DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis Proposal in Informatics

Automated Unit Testing of Solidity Smart Contracts in an Educational Context

Automatisiertes Unit Testing von Solidity Smart Contracts im Bildungskontext

Author: Batuhan Erden
Supervisor: Prof. Dr. Florian Matthes
Advisor: Felix Hoops, M.Sc. & Burak Öz, M.Sc.

Contents

1 Introduction

Blockchains have recently become increasingly prevalent, revolutionising how information is stored and processed. The concept of this decentralised digital record system has gained significant attention due to its revolutionary nature. With the emergence of decentralised transactions, the popularity was further augmented and helped the development of the first example of Blockchain technology, Bitcoin. Since then, Blockchain technology has grown even further across several industries, including finance and healthcare, and allowed anyone to partake in the Blockchain world with its decentralised and accessible nature.

The emerging Blockchain technology, a rapidly approaching phenomenon, supports a decentralised computing paradigm shift. [1] This technology has evolved even further to leverage its decentralized nature for creating smart contracts, which are simply programs stored on the Blockchain that enforce specific rules and check whether specified conditions are met before any transaction. Ethereum is the most popular and widely used cryptocurrency supporting smart contracts. Since smart contracts are encoded in the Blockchain system, they benefit from a tamper-resistant and transparent environment, allowing them to enforce a set of terms based on specific conditions without requiring third-party involvement. As this thesis focuses on the educational context, our goal is to educate students through unit-tested exercises. This approach will help them understand the significance of developing robust smart contracts that can pass multiple crucial tests. These tests ensure that the smart contracts work as intended and are free of vulnerabilities, as even a minor error in the code could result in serious consequences. For instance, in financial cases involving numerous non-refundable transactions, errors would result in defective transactions that cannot be reversed. Therefore, it is essential to compare the capabilities of popular Ethereum smart contract development tools, hereafter referred to as *test runner frameworks*, to determine the functionalities for smart contract development and testing. This comparison can help create a service that caters to the requirements of our *educational/academic use case*, which we will discuss more in the next section.

Finally, the status quo for automated unit testing for smart contracts is evolving rapidly as the Blockchain ecosystem is still nascent and continually growing. Consequently, many developments aim to facilitate a more comprehensive selection of test runner frameworks. Those frameworks are progressively refined to become more robust and adaptable to numerous architectures. The main goal of this thesis is to successfully design, implement, and evaluate a scalable service for automated smart contract unit testing. The outcome should be compatible with our *educational/academic use case* where students submit smart contract inputs to the service. These inputs are tested with specific unit tests to provide insights into their smart contract's performance.

2 Problem Statement

This thesis aims to develop a service to provide developers with a modular and scalable approach to testing their smart contracts, allowing them to gain valuable insights into the efficiency and functional correctness of their smart contract implementations through performance metrics and test results. Considering all the problems intended to be fixed with this work, the service should be compatible with our *educational/academic use case*, which involves students submitting their smart contract inputs to the service to receive correctness and performance metrics via unit tests. In our specific use case, we will be deploying this service to be utilized by the students of the *Blockchain-based Systems Engineering* course and the unit tests will be created by hand by teaching staff for every exercise to assess functional correctness.

First, selecting the best-performing test runner framework is essential in developing an efficient and scalable service that can handle simultaneous requests without crashing using message queue platforms (e.g., RabbitMQ). It is crucial to consider the final evaluation criteria for the frameworks, which include speed and efficiency, their ease of deployment as Docker containers, possibly in a Docker swarm, and how easy they are to maintain and scale throughout the software development process. To enhance convenience and security and enable the spawning of other containers, deploying the service as a Docker container is preferable. However, the feasibility of Docker in a Docker needs to be explored further. Defining and developing a viable alternative solution to achieve similar functionality is essential if this approach proves impractical.

Second, the developers who use this service should be informed about how their smart contracts have performed. Choosing performance metrics requires further research. However, widely-used metrics like gas usage, the amount of on-chain storage, and execution time can be practical in evaluating smart contract performance. Giving as many metrics to the user as possible is more convenient. However, the two primary questions that need to be addressed are: first, which metrics are of interest, and second, what are the options for getting all the data points of interest from each test runner framework?

Third, to protect the service from attack attempts, it is necessary to adopt secure coding practices and develop the service considering potential attack vectors that may arise from the tested smart contract. This thesis focuses on building a service for smart contract development and testing rather than securing the contracts themselves, which would be out of the scope of this work. Therefore, the most crucial security measure is safeguarding the service's integrity from any compromise caused by the code submitted to the service, even though we do not expect much danger from the code under test. At first, the errors in the submitted smart contracts should be identified and handled. If a smart contract cannot be compiled, the service should not be affected and disrupted, and the developer submitting the contract should be informed. Second, there should be a limit on how often students can submit smart contracts, and the optimal duration for running individual smart contracts should be set so that they will not keep the service busy. Third, using gas limits as a security feature is likely possible. For instance, we could set a daily gas limit; if a developer/student exceeds that, they will not be allowed to submit further on that day. Last is how the service can be protected against accidental Distributed Denial of Service (DDoS). We assume that any DDoS attack will be accidental since these services will be deployed with rate limits set for student submissions. Additionally, being behind a university login might help protect the system from such attacks.

In a nutshell, the result of this study will help develop a comprehensive understanding of the strengths and weaknesses of different test runner frameworks. Ultimately, the thesis aims to create a secure, efficient, and easily-scalable service that provides a seamless and practical approach to testing smart contracts, enabling developers to produce reliable, secure, and scalable smart contracts. The service should always be available, eventually consistent, efficient, and protected from any errors in the submitted smart contracts that may cause the service to break and become unavailable.

3 Research Questions

This thesis will answer the following research questions through research and practical work.

- **RQ01:** What are the requirements for educational unit testing?
  a) What is the core use case?
  b) What are exemplary exercises that we would like students to do?
- **RQ02:** What is the status quo in automated smart contract testing?
  a) Are there examples of smart contract testing as a service?
  b) What are the most commonly used tools with smart contract testing capabilities?
  c) How can we characterize those tools in terms of what key features they have and how they can measure performance?
- **RQ03:** What do we have to consider regarding security and stability when using a testing tool in a way that is not entirely intended?
  a) How can errors and crashes in the contract execution be handled?
  b) What measures do we need to prevent accidental or intentional system overload?
- **RQ04:** How can a learning platform giving feedback through automated smart contract unit testing be developed?
  a) What considerations need to be made to ensure the service is scalable and expandable?

4 Methodology

Through this work, the following approach will be taken to answer each research question defined in the previous section. Even though the project steps may resemble a waterfall model, we intend to stick to a more agile model. We plan to release a functional Minimum Viable Product as soon as it meets the minimum requirements. Then, with the advisors, we will act as the end users, test the service in each iteration, and then incorporate iterative updates. Here is a clear high-level overview of the methodology: Here is a clear view of the high-level methodology:

1) **Literature Review:** The requirements and the problem must be well-defined before starting the development phase. The approach we plan to take begins with a thorough literature review of related work and how others have previously used automated smart contract testing. It is vital to familiarise oneself with previous work on comparing different test runner frameworks and to find out what strategies other researchers have used to determine the advantages and disadvantages of these different frameworks. In addition, what is also critical is the performance metrics used by others to rank or score smart contracts. Selecting metrics will depend on past validation, what others have proved, and compatibility across all test frameworks employed. We are also planning to explore the possible attack vectors against the service's integrity that have already been tried or are worth noting. Last but not least, through both scientific and practical work, different ways of building such a modular service for smart contract testing will also be researched.

2) **Requirements & Problem Analysis:** Before implementing the end service, we will compare various test runner frameworks and how convenient they would be for automated smart contract testing. The most widespread ones, which can be candidates in this work, are Foundry, Truffle, Waffle, and Hardhat, as well as Mocha and Chai, with the possible inclusion of other frameworks emerging during the research. With this analysis, the strengths and limitations of each test runner framework can be assessed to offer valuable insights into its effectiveness and suitability for our *educational/academic use case*. The final evaluation criteria for the frameworks should include the speed and efficiency, ease of deployment as Docker containers, possibly in a Docker swarm, and manageability throughout the whole software development process as we build this service to be easily maintainable and scalable. In conclusion, we want to stress the criticality of maintaining the security and integrity of our service. The containers for testing smart contracts must prevent unauthorized access or remote code execution. They should solely serve the purpose of testing the contracts and providing metrics to developers.

3) **Local Setup & Development:** At this phase, I plan to work with as many test runner frameworks as possible to collect enough data to compare them and discover the most feasible one. Once enough information is gathered, the appropriate test runner frameworks will be set up and tested locally by writing a common test case in compatible formats that each of them understands or different test cases for different frameworks. Only after we have all the frameworks set up to run in the local environment will we be able to look for ways to deploy them as Docker containers since Docker is probably the most convenient way. Once everything is set, we will benchmark each framework locally to find the most promising framework. After selecting the frameworks to work with, a worker service should be implemented, preferably in the form of another Docker container that handles the entire pipeline from processing user input to creating and destroying containers. This worker service will be implemented as a Node.js or Django application based on a particular design selection process to enhance manageability and performance. Furthermore, this primary worker service can initiate sub-containers within its container, Docker in Docker, or implement a workaround to achieve the same functionality. With the containers readily available in the local environment, the work can now commence with the initial deployment of said containers. At this point, it should already be clear how much computing power the system we will be deploying this service will need. Accordingly, it is probably the best time to deploy the service and look for ways to make it both easily scalable and manageable.

4) **Remote Deployment & Data Storage:** Initially, the initial requirement is to make the system, which we will be deploying the container, powerful enough to process a single request by executing the smart contract inputs with the predefined unit tests. The results returned by the service after execution will include general performance metrics compatible with any test runner framework and perhaps additional information, if applicable. In addition, each run's results should be stored in a database and returned via the query. The choice of database, which will most likely be a NoSQL database, will be based on its suitability for the test service. In addition, the service should also provide high maintainability, allowing for easy incorporation of new features and updates, for which we will try to design a Continuous Integration/Continuous Deployment (CI/CD) pipeline to manage the service build following every alteration.

5) **Scalability:** Once the service is ready for single synchronous requests, the next step is to prepare it for scalability and concurrent request handling. We anticipate the service will handle the load of approximately *400-500* developers, most of whom will likely submit their contracts during peak times. For instance, in the *educational/academic use case*, students must submit their contracts before a pre-defined deadline. It is anticipated that most submissions will occur during the final hours leading up to the deadline. Furthermore, the service should be ready to handle this load. To achieve this, we plan to incorporate a message queue into the service to manage the flow of requests to the service. Depending on the server's capacity, this queue will balance the load by executing the contracts individually or in small batches. On the other hand, the service should also be easily scalable in case there are more users in the future. To do so, we plan to complement the service by including Docker Swarm to distribute containers automatically across the cluster, should there be any, as this will allow the service to be scaled up or down as needed.

6) **Security & Stability:** Ensuring the integrity of the service from any submitted code is essential. The service must identify and handle errors in submitted smart contracts, including those that cannot be compiled. Additionally, we can use gas limits or submission frequency limits to control code submissions and set runtime durations for smart contracts. We should also evaluate the service's performance in the event of an accidental DDoS, should it occur. Secondly, we plan to incorporate prior authorization for the service, a basic authentication and authorization layer. This authorization layer will likely utilize JWK (JSON Web Key) to ensure compatibility with our use cases and streamline the authentication process.

7) **Evaluation:** The final evaluation of the proposed solution involves assessing its effectiveness in addressing the abovementioned problems. The message queue should allow the service to handle a substantial workload without failures and deliver accurate test results while maintaining high availability. Additionally, it might be insightful to do some scaling analysis. Ultimately, developing a simple backend application that uses this service would be an excellent way to demonstrate and test that the service is working as expected.

The findings and results will be used in the thesis to demonstrate that the developed service is practical, powerful, and trustworthy for users and instructors.

5 Starting Literature

We wanted to begin our literature review by examining the security of dockerised systems, the metrics used to evaluate the performance of smart contracts, and the basic techniques for developing a tool for automated smart contract testing.

- **Useful Performance Metrics for Smart Contract Testing:** Several metrics have been evaluated in the scientific community to assess the performance of smart contracts. These metrics include contract execution time, gas cost (i.e., the fee for executing a smart contract), and block state update time. [2] Another performance metric that can be useful for evaluating the effectiveness of a testing tool is instruction coverage. This metric has been employed in various research papers to assess a tool's ability to detect vulnerabilities by counting the number of executed instructions. The idea is that the more code a tool executes, the higher the likelihood of discovering vulnerabilities. [3]
- **Automated Smart Contract Tool Building:** The work of Solorio, Kevin and Kanna, Randall and Hoover, and David H demonstrates that it is viable to develop and deploy effective tools, which continue to apply to this day. Their work showcases promising performance results. [4] Foundry, one of the most promising smart contract development tools, has demonstrated its containerised effectiveness. [5]
- **Security (Nice to have):** Security is one of the critical aspects of cloud applications today. Docker also suffers from various security attacks, especially Container Breakouts, where an attacker breaks out of a container to get access to the host and other containers. [6] It is also possible to abuse Docker API to do remote code execution, which can be mitigated by several technics, like not allowing access to Docker daemon over TCP/HTTP. [7]

6 Schedule

The following schedule must be followed to ensure the success and timeliness of this thesis. Here is the proposed schedule for this thesis:

Period (6 months)	Expected Work
[May 15 - June 15)	Gathering enough information required to build the modular system by reviewing related work.
[June 15 - July 30)	Completing the basic setup of the service using different smart contract development tools such as Foundry, Truffle, Waffle, and Hardhat, with the possible inclusion of other frameworks emerging during the research.
[July 30 - August 30)	Deploying the service to the cluster and ensuring it can handle a single request and produce the expected results.
[August 30 - September 30)	Making the deployed system suitable for intense load and using other technologies like RabbitMQ and Docker Swarm to ensure concurrency, availability, and scalability. Moreover, the service should be secured against faulty smart contract inputs and must be kept available at all times by setting a submission limit or using gas limits.
[September 30 - November 15)	Doing the evaluation and testing, putting everything down into writing, and gathering results and comparisons to prove that everything works as expected. In this period, the thesis should be written and completed.

Bibliography

[1] R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, and A. Singh. "Empirical vulnerability analysis of automated smart contracts security testing on blockchains". In: *arXiv preprint arXiv:1809.02702* (2018).

[2] R. C. Lunardi, H. C. Nunes, V. d. S. Branco, B. H. Lipper, C. V. Neu, and A. F. Zorzo. "Performance and cost evaluation of smart contracts in collaborative health care environments". In: *arXiv preprint arXiv:1912.09773* (2019).

[3] M. Ren, Z. Yin, F. Ma, Z. Xu, Y. Jiang, C. Sun, H. Li, and Y. Cai. "Empirical evaluation of smart contract testing: What is the best choice?" In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2021, pp. 566–579.

[4] K. Solorio, R. Kanna, and D. H. Hoover. *Hands-on Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment*. O'Reilly Media, 2019.

[5] *Foundry Tutorial: Docker*. `https://book.getfoundry.sh/tutorials/foundry-docker`. [Accessed: March 11, 2023].

[6] R. Yasrab. "Mitigating docker security issues". In: *arXiv preprint arXiv:1804.05039* (2018).

[7] M. Cherny and S. Dulce. "Well, that escalated quickly! how abusing docker api led to remote code execution, same origin bypass and persistence in the hypervisor via shadow containers". In: *BlackHat* 17 (2017), p. 50.