robust for various applications like financial transactions which span to supply chains and also beyond. (Leng & Zhou & Zhao & Huang, 2020).

Smart contracts(Zheng a & Xie & Dai & Chen & Chen & Weng & Imran, 2019), (Parasu & Reddy , 2019) ,( Kõlvart & Poola & Rull , 2016) testing tools became popular after Ethereum come into existence in 2015. The development of smart contract in blockchain technology witnessed significant progress. The smart contracts had some issues – reliability, scalability, and security. Smart contracts development is going not in default way for developing the software products. This is a reason why smart contracts can’t perform exhaustive testing as well as it is expensive. Industry fields like smart contracts testing and analysis of the code for vulnerability attract much research (Luu & Chu & Olickel & Saxena & Hobor , 2016).Cryptocurrencies record transactions in a data format called blocks. The two most popular cryptocurrencies, Bitcoin and Ethereum, allow a set of written rules or files to process transactions. This has proven to be very useful in the concept of smart contracts to complete management projects on the blockchain. Recently, Ethereum’s smart contract system has become increasingly popular, supporting thousands of contracts and earning millions of dollars in revenue(Metcalfe , 2020).

In this thesis, the author study the security of managing smart contracts on Ethereum in an open decentralized network similar to cryptocurrencies. In Ethereum number of new security challenges that enable adversaries to manipulate smart contracts to their advantage. These errors indicate hidden gaps in the understanding of the semantics distributed on the website. As an update, the author provide additional ways to work with Ethereum so contracts are less complex. For those writing contracts for Ethereum, the author created a diagnostic tool called Oyente to look for possible errors. Of Ethereum’s 19,336 contracts, Oyente examined 8,833 of them for risks, including the TheDAO bug that caused $60 million in losses in June 2016. The author will also discuss the possibility of other attacks from some research that qualified and confirmed the attack in the Ethereum mainnet (Atzei & Bartoletti & Cimoli, 2017) , (Parizi, & Dehghantanha & Choo & Singh , 2018).

Was selected tools based on which blockchain automation tools list was created(Gattermayer J., Pˇrevr´atil M., Kubˇena J,2024), (Parasu & Reddy , 2019).Shown in (Fig.2.2.1).

A screenshot of a computer

Description automatically generated

Fig.2.2.1. List of blockchain automation testing tools

For testing bitcoin author desire to show BitcoinJ Testing (bitcoinj) (Parasu & Reddy , 2019) BitcoinJ is a java library for working with Bitcoin protocol. It provides tools for working with Bitcoin transactions, wallets managing and interacting with bitcoin networks. Automation testing for BitcoinJ. Involves creating scripts and test cases to automatically verify functionality, security and performance of applications using BitcoinJ library. Function Testing: Transaction Handing - Automated tests can be designed to create sign and broadcast transactions to ensure that the library correctly handles the transaction lifecycle(Parasu & Reddy , 2019). Wallet Functionality: Test the functionality related to creating wallets, generating addresses, and managing private keys. Network interaction: The library can connect to Bitcoin network, send and receive transactions and synchronize with blockchain(Bitcoinj , 2009).

Ethereum Tester is using for testing Ethereum (Leid & Merwe & Visser , 2020).Ethereum, one of the most used platforms for building blockchains applications and includes tools that make application development easier and test. Ethereum tester for Web3 integration, APIs, smart contracts, Backend and several other blockchain tests. Testnet simulate production like a blockchain where the real Ethereum and tokens can be tested without incurring actual costs or risks associated with using the main Ethereum network. This helps developers and testers perform simulations.( Barboni & Morichetta & Polini , 2021) Since its inception, the adoption of blockchain technology has continued to increase and new applications are being explored. This goes hand-in-hand with the natural trust that can be brought about by adopting these technologies in certain situations. Additionally, over the past few years, many blockchain technologies have been enhanced with the support of smart contracts. These can be thought of as software programs that can be deployed and run on a blockchain, and in the process produce data (operations) that are stored in the blockchain itself. These smart contracts allow stakeholders to participate in the execution of the smart contract with sufficient assurance that their work and that of other parties involved in the contract are being performed in accordance with the terms of the contract itself. These types of methods have been successfully implemented in computer engineering (Porru & Pinna & Marchesi & Tonelli, 2017), especially in the integration of control and collaboration systems (Zheng & Xie & Dai & Chen & Wang & Imran , 2019). Compared with traditional software applications, the creation of smart contracts presents unique characteristics and challenges due to the environment in which they are delivered and executed (Zheng & Xie & Dai & Chen and Wang , 2018). New tools and appropriate testing are needed to enable developers to write and release secure code. In particular, several reasons can be identified why smart contractors often require more reliable guarantees and comprehensive testing(Barboni & Morichetta & Polini ,2021).List of relevant characteristics (Barboni & Morichetta & Polini ,2021):

- Smart Contracts manage valuable assets,

- Transactions are irreversible,

- Smart Contracts are immutable,

- Blockchain environment,

- New software stack,

- Lack of best practices,

-Lack of mature testing tools.

Exonum Testkit (Parasu & Reddy , 2019) Testing the full service of the blockchain application has been actively carried out with the help of the Exonum Test Kit features. This instrument enables us to conduct API testing without the need for any network operation during the execution of transactions using the Consensus algorithm. (Yanovich & Ivashchenko & Ostrovsky & Shevchenko & Sidorov , 2018).

Hyperledger Composer - (Parasu & Reddy , 2019) Hyperledger Composer is an open source framework designed to simplify building and testing blockchain networks the schema shown in ( Suliyanti & Sari , 2021). Developed as part of the Hyperledger project, author provides developers with an easy to-use environment to showcase , distribute and test blockchain applications. One of the most important features of Hyperledger Composer is its low hardware requirements, which makes it accessible to many developers. Composer uses technologies like Docker for deployment and networking, eliminating the need for installation and configuration, allowing developers to focus on building and deploying their solutions. Inventor provides a modeling language to help developers describe how their blockchain network is constructed using logical concepts such as assets, participants, and events. These modeling languages eliminate the complexity of block development and allow rapid iteration of programming. In addition to modeling capabilities, Creator also provides a user interface (UI) that facilitates interaction with the blockchain network. These user interfaces allow users to view and interact with blockchain data and transactions, making it easier to understand and manage the network. Composer also features a command line interface (CLI) that allows developers to create tasks such as deploying smart contracts, testing network configurations, and systematically handling block assets and events. One of the main advantages of Hyperledger Composer is its support for self-testing. Developers can write independent tests to verify the correctness and reliability of their blockchain applications and ensure that they meet stated requirements and function as expected under different circumstances. Inventor's testing capabilities include machine testing, interactive testing, and unit testing, providing developers with the tools to build strong and reliable blockchain networks. Overall, Hyperledger Composer is a powerful tool for configuring, deploying, and testing blockchain networks. Composer has low hardware requirements, a friendly interface, and supports machine testing, helping developers speed up the development process and easily create high-quality blockchain applications.( Suliyanti & Sari , 2021).

Populous (Parasu & Reddy, 2019) - the native cryptocurrency of Populous weigns in the coin consisting with PPT (Populous Platform Token), which represents a breakthrough in blockchain technology not occurring only in invoice finances, but becoming a general realization. Through blockchain technology, Populous improves operational efficiency of invoice financing by giving a straight way to business to get instant liquidity, so helping the investors get triangular income. Our research undertakes Populous to its ultimate features and functionalities exploring the ability of Populous in remodeling the financial landscape and possibly to a complete alteration of traditional financing systems. By performing an analysis of the blockchain architecture feature that has been created by Populus, the smart contract opportunities and the application in the real world, the research will explain the role that has been played by the platform in providing solutions to invoice financing implementations which are efficient, transparent and secure. Additionally, by comparating and contrasting Pospos problems and its outcomes for the DeFi sector to other instruments, this paper is also providing a comprehensive view of it influence on the future financial market.

Manticore (Mossberg & Manzano & Hennenfent & Groce & Grieco & Feist & Brunson & Dinaburg, 2019) is a strong security analysis tool for testing and analyzing blockchain applications, specifically smart contracts. Developed by safety research firm Trail of Bits, Manticore provides a number of tools and features to identify vulnerabilities and assess the security of blockchain systems(Mossberg & Manzano & Hennenfent & Groce & Grieco & Feist & Brunson & Dinaburg, 2019). One of the most important features of Manticore is its support for encoding, a method of analyzing the behavior of a program by looking for all possible encodings rather than encoding them. This enables Manticore to automatically generate test cases that examine different operating systems, allowing developers to identify security vulnerabilities such as logic errors, buffer overflows, and general vulnerabilities. In addition to benchmark testing, Manticore also supports virtualization, allowing developers to execute smart contracts in a virtual environment and evaluate their performance under different conditions. This helps developers identify issues that may not be obvious during simulation, such as uncertainties related to gas usage, external dependencies, and interactions with other contracts(Mossberg & Manzano & Hennenfent & Groce & Grieco & Feist & Brunson & Dinaburg, 2019). Manticore also provides many monitoring and management tools to help developers identify and fix security vulnerabilities in smart contracts. These tools include government research, which helps developers explore the nature of smart contracts and identify potential security risks, and audit research, which helps developers investigate the implementation of smart contracts and identify security issues. In addition, Manticore also supports asset testing. Developers can specify the assets that their smart contracts need to meet and automatically create test cases to verify the assets(Mossberg & Manzano & Hennenfent & Groce & Grieco & Feist & Brunson & Dinaburg, 2019). This enables developers to arrange that their smart contracts work as expected and follow correct security practices. Overall, Manticore is a powerful tool for testing the security of blockchain applications, providing a lot of features for identifying vulnerabilities and assessing the security of smart contracts. Manticore uses coding and integration techniques as well as analysis and debugging tools to help developers build secure, robust blockchain systems.

Corda Testing Tool (Parasu & Reddy , 2019) is an open source decentralized blockchain platform. It has a testing module designed to support:

- Contract test writing,

- Integration testing,

- Process test writing,

- Product testing.

Corda is a platform for recording and processing financial transactions that is distributed across multiple ledgers. agreements, created to execute the plan outlined in this document. The Corda platform backs smart contracts, aligning with the definition of. Clack, Bakshi, Braine remains unchanged. 3 Our intelligent contract is a binding agreement that is enforced autonomously. capable of being automated through computer programming with human input and oversight, as well as controlled by humans. whose rights and duties, as stated in legal language, can be enforced by law. The intelligent contract connects the logic of a business with its data to the corresponding legal framework. Writing in a way to make sure that the financial contracts on the platform are firmly established. firmly established in legal grounds and can be upheld, ensuring a clear course of action to adhere to. occurrence of unclearness, doubt or disagreement. (Brown & Carlyle & Grigg & Hearn , 2016).

Another automated tester is Embark (Jayachandra Parasu & Vishwanatha Reddy , 2019)is a framework for building, testing, and deploying blockchain software. It can create and send distribution requests. Decentralized applications use one or more decentralized technologies. Embark currently includes decentralized storage (IPFS), EVM blockchain (Ethereum), and communication platforms (Whisper and Orbit). Swarm is supported for deployment.

Truffle (Trufle,2020) - is a commonly used tool among Ethereum developers with various testing capabilities, including consensus testing. This framework is more than just testing and is a great addition to your testing toolbox (Lan & Sherry & Popa & and Ratnasamy, 2016)(Trufle,2020).

HardHat (Hardhat) - refers to an open source environment intended for development for Ethereum smart contract. It ensures a dynamic environment to develop, test, and implement smart contracts and distributed applications in the form of dApps on this Ethereum blockchain platform(Hardhat, 2023). At first, Hardhat was created by Nomic Lab, which has seen an impressive uptake within the Ethereum community because of the feature’s flexibility, expansibility, as well as the excellent plugins' set of tools it has. Testing Framework: Testing is an important part of the project(Hardhat, 2023). The hardhat plug-in automatically run tests and debugs smart contracts through the use of test framework. It gives them an opportunity to check their program for both correctness and reliability, so the final code, ready to be deployed on the Ethereum main platform, would not cause any bugs(Hardhat, 2023).

The Anvil development chain is the part of Foundry project that will contribute to the development of blockchain systems(Gattermayer J., Pˇrevr´atil M., Kubˇena J,2024). Developed with Rust as the programming language of choice Anvil tries to take advantage of the language’s speed and execute it(Foundry,2023). It provides required operations; for instance, functions for debugging a transaction, modification of-parity, and control of several proceedings within a chain(Foundry,2023). This makes Anvil a very useful tool for developers who are working on applications using block chain technology (Foundry,2023).

Each testing tool was tested by author and after long research and many experimental tests was detected that 1 of the tools are not working at this moment correctly - Hyperledger Composer, but the other tools are properly working till now. The final list with criteria’s is shown in (Fig 2.2.2).

A screenshot of a computer

Description automatically generated

Fig. 2.2.2. The final list with criteria’s to each testing tool

Testing Features:

* 5: The tool supports a wide range of testing methods (unit, integration, gas, security, etc.) and provides advanced features such as performance analysis.
* 4: Supports most testing methods but lacks some advanced features.
* 3: Provides basic testing features like unit and integration testing, but lacks advanced tools like performance or security testing.
* 2: Limited testing functionality, only supports basic unit tests.
* 1-0: Minimal or no built-in testing functionality.

Developer Support and Documentation

* 5: Excellent support with extensive documentation, large active community, and regular updates.
* 4: Good support with a decent amount of tutorials, but less active community or slower updates.
* 3: Sufficient documentation and community support, but with limited resources.
* 2: Limited documentation and minimal community activity.
* 1-0: Poor or outdated documentation, inactive or nonexistent community.

Performance:

* **5**: Excellent performance, handles large-scale testing without issues, provides tools for stress testing.
* 4: Good performance, capable of handling moderate to large-scale testing.
* 3: Sufficient for average use cases, but struggles with large-scale tests.
* 2: Limited performance for large-scale or complex testing environments.
* 1-0: Poor performance or significant issues with scalability.

Flexibility and Extensibility:

* 5: Highly flexible and easily extensible, supports custom plugins and configurations.
* 4: Good flexibility, but less customizable than the top tools.
* 3: Sufficient flexibility for most use cases, but limited customizability.
* 2: Limited flexibility and hard to extend.
* 1-0: Rigid, little to no flexibility.

Tool Ecosystem and Plugins :

* 5: Very rich ecosystem with multiple plugins and integrations for deployment, debugging, and testing.
* 4: Good ecosystem with some useful plugins, but less variety.
* 3: Adequate ecosystem with limited plugins and integrations.
* 2: Few plugins or extensions available.
* 1-0: No plugin or ecosystem support.

Security Features:

* 5: Excellent security testing features, including symbolic execution, fuzzing, and vulnerability scanning.
* 4: Good security testing, but lacks advanced features like symbolic execution.
* 3: Basic security testing features are available.
* 2: Limited security testing.
* 1-0: No security testing capabilities.

Ease of Use:

* 5: Very easy to use with a user-friendly interface, clear documentation, and quick installation.
* 4: Easy to use, but requires some prior knowledge of blockchain technology.
* 3: Moderate learning curve, good documentation but requires more setup or technical understanding.
* 2: Difficult to set up or requires deep technical knowledge to use.
* 1-0: Very hard to use with poor documentation or complex setup.

Author would like to highlight 3 tools with the author use in experiments:

* Truffle
* Hardhat
* Anvil/Foundry

The author aims to highlight three tools used in experiments: Truffle, Hardhat, and Anvil/Foundry.

The Ethereum smart contract development and testing sector is dominated by two major tools: Truffle and Hardhat. Both tools offer very good documentation, have rich functionalities, and an interactive shell which can be attractive for smart contract developers. This research aims to find out which one of these tools – Truffle, Hardhat, or Foundry – is more effective, easy to use, and performs well than the others under different situations (Truffle, 2020),( Hardhat, 2023),( Foundry, 2023).

The reasons for choosing Truffle and Hardhat are quite simple: they are both popular, well-documented, and are used by numerous developers in the blockchain sphere, as is Foundry. These tools have a number of uses and offer explicit guidelines, so it is logical to use them as the basis for comparison (Truffle, 2020),( Hardhat, 2023),( Foundry, 2023).

The author’s basic assumption is thus based on using and creating smart contracts through Truffle, Hardhat, and Foundry. Several experiments will be performed to examine the efficiency and utility of each tool in diverse use cases, including contract deployment, testing, debugging, and integration with other services (Truffle, 2020),( Hardhat, 2023),( Foundry, 2023).

The main motivation for pitting these frameworks against each other is the possibility to improve the SDLC processes of blockchains. The author will analyze the relative merits and demerits of the various options to establish their applicability in meeting certain project needs and challenges (Truffle, 2020),( Hardhat, 2023),( Foundry, 2023).

One of the main goals is to focus on the need to audit smart contracts for Ethereum in order to prevent threats and guarantee the successful implementation of a transaction on the blockchain. With the help of proper testing frameworks added to Truffle, Hardhat, and Foundry, the author will show how these tools minimize risk and enhance the security of identity and data interchange (Truffle, 2020),( Hardhat, 2023),( Foundry, 2023).

The results generated by this study will provide for a very important base for participants in the industry and for developers of the blockchain. Through the comparison of different smart contract development tools and their relative strengths they offer, companies can preemptively make rational choices on which tool to use to satisfy their smart contract need. These insights will help organizations to eliminate the waste that is prevalent in the development process of such solutions and also optimize code quality and reliability of the blockchain applications.

As a final summary, the Truffle and Hardhat tools for the development and testing of the Ethereum's smart contract constitute a critical element of the process toward blockchain innovation. By use of these examples in real-world scenario, author can get invaluable feedback on the strengths and limitations of these tools, and that feedback can help move the world towards more radical and more secure innovations in the very system of Blockchain production.

2.3 Test Blockchain Applications principles

Blockchain-powered decentralized apps popularity is supported by two fundamental characteristics, which makes them highly trustable: decentralization and immutability. Nevertheless, among a variety of such systems the lack of detailed enough testing methods is an apparent issue that still exists and is exploited by security breaches.Testing techniques commonly used may not correspond to decentralized applications that run on blockchain exactly. Often these methods are framework-agnostic and take care of testing the frontend user interface or the backend code, but disregard the granular interconnections among these aspects.The way to respond to that obstacle is the development of creative means of testing that are tailor-made for the specials features of the blockchain operatings on the basis of decentralization. The learning strategy that has inspired me in this school year is the two-step process as presented in this program.In the beginning different events are applied to show the non-obvious relations between lightweight client-side events and the more heavyweight blockchain-side contracts. This is the stage when how user actions on the frontend are directly reflected on the back of blockchain is explained(Staroletov & Galkin , 2022a). The purpose of the step is to show what kind of application and its behavior is designed, as well as to provide suitable responses to user actions.Statewide after this, the complex strategies are implemented to verify the integrity and applicability of the entire distributed application in detail. The methods involve a variety of techniques like unit testing, integration testing, and end-to-end testing, of which third condition is more suited to the structure of blockchain technologies.Testing of the unit settings means testing corresponding individual components or smart contracts of the model to check their proper behaviour and ability to handle errors effectively. Integrative testing’s goal is to make sure that the interconnection and communication between the different application modules or layers is properly working like frontend interface and the backend blockchain logic.User-centric testing means all functions are verified from the user's point of view, so it imitates full-scale implementation that is able to locate any existing problems or risks. On the contrary, a complete solution providing a flawless work of all modules on the client to the underlying blockchain infrastructure improves the general reliability and security of decentralized applications. (Gao & Liu & li & Liu & Yaung ,2019). A blockchain application is a type of contemporary software that operates within its own environment and communicates with other instances of applications. These apps operate in a decentralized manner across numerous nodes and handle requests from many users. Therefore, these are advanced applications that are at risk for particular errors because of unpredictable network behavior. Furthermore, they are prone to mistakes that are present in every software system. Learning how to test cryptocurrency applications is crucial due to the potential financial losses that bugs can cause. This paper investigates the internal quality assurance processes of Bitcoin and Ethereum platforms across different levels of logical structure. Following that, author discuss about test setup made for conducting functional tests on cryptocurrency payment gateways(Staroletov & Galkin , 2022a). The solution abstracts software to enable API calls to virtualized nodes on different platforms, mimicking real blockchain networks through emulators. Challenges in testing blockchain applications using real networks it is Contemporary managers must realize that incorporating a blockchain element into a business information system is impossible without adequate testing. Testing should be the primary concern during the development of systems that rely on distributed consensus-based algorithms within a blockchain-enabled platform, as the fate of digital funds is directly tied to these operations. Nevertheless, the complexity of guaranteeing the quality of these software classes is objectively due to various blockchain-related problems(Staroletov & Galkin , 2022a) (Gao & Liu & li & Liu & Yaung ,2019).

Some major issues in blockchain applications testing:

* Average rate of new blocks generation in the blockchain,
* Cost,
* Size of blockchain,
* Security.

Average rate of new blocks generation in the blockchain - The typical speed at which new blocks are produced in the blockchain. Velocity is a key element in automated testing across various blockchain ecosystems. Certainly, besides waiting for the next transaction block, it is also important to wait for confirmation blocks to confidently utilize necessary Proof-of-Stake algorithms. However, numerous widely-used blockchains continue to experience delays of 20-30 minutes (or longer during peak network activity). Additionally, the speed of processing transactions within the block is also impacted by a mining fee set by the sender. Automated tests, naturally, are unable to dedicate an hour or two to execute a single test script(Staroletov & Galkin , 2022b).

Cost - in addition to paying for transactions, people must have the ability to validate the transfers. Given all of this, testing on a live blockchain can be costly and inconvenient. Of course, there is a danger of losing funds throughout the testing procedure if errors occur. Of course, there are so-called test networks, but their conditions are frequently poor (peaple mention a limited number of miners, regular efforts to "51% attack" by some enthusiasts, and not every blockchain has a working test network) (Staroletov & Galkin , 2022b).

Size of blockchain – To make complete transaction different block-chains utilized in project should be synchronized , for a considerable amount of time , use less as possible physical memory and they must run on powerful servers(Staroletov & Galkin , 2022b).

It is necessary to avail the wallet of users with secure access because wallets have got expensive digital currency assets and remain confidential. In order to avoid having access forbidden, security measures must be used regularly and across the software development life cycle.In order to ensure that the website is safe undetected access for testers, customers, and other participants to the website is not tolerated. Permission to access the resource should be granted based on the role and responsibilities with authentication and authorization mechanism in place to verify the Identity /permission.Among multifactor authentication added to encryption techniques which are just extra layers of security there regularly audits and penetration test made for identifying and resolving the vulnerabilities.Secure channels, such as encryption protocols, contribute to the trust and safety of the blockchain network by preventing unauthorized access to data shared between users and the network.Blockchain developers can gain a competitive advantage and build confidence by putting security first and employing rigorous access controls when they create their applications. These controls may include several measures such as authentication and encryption which, in turn, defy cyber threats and protect users' data and assets. (Staroletov & Galkin , 2022b).

3 Testing Blockchain automation testing tools

Comparison of Truffle and Hardhat for Automated Functional Testing in Blockchain Development - Blockchain implementation involves a number of robust testing tools to validate the integrity, safety and operability of smart contracts and decentralized application services (Dapp). Currently, the truffle framework and Hard hat are top pre-eminent test automation frameworks used by majority of application developers. This will include an appraisal of both tools' positives, negatives, and their applicability to blockchain applications(Truffle,2020) (Hardhat, 2023).

Truffle is a well-known development framework for Ethereum-based decentralized applications, coming with all the tools one needs to work on their projects easily and quickly — contract compiling, deployment, testing, etc(Truffle,2020) (Hardhat, 2023).

Testing Suite - smart contracts writing with Truffle provides a built-in testing engine featuring Mocha and Chai for writing and running test cases with JavaScript or Solidity.Fixture Support: Truffle’s adoption for fixtures always settles up predefined contract states for testing and, therefore, the test-time gets more efficient and repeatable.Contract Interaction: The use of truffle to link the web3.js and of course, the comprehensive testing of contracts through the variable flows, facilitates the interaction of the contracts and their transaction management.Integrated Development Environment (IDE) - Riley introduces the concept of Truffle, a package that not only provides an integrated development environment but also allows developers to execute tests easily in the same environment.Limitations Truffle testing system maybe lacks the level of flexibility and customization that other testing frameworks have, hence, a limitation of the applications of varied scenarios.Performance: To address this limitation, Truffle would improve its scalability and depend less on JS-based testing frameworks for better testing performance within large projects.Hardhat is a strong tool on the Ethereum development environment which comes with a variety of options that make it possible for smart contract development, testing, and deployment.Rich Testing Capabilities - tenvironment of hardhat is made robust enough for the writing execution and debugging the tests by using mocha, chai, and Ether.js(Truffle,2020) (Hardhat, 2023).

Fixture and Snapshot Support: The Opportunities presented by Hardhat can thus be employed by creating situations that allow for fitting endpoints of contract and their snapshots for restoration of the contract state, thus improving test isolation and repeatability respectively.Scriptable Tasks - the scriptable tasks function provides developers with an opportunity to automate the whole workflow from testing and enabling the integration of external modules, which means that the testing process will be greatly simplified.TypeScript Support - With Hardhat, the developers to can create test and smart contract in TypeScript and when used, provides them benefits like good type safety and code readability.Learning Curve - given the powerful functionalities and the bespoke nature of Ethereum's development tools, it's quite possible for beginners to get intimidated by the fact that they are not familiar with Ethereum's workflows and tools.Community and Ecosystem: Currently, it defacto became a go-to tool in the Ethereum developer community, though, it is likely that for newcomers it will be more challenging to find information or required support due to the smaller ecosystem, completely opposite to already established Truffle(Truffle,2020) (Hardhat, 2023).  
  
 For this thesis author use one smart contract for all 3 testing tools

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Pavlo\_Nikolaiev\_test {

    string public name = "Test Token";

    string public symbol = "TTK";

    uint8 public decimals = 18;

    uint256 public totalSupply;

    mapping(address => uint256) public balanceOf;

    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(uint256 \_initialSupply) {

        totalSupply = \_initialSupply;

        balanceOf[msg.sender] = totalSupply;

    }

    function transfer(address \_to, uint256 \_value) public returns (bool success) {

        require(\_to != address(0), "Invalid address");

        require(balanceOf[msg.sender] >= \_value, "Insufficient balance");

        balanceOf[msg.sender] -= \_value;

        balanceOf[\_to] += \_value;

        emit Transfer(msg.sender, \_to, \_value);

        return true;

    }

    function approve(address \_spender, uint256 \_value) public returns (bool success) {

        allowance[msg.sender][\_spender] = \_value;

        emit Approval(msg.sender, \_spender, \_value);

        return true;

    }

    function transferFrom(address \_from, address \_to, uint256 \_value) public returns (bool success) {

        require(\_to != address(0), "Invalid address");

        require(balanceOf[\_from] >= \_value, "Insufficient balance");

        require(allowance[\_from][msg.sender] >= \_value, "Allowance exceeded");

        balanceOf[\_from] -= \_value;

        balanceOf[\_to] += \_value;

        allowance[\_from][msg.sender] -= \_value;

        emit Transfer(\_from, \_to, \_value);

        return true;

    }

}

The methodology of automatic testing tools - in the context of developing blockchain-based applications the choice of this automated functional testing tool is one of the key factors within the dynamic and competitive context of developing and deploying blockchain solutions. This research aims to define a comprehensive methodology for comparing the leading testing frameworks: Such environments include Truffle, Hardhat, and Anvil/Foundry. The approach will be based on a set of experiments aimed at comparing and assessing strengths and limitations of the given tools, including their usability, efficiency, and functionality when it comes to Ethereum smart contract development.

Test Environment Setup:

We will set up the local Ethereum the same way every tool is used (Ganache for Truffle, Hardhat Network for Hardhat, Anvil for Foundry).

Test Cases Development:

* A set of standardized test cases will be created for the ERC20 token contract, focusing on:
* Deployment Tests: Confirming that the targeted deployment is successful.
* Functionality Tests: To guarantee correct functioning of Transfer, Approve and -transfer From.
* Event Emission Tests: Confirming that events are fired correctly.

Performance Metrics:

The following metrics will be analyzed:

* Execution Time: The time taken by tests to run.
* Success Rate: Passed tests against failed tests.
* Error Handling: Accuracy of communication of failed tests.
* User Experience: A survey of developers regarding setup and ease of use.

Testing Procedures - each tool will be evaluated by starting up a project, compiling the smart contract and deploying it to a local network and running unit test cases.

Data Analysis - using the results of the tools, potential differences between their success rates and times in executing specific tasks will be investigated, while qualitative judgement of their usability by the participants of the study will be evaluated.

3.1 Truffle

Before author can start testing a new project on the Ethereum blockchain via Truffle, he has to set up a new project. This can be done by making new directory for the project and cd into it. The Truffle framework offers basic elements when it comes to writing and testing the smart contracts.

After moving to the correct directory a person can start the project through the use of the command truffle init. This command blessing creates proper project build structure of Truffle which includes different folders such as contracts, migrations and tests. Once the structure of the project is set, the project can be opened in any text editor of the user. For this project, Visual Studio Code is used; it can be opened from the command line with the command code . in the correct directory.

Having established the Truffle project, the next step is to develop a smart contract which will be necessary in the testing. New to the contracts folder must be the file called Pavlo\_Troffie\_test.sol. Solidity will be used for the programing of the smart contract. Below is an example of how the smart contract can be structured.

Now the smart contract (Trufle,2020) is ready and should be tested , this operation have to be done with truffle compile command which have to build the smart contract. At this moment project contain smart contract and after build this smart contract project contain new.json file which has name of the project and function needed to test smart contract. For simplicity, the author will deploy this smart contract on the local blockchain. The process for launching the smart contract to the Ethereum main network, where real Ethereum dwells, is similar. But first, author launch the local blockchain using (Trufle,2020) Truffle it can be done with truffle develop after this operation will get the result which contain host ,port ,accounts and private keys for this accounts, but this value always different in different projects but the technology is the same it is easy to prove just create a new project and values will be different, all information about mine shown in Fig 3.1.1.

A screenshot of a computer

Description automatically generated

Fig. 3.1.1 Local blockchain launch

The most important information (Trufle,2020) at this moment it is host and port which author should use in project settings , because file truffle-config.js is empty at this moment so this information should be pasted in file for successful project build :

module.exports = {

networks: {

networks: {

development: {

host: "127.0.0.1",

port: 9545,

network\_id: "\*",

accounts: {

default\_balance\_ether: 1000

}

}

}

},

mocha: {

},

compilers: {

solc: {

version: "0.8.21",

}

}

};

For deploy smart contracts to migration folder some instruction should be added, but first should be created a file in folder – filename.js and this file should contain the following instructions:

const Pavlo\_Troffie\_test = artifacts.require("Pavlo\_Troffie\_test");

module.exports = function(deployer) {

deployer.deploy(Pavlo\_Troffie\_test);

};

A screenshot of a computer

Description automatically generated

Fig. 3.1.2a Blockchain test transaction a

A screenshot of a computer

Description automatically generated

Fig. 3.1.2 Blockchain test transaction b

As shown in Fig. 3.1.2a and 3.1.2b smart contract (Trufle,2020) can be deployed using truffle migrate. The result of this command will be test transaction which similar to real one and contain same information as normal .On result will shown Network name in which was tested transaction, network id, block gas limit , transaction hash it is information about transaction which will shown to other people , contract address , gas price, and total cost.

As shown on the 2 results the values are different and if use real blockchain and send real blockchain coins the principles will not change.

Now project testing can be develop (Trufle,2020) for do this ,in folder test should be created file filename.js in which should write tests. In test which will shown letter was checked the problem which can be happened with smart contract and it protect people who created smart contracts and people who are use it. The tests which have bean used in thesis showed below:

const Pavlo\_Troffie\_test = artifacts.require("Pavlo\_Troffie\_test");

contract("Pavlo\_Troffie\_test", (accounts) => {

let auctionInstance;

beforeEach(async () => {

auctionInstance = await Pavlo\_Troffie\_test.new({ from: accounts[0] });

});

it("should return the correct worker address", async () => {

const workerAddress = await auctionInstance.worker();

assert.equal(workerAddress, accounts[0], "worker address is incorrect");

});

it("should allow the worker to start the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const isAuctionStarted = await auctionInstance.auctionStarted();

assert.isTrue(isAuctionStarted, "Auction should be started");

});

it("should revert if a non-worker tries to start the auction", async () => {

try {

await auctionInstance.startAuction({ from: accounts[1] });

assert.fail("Are not worker");

} catch (error) {

assert(error.message.includes("revert"), "Are not worker");

}

});

it("should allow users to place bids", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const initialBalance = await web3.eth.getBalance(accounts[1]);

const bidAmount = web3.utils.toWei("1", "ether");

await auctionInstance.bid({ from: accounts[1], value: bidAmount });

const latestBid = await auctionInstance.latestBid();

const latestBidder = await auctionInstance.latestBidder();

const finalBalance = await web3.eth.getBalance(accounts[1]);

assert.equal(latestBid.toString(), bidAmount, "Latest bid is incorrect");

assert.equal(latestBidder, accounts[1], "Latest bidder is incorrect");

assert(finalBalance < initialBalance, "Bid amount should be deducted from balance");

});

it("should Fail if bid amount is lower than the current highest bid", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const initialBid = web3.utils.toWei("2", "ether");

await auctionInstance.bid({ from: accounts[1], value: initialBid });

try {

const lowerBid = web3.utils.toWei("1", "ether");

await auctionInstance.bid({ from: accounts[2], value: lowerBid });

assert.fail("Test should be Failed if bid amount is lower than the current highest bid");

} catch (error) {

assert(error.message.includes("revert"), "Test should be Failed if bid amount is lower than the current highest bid");

}

});

it("should allow the worker to finish the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

await auctionInstance.bid({ from: accounts[1], value: web3.utils.toWei("1", "ether") });

const initialBalance = await web3.eth.getBalance(accounts[0]);

await auctionInstance.finishAuction({ from: accounts[0] });

const finalBalance = await web3.eth.getBalance(accounts[0]);

assert(finalBalance > initialBalance, "Funds were not transferred to seller");

});

it("should Fail if a non-worker tries to finish the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

await auctionInstance.bid({ from: accounts[1], value: web3.utils.toWei("1", "ether") });

try {

await auctionInstance.finishAuction({ from: accounts[1] });

assert.fail("Fail test if caller is not the worker");

} catch (error) {

assert(error.message.includes("revert"), "Fail test if caller is not the worker");

}

});

});

To check tests in consol should be written truffle test which will start testing smart contract. If (Trufle,2020) some tests are fail that mean some smart contract functions are incorrect and can effect to operation on this blockchain. But if all have passed it shown that all function was written well , the result shown in Fig. 3.1.2.There are result of 7 tests which have passed.

A screenshot of a computer program

Description automatically generated

Fig. 3.1.2 Blockchain test results

After getting this result smart contract have written correct and tests are prove it. The time of some test are shown for some are not shown it is because when it is not shown the time of work less then 50ms and the result shown that 4 tests are executed long time , 1 executed little bit longer then needed and 2 the simplest ones are working fast. In that regard, Truffle has become the first choice test that displays a smooth platform enabling users to go through the test with the greatest ease. Its docs, uploaded to official portal, helps people who are clueless especially in the areas it is lacking by providing them additional knowledge. The detailed constant is not only the step-by-step documentation of setup and installation but also provides the best onboarding experience for Truffle first-time users. Although a basic knowledge of node.js is fundamental for effective Solidity developing with Truffle framework, it is not the only prerequisite. The tool's functionality and complexity need users to be hit right in the middle to have a wide understanding of the software programming foundations which, most of the time, will lead them going deep into its features. Meanwhile, however, despite of a certain deficiency in Truffle's reputation, which was caused by its straightforward interface, Truffle will remain one of the first choices for the beginners as the developer team will still consistently solve the problems faced by the network. One of the fundamental activities of Truffle’s team is the creation and implementation of novel approaches and designs which benefit users with end-to-end programmable corpus.Additionally Truffle's dynamic system of testing and its exhaustive documentation, this, in fact, is the most significant reason responsible for the maintenance of the integrity within the blockchain towers. Truffle solves this problem by offering developers user-friendly tools and resources to overcome the intricacies of blockchain technology. That way developers can deploy robust and reliable solutions more quickly.

Generally speaking, the ease of use, respectable documentation, and innovative nature of Truffle are the major factors that put it in the spotlight as the most useful blockchain automation testing option. With the decentralized nature of blockchain technology continually changing, Truffle is still an indispensable partner for developers that are trying to eliminate investment costs of testing and secure the functionality of their creations.

3.2 Hardhat

Overview of Testing Process - constructing integrated testing the process implied the application of Hardhat testing tools which were used to examine the functioning and behavior of the "Lock" smart contract(Truffle,2020) (Hardhat, 2023):

The tests were made to ensure the validity of the accurateness of contract deployment, which were used to validate the contract state modifications, and lastly to verify the execution of contract functions(Truffle,2020) (Hardhat, 2023):

const Pavlo\_Troffie\_test = artifacts.require("Pavlo\_Troffie\_test");

contract("Pavlo\_Troffie\_test", (accounts) => {

let auctionInstance;

beforeEach(async () => {

auctionInstance = await Pavlo\_Troffie\_test.new({ from: accounts[0] });

});

it("should return the correct worker address", async () => {

const workerAddress = await auctionInstance.worker();

assert.equal(workerAddress, accounts[0], "worker address is incorrect");

});

it("should allow the worker to start the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const isAuctionStarted = await auctionInstance.auctionStarted();

assert.isTrue(isAuctionStarted, "Auction should be started");

});

it("should revert if a non-worker tries to start the auction", async () => {

try {

await auctionInstance.startAuction({ from: accounts[1] });

assert.fail("Are not worker");

} catch (error) {

assert(error.message.includes("revert"), "Are not worker");

}

});

it("should allow users to place bids", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const initialBalance = await web3.eth.getBalance(accounts[1]);

const bidAmount = web3.utils.toWei("1", "ether");

await auctionInstance.bid({ from: accounts[1], value: bidAmount });

const latestBid = await auctionInstance.latestBid();

const latestBidder = await auctionInstance.latestBidder();

const finalBalance = await web3.eth.getBalance(accounts[1]);

assert.equal(latestBid.toString(), bidAmount, "Latest bid is incorrect");

assert.equal(latestBidder, accounts[1], "Latest bidder is incorrect");

assert(finalBalance < initialBalance, "Bid amount should be deducted from balance");

});

it("should Fail if bid amount is lower than the current highest bid", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

const initialBid = web3.utils.toWei("2", "ether");

await auctionInstance.bid({ from: accounts[1], value: initialBid });

try {

const lowerBid = web3.utils.toWei("1", "ether");

await auctionInstance.bid({ from: accounts[2], value: lowerBid });

assert.fail("Test should be Failed if bid amount is lower than the current highest bid");

} catch (error) {

assert(error.message.includes("revert"), "Test should be Failed if bid amount is lower than the current highest bid");

}

});

it("should allow the worker to finish the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

await auctionInstance.bid({ from: accounts[1], value: web3.utils.toWei("1", "ether") });

const initialBalance = await web3.eth.getBalance(accounts[0]);

await auctionInstance.finishAuction({ from: accounts[0] });

const finalBalance = await web3.eth.getBalance(accounts[0]);

assert(finalBalance > initialBalance, "Funds were not transferred to seller");

});

it("should Fail if a non-worker tries to finish the auction", async () => {

await auctionInstance.startAuction({ from: accounts[0] });

await auctionInstance.bid({ from: accounts[1], value: web3.utils.toWei("1", "ether") });

try {

await auctionInstance.finishAuction({ from: accounts[1] });

assert.fail("Fail test if caller is not the worker");

} catch (error) {

assert(error.message.includes("revert"), "Fail test if caller is not the worker");

}

});

});

Test Cases Executed - this function was replacing by the series of test cases which was used to evaluate the "Lock" smart contract by assurance of some things, such as contract deployment plus validating contract properties, withdraw function, event emission for withdraw, and fund transfer to contract owner.

Deployment Tests - Verify if a correct deployment of the "Lock" smart contract with correct parameters has been carried out.

1. Ensure that the contract is sent with the original unlocking time.
2. Make sure the contract owner is turned into a variable and changes according to the needs.
3. Ensure the contract is authorized to remove and save the designated funds to the contract account, with deployment.
4. Check a case when the dialogue system will not respond if the unlock time is already in the past.

Withdrawal Functionality test - Test the withdrawal of fund from the "Lock contract" back to the EOA to ensure proper operation.

1. Comprise such that to withdraw would require the unlock condition to be met.
2. Verify recipients are the owner again and the unlocking function reverts if called from a non-owner account after the unlock time.
3. Ensure a withdrawal appears to be a failure if the owner attempts to initiate them after the proposed locking-in period.

Event Emission Tests - validate an event for the availability of the fund withdrawal, that it was successful.

1. Check that the "Withdrawal" event has correct parameters and if funds are being withdrawn.

Fund Transfer Tests - check if the funds are transferred from the "Lock" contact to the RI owner during the withdrawal.

1. Make sure that once you are going to receive the funds, they are exactly the amount that is being described in the contract, which will be transferred from the contract to the owner's account.

Test results - the scripts were executed with zilch problems which generated a successful completion of the scripts to verify the robustness and reliability of the " Pavlo\_Nikolaiev\_test " smart contract. Specifically:

* Deployment: The attempted hack showed that the contract parameters are well set in the proper unlock time and owner properties.
* Funds Handling: Testing of the code showed the ability to accept and store funds within the program automatically.
* Validation of Withdrawal: The program was equipped with interfaces, which were instrumental in ensuring the authentification of withdrawal requests, thus preventing premature or unprivileged access to withdrawal features.
* Event Emission: The "Withdrawal" event emission is indeed being verified upon the withdrawal process execution in the provided contract.
* Fund Transfer: Unlike other technologies, interoperability was achieved without any bugs in the form of smooth switching of funds between contract participants and the owner during withdrawals.

Analysis of Findings - the ran of all test cases is as a reason for the robustness and reliability of security system implemented via "Lock" smart contract. The smart contract is well-rounded enough to capture the essence of various conditions and it is seen to perform any required verification of inputs, states transitions, and other contract specifications.

Implications and Recommendation - The study presents the viability of Hardhat tools for smart contract testing employments in which developers can be rest assured that the implementations have been done with accuracies. Foundations of the Hardhat solution described above as well as of building on it for the smart contracts development projects afterward is the main advice. However, the deployment of auto testing functions into the development process is expected to contribute to the improvement of software quality and reduction of the risk of missing bugs or vulnerabilities.

Conclusion - the output of the proving procedure confirms the validity and consistency of the "Lock" smart contract code. Developers can accomplish verification of smart contract functionality, risk management, and software quality preservation for decentralized applications through Hardhat testing tools utilization.

3.3 Foundry

Foundry is a highly optimizing toolkit used for Ethereum development and it is known for it’s performance and user-friendly features. It offers a single set of tools that can be used for writing, compiling, and deploying smart contracts into Ethereum environment without compromising with the existing tools and libraries.

Fundamentally, Foundry is comprised of several components which include forge the tool used for compiling, testing and deployment of smart contracts, cast which is used to perform interaction with Ethereum. Foundry uses Solidity for contract deployment and most of the testing which helps in having a coherent development workflow. This alignment enables developers to carry out their Solidity actions in writing tests most efficiently and quite clearly.

As a prerequisite to testing, some settings need to be made to the Foundry environment. Developers begin a new project folder which becomes the working directory for the smart contract and related test cases. This is typically done using the command:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "forge-std/Test.sol";

import "./Pavlo.sol";

contract PavloTest is Test {

    Pavlo token;

    address owner;

    address recipient;

    uint256 initialSupply = 1000 \* 10\*\*18;

    function setUp() public {

        owner = address(this);

        recipient = address(0x1);

        token = new Pavlo(initialSupply);

    }

    function testInitialBalance() public {

        assertEq(token.balanceOf(owner), initialSupply, "Initial balance should be the total supply");

    }

    function testTransfer() public {

        token.transfer(recipient, 100 \* 10\*\*18);

        assertEq(token.balanceOf(recipient), 100 \* 10\*\*18, "Recipient should receive 100 TTK");

        assertEq(token.balanceOf(owner), initialSupply - 100 \* 10\*\*18, "Owner balance should be reduced by 100 TTK");

    }

    function testTransferFailsOnInsufficientBalance() public {

        vm.expectRevert("Insufficient balance");

        token.transfer(address(0x2), initialSupply + 1); // Attempting to transfer more than balance

    }

    function testApprove() public {

        token.approve(address(0x2), 200 \* 10\*\*18);

        assertEq(token.allowance(owner, address(0x2)), 200 \* 10\*\*18, "Allowance should be set correctly");

    }

    function testTransferFrom() public {

        token.approve(address(this), 200 \* 10\*\*18);

        token.transferFrom(owner, recipient, 200 \* 10\*\*18);

        assertEq(token.balanceOf(recipient), 200 \* 10\*\*18, "Recipient should receive 200 TTK");

        assertEq(token.balanceOf(owner), initialSupply - 200 \* 10\*\*18, "Owner balance should be reduced by 200 TTK");

    }

    function testTransferFromFailsOnAllowanceExceeded() public {

        vm.expectRevert("Allowance exceeded");

        token.transferFrom(owner, recipient, 300 \* 10\*\*18);

    }

}

Testing Observations:

* Initial Balance Check: The starting balance of the token is checked accurately against the total amount of supply. To be able to execute this test on a contract, there should be an initial contract state deployment done, and this test guarantees that it has been done right.
* Transfer Functionality: The test of the transfer proves that token can be transfer from one address to another. Also, the subsequent check for lack of sufficient balance further affirm the contract capability to handle errors in the case where such transfers are attempted by an unauthorized individual.
* Allowance Mechanism: The approve test verifies that allowances can be set correctly, and the secure method exists to allow third parties to approve the token transfers of the owner.
* Delegated Transfers: The transfer from test checks that delegated transfers work as intended within the contract as the allowances allowed by the token owner’s contract. Moreover, the allowance exceeded test ensures that once again the contract does not allow the owner to withdraw more than needed thereby protecting the owner’s resources.

Performance and Developer Experience:

One noticeable beneficial aspect, related to Foundry’s performance increase is that tests can be executed very quickly, which is useful in cases when multiple contracts and time-consuming test suites have to be used within large lids. Its testing framework developed on Solidity continues the simplified development process which helps developers to concentrate on their work not spending time on the translation from one language to another.

However, Foundry has great docs and people, and since the ecosystem is still new, it is very valuable to have a clear idea of how Ethereum works. It is a community that brings together individuals who share knowledge of standards and best practices to improve the quality of designing smart contracts.

Implications for Smart Contract Development- From testing of Pavlo smart contract using Foundry, it is clear that adequate testing is vital in the deployment of smart contracts. Using tools such as Foundry to raise the bar in running tests helps the developers to ensure they are dealing with contracts performing as expected for the overall success of the blockchain.

This way the results show that Foundry is not only a tool for discovering problems but also for developing confidence in the contracts that have been deployed. This confidence is important given the dynamic technological environment in which decentralized application developers aim at developing robust and Secure dApps.

And thus, on balance, Foundry becomes quite a valuable tool in the toolkit for Ethereum developers tightening the loop of the smart contract development cycle from writing to testing to deployment. Among its common features there is performance, simplicity of use, and integration with Solidity, that makes Fox as an essential tool for anyone engaged in Ethereum development.

list of references

*All information sources referenced in Chapters 1 and 2 shall be described here.*

*Information sources and references must be drawn up in accordance with the faculty’s guidelines:*[*https://ebooks.rtu.lv/product/formatting-and-style-guidelines-for-graduation-thesis/?lang=en*](https://ebooks.rtu.lv/product/formatting-and-style-guidelines-for-graduation-thesis/?lang=en)

Kimani, DK, Kweku, KA. Rexford ,AR , Ullah,SU, Frecknall-Hughes,JF, Kim JK, *Blockchain, business and the fourth industrial revolution: Whence, whither, wherefore and how?* Edition. ScienceDirect,pp. 1-3, 2020 [viewed 13 of December 2023]

<https://www.sciencedirect.com/science/article/abs/pii/S0040162520310805>

Leng J. , Zhou M. , Zhao L. , Huang Y., *Blockchain Security: A Survey of Techniques and Research Directions,* IEEE, pp. 2490 – 2510, 2020 [viewed 13 of December 2023]

<https://ieeexplore.ieee.org/abstract/document/9271868>

Parasu JP , Reddy VR , *Blockchain Testing* , MFX, pp. 2-7 ,2019 [viewed 13 of December 2023]

<https://www.qteksystems.com/wp-content/uploads/2019/11/Block_Chain_Testing_Final.pdf>

Eder J, Hegnauer T, Rodrigues B ,*Bifröst: a Modular Blockchain Interoperability API ,*Ieeexplore, 2020 [viewed 13 of December 2023]

<https://ieeexplore.ieee.org/abstract/document/8990860>

Lal C, Marijan D , *Blockchain Testing: Challenges, Techniques, and Research Directions* , Corner University , 2021 [viewed 13 of December 2023]

<https://arxiv.org/abs/2103.10074>

Ali M , Nelson J,Blockstack: *A Global Naming and Storage System Secured by Blockchains* 2016, [viewed 13 of December 2023]

<https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali>

Zheng Z ,Xie S, Dai H, Chen X , Wang H, *Blockchain challenges and opportunities: a survey* 2018 [viewed 13 of December 2023]

<https://www.inderscienceonline.com/doi/abs/10.1504/IJWGS.2018.095647>

Yli-Huumo J , Ko,Sujin Choi D , Park S, Smolander K , *Where Is Current Research on Blockchain Technology?—A Systematic Review* PlosOne, 2016 [viewed 13 of December 2023]

<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0163477>

Kõlvart M, Poola M , Rull A , *Smart Contracts ,* Springer link, 2016 [viewed 13 of December 2023]

<https://link.springer.com/chapter/10.1007/978-3-319-26896-5_7>

Luu l , Chu D , Olickel H , Saxena P, Hobor A , *Making Smart Contracts Smarter* , Digital library , 2016 [viewed 13 of December 2023]

<https://dl.acm.org/doi/abs/10.1145/2976749.2978309>

Kuznetsov, A.; Kuznetsova, K.; Babenko, V. , *Statistical Testing of Blockchain Hash Algorithms* , ODMU , 2019 [viewed 13 of December 2023]

<https://www.odmu.edu.ua/xmlui/handle/123456789/12276>

Lan C, Sherry C, Popa R, Ratnasamy S, *Embark: Securely Outsourcing Middleboxes to the Cloud* , USENIX , 2016 [viewed 13 of December 2023]

<https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/lan>

Mossberg M, Manzano F, Hennenfent E , *Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts* IEEE , 2020 [viewed 13 of December 2023]

<https://ieeexplore.ieee.org/abstract/document/8952204>

Leid A., Merwe B., Willem Visser., *Testing Ethereum Smart Contracts: A Comparison of Symbolic Analysis and Fuzz Testing Tools,* Association for Computing Machinery, 2020 [viewed 13 of December 2023]

<https://dl.acm.org/doi/abs/10.1145/3410886.3410907>

Yanovich Y., Ivashchenko i., Ostrovsky A., Shevchenko A., Sidorov A.,

*Exonum: Byzantine fault tolerant protocol for blockchains* , EXONUM ,pp. 1-4 , 2019

[viewed 13 of December 2023] ,

<http://www.webprime.co.kr/renderfarm/content/downloads/wp_consensus_181227.pdf>

Brown R,, Carlyle J., Grigg I, Hearn H. *Corda: An Introduction* ,pp. 3-8, 2016 [viewed 13 of December 2023]

<https://www.smallake.kr/wp-content/uploads/2016/10/corda-introductory-whitepaper-final.pdf>

Andoni M,, Robu V., Flynn D., Abram S, Geach D, Jenkins D., McCallum P., Peacock A , *Blockchain technology in the energy sector: A systematic review of challenges and opportunities* , ScientDerect ,pp.1-20, 2018 [viewed 13 of December 2023]

<https://www.sciencedirect.com/science/article/pii/S1364032118307184>

Koul R, *Blockchain Oriented Software Testing - Challenges and Approaches ,* IEEE , 2018 [viewed 13 of December 2023]

<https://ieeexplore.ieee.org/abstract/document/8529728>

 Walker A.,  Schmidt C.,  Dubey A. *Testing at scale of IoT blockchain applications ,* ScientDerect ,pp. 1-3, 2019 [viewed 13 of December 2023]

<https://www.sciencedirect.com/science/article/abs/pii/S0065245819300348>

Barboni M,, Morichetta A., Polini A., *A mutation testing approach and tool for the Ethereum blockchain ,* ScientDerect ,pp.1-3, 2021 [viewed 13 of December 2023]

<https://www.sciencedirect.com/science/article/abs/pii/S0164121222001418>

Li D, Wong W, Guo J, A ,*Survey on Blockchain for Enterprise Using Hyperledger Fabric and Composer* , IEEE , 2020 [viewed 13 of December 2023 ]

<https://ieeexplore.ieee.org/abstract/document/9045815>

IBM, *What is blockchain?*, IBM, pp. 1-11, 2021[viewed 14 of February]

<https://www.ibm.com/topics/blockchain>

S. Dong & A. Khushnood & M. Li & J. Kamruzzaman, *Blockchain technology and application: an overview ,* CC BY 4.0,pp. 1 -39,Available from : doi: 10.7717/peerj-cs.1705

<https://www.researchgate.net/publication/376051976_Blockchain_technology_and_application_an_overview>

W. Metcalfe , *Ethereum, Smart Contracts, DApps* , ResearchGate ,pp.77-8- 2020[viewed 14 of February 2024]

<https://www.researchgate.net/publication/340652603_Ethereum_Smart_Contracts_DApps>

Atzei, N., Bartoletti, M., Cimoli, T , *A survey of attacks on Ethereum smart contracts (SoK)*. pp. 1-16 , 2017 [viewed 14 of February 2024]

<https://eprint.iacr.org/2016/1007.pdf>

Parizi, R.M., Dehghantanha, A., Choo, K.K.R., Singh, A. , *Empirical vulnerability analysis of automated smart contracts security testing on blockchains*. Cornell University , 2018 [viewed 14 of February 2024]

<https://arxiv.org/abs/1809.02702>

Porru S. , Pinna A. , Marchesi M. , Tonelli R. , *Blockchain-Oriented Software Engineering: Challenges and New Directions* , ResearchGate ,pp. 1-4, 2017 [viewed 14 of February 2024]

<https://www.researchgate.net/publication/318123530_Blockchain-Oriented_Software_Engineering_Challenges_and_New_Directions>

Zheng Z ,Xie S, Dai H, Chen X , Wang J , Imran M. , *An Overview on Smart Contracts: Challenges Advances and Platforms ,* arXiv:1912.10370v1 ,pp. 1 - 16, 2019[viewed 15 of February 2024]

<https://arxiv.org/pdf/1912.10370.pdf>

Suliyanti N. , Sari F. , *Blockchain-Based Implementation of Building Information Modeling Information Using Hyperledger Composer ,* ResearchGate , pp. 1-12, 2021[viewed 15 of February 2024]

<https://www.researchgate.net/publication/348146232_Blockchain-Based_Implementation_of_Building_Information_Modeling_Information_Using_Hyperledger_Composer>

Staroletov S., Galkin R. , *Testing Methods for Blockchain Applications* , ResearchGate,pp.1-4, 2022[viewed 20 of February 2024]

<https://www.researchgate.net/publication/357898109_Testing_Methods_for_Blockchain_Applications>

Staroletov S., Galkin R. , *Testing Methods for Blockchain Applications* , ResearchGate,pp.1-4, 2021[viewed 26 of February 2024]

<https://www.researchgate.net/publication/366307220_Testing_Methods_for_Blockchain_Applications2>

Algarni A., Attaallah A., Eassa F., Khemakhem M., *A security testing mechanism for detecting attacks in distributed software applications using blockchain*, ResearchGate ,pp 1-15,2023[viewed 26 of February 2024]

<https://www.researchgate.net/publication/367303034_A_security_testing_mechanism_for_detecting_attacks_in_distributed_software_applications_using_blockchain>

Bitcoinj ,*What is bitcoinj?,* bitcoinj,2009[viewed 26 of February 2024]

<https://bitcoinj.org/>

Truffle suite , *TRUFFLE* , Truffle suite,2020[viewed 26 of February 2024]

<https://archive.trufflesuite.com/docs/truffle>

Gao J. , Liu H. ,Li Y., Liu C., Yaung Z , *Towards Automated Testing of Blockchain-Based Decentralized Applications*, IEEE,2019 [viewed 18 of March 2024]

<https://ieeexplore.ieee.org/abstract/document/8813282>

Hardhat, *Ethereum development environment for professionals,* hardhat, 2023[ viewed 29 of March 2024] , <https://hardhat.org/>

Gattermayer J., Pˇrevr´atil M., Kubˇena J,. *Evaluating Testing Frameworks for Ethereum Blockchain Applications: Introducing Wake.* Ackee.2024, [ viewed 29 of March 2024] <https://ackee.xyz/Wake_testing_framework_Whitepaper_draft_1.pdf>

Foundry, *Foundry Book* , 2023 <https://book.getfoundry.sh/>