Block-Chain Oriented Software testing approach

Ashray Kakadiya

Student, LDRP Institute of Technology and Research, Gandhinagar, Gujart

---

ABSTRACT - The block-chain technology presents a very innovative and secure way of managing transactions online. Hailed as one of the greatest inventions after the Internet, it has taken the digital world by storm and is disrupting many industries. In the past few years this technology has gained enormous importance and its application area has evolved into a wider context. The mass adoption of block-chain based applications has increased dramatically and a plethora of such applications are now available for use. As a result, block-chain based software development is also growing at a staggering rate.

This paper acknowledges the need for software engineers to devise specialized tools and techniques for block-chain oriented Software testing. The aim of this paper is to develop testing methodology for Block-chain oriented software as currently there is no such approach available in literature. It also highlights the challenges currently faced as well as the approaches followed for testing such applications in order to ensure high standards of quality.

Keywords - Block-Chain, Testing, Software testing lifecycle, Block-chain Oriented Software.

1. INTRODUCTION

Blockchain provides users with a safe and secure way of managing their transactions online [1]. It creates an environment of trust without the need of any external middle parties. It is a distributed ledger which is shared, replicated and synchronized among the members of a public or private peer-to-peer network. The ledger permanently records the history of asset exchanges amongst the members of the network in a linear and chronological order. Every transaction recorded in the ledger has a timestamp and unique cryptographic signature associated with it. Once the information gets stored in the blockchain, it cannot be changed or tampered. All the confirmed and verified transactions are combined into a block and chained to the most current block to form a blockchain.

The blockchain technology presents a completely new approach to software development. It's decentralized nature along with the anonymous nature of the nodes involved further adds to the complexity of the testing process. A traditional method of Software testing may not be valid anymore. The immutable nature of the blockchain further implies that if a bug goes into the production system, it may require complete revision of the code. Thus, using correct testing techniques and methodologies becomes more critical in this case [2].

A primary factor that influences the level of testing is whether the implementation is based on a public platform like Ethereum or customized platform that is built for an organization or consortium of organizations. In case of a private blockchain it is somewhat easy to simulate all the scenarios and test them internally. Since private blockchain operates in a controlled environment traditional testing method can prove handy. A detailed test strategy can be designed since the functionality is customized.

The complexity escalates when the implementation is on a public platform. In public blockchain implementation there is no upper limit on the nodes that can participate, nodes can join and leave in an ad-hoc manner, consensus may not be reached easily lowering the speed of transactions, a hard fork may get created and many more issues [2]. It becomes very difficult to visualize and design test strategies and test cases covering all aspects.

In this paper, we are proposing a detailed four phases testing lifecycle designed specially for blockchain oriented soft-wares. Section 2 of this paper talks about the previous work done in this field of blockchain testing. Section 3 contains our detailed proposed solution of Testing lifecycle. In section 4 we conclude our paper and provide possible future works that can be carried out further on our proposed solution. Section 5 and 6 is Acknowledgment and References used for this paper.

2. LITERATURE REVIEW

A block chain is a form of data structure where information is stored with some additional information of validation. Most applications that are developed on the block chain must guarantee data integrity and uniqueness to ensure blockchain-based systems are trustworthy which, in the case of block chain-oriented Software (BOS) is that of security-critical systems.

There is a need for testing suites for BOS. General suites include [3]: 1. Smart Contract Testing (SCT), namely specific tests for checking that smart contracts i) satisfy the contractor's specifications ii) comply with the laws of the legal systems involved, and iii) Do not include unfair contract terms. 2. Blockchain Transaction Testing (BTT), such as tests against double spending and to ensure status

integrity. There are various tools which uses automated testing for smart contracts which is as important as any other kind of software testing, verifying if necessary hooks are present so that external automated scripts can instruct the platform, observe the outcome, and verify that the outcome is as per expectation.

In the banking systems if we don't have these hooks then smart contracts functioning would be difficult. There are now multiple smart contract testing frameworks for Ethereum [4]. The alternative, and more recent, approach was proposed by the people at Eris Industries, a New York-based smart contract software design firm. They suggested testing Ethereum contracts using other Ethereum contracts [5]. We will still need scripted interactions to kick the process off, but effectively the test logic is built into the contracts themselves. Infosys issued a white paper titled "Assuring success in blockchain implementations by engineering quality in validation" [6]. In this paper, they discuss various challenges in testing blockchain implementations and list out the testing phases with the volume of tests, methodology and tools. They also discuss test strategy across various test phases with a call-out on the key activities.

Another paper shows design of Land Administration and Title Registration Model Based on Blockchain Technology [7]. In this paper, they discuss how to use blockchain technology and propose a model to perform testing for Verification using Markov Chain. Markov chain is a widely recognized approach to guarantee the correctness of a system by checking that any of its behaviors is a model for a given property.

As Blockchain Oriented Software projects work with the blockchain technology which is distributed in nature, testing is done in isolation and requires proper mocking of objects capable of effectively simulate the blockchain. The bitcoin [8] is the first and most popular cryptocurrency. It is a blockchain oriented software and has been receiving a lot of attention [9]. One of its technical features is that it enables reliable transactions without a centralized management mechanism even if there are unreliable participants in the network, and this feature is obtained by the invention of blockchain technology.

In our research paper, we plan on creating a complete Software testing life cycle to test BOS projects like Bitcoin. Unlike the methods discussed above which focuses on testing only particular key functionalities like Smart Contracts and double spending, we propose a new Software Testing Life Cycle which will test the software in all perspectives.

3. PROPOSED SOLUTION

In our proposed solution, we plan on creating a complete Software testing life cycle to test BOS projects like Bitcoin described above. Unlike the implemented works, which focuses on testing only particular key functionalities like Smart Contracts and double spending, we propose a new Blockchain Oriented Software Testing Life Cycle which will test the software in all perspectives. The testing lifecycle has four phases as shown in Fig 1.

3.1 Phase 1: System Overview

The first phase of our BOS testing lifecycle is the system overview phase. We suggest that there should be an early involvement of the testers in the SDLC so that they have a better idea of all components involved and also about which team is reaponsible for which component. Then a component map is generated. This component map contains all the components and sub components of the complete system including all the interfaces. It gives a good idea of the overall working of the system.

From this complete componet map, a system component map is generated, which contains the shortlisting of all the components that pertain to blockchain technolody. The components shortlisted are again mapped into a component diagram and this defines the scope of testing. Once the scope of testing is define, the whole team has a clear idea of what is to be tested and which team is responsible for which component. The output of phase 1 is System component map determining the testing scope.

3.2 Phase 2: Test Design

In phase 2, a detailed level test strategy needs to be designed specific to blockchain. We identify the key components that need to be checked in the system. We are using a model called hyper ledger composer to test the block chain oriented software's. It has its own modeling language. The output of the second phase will be a detailed level test strategy.

3.2.1 Model of structure of blocks, transactions and contracts is designed for testing

Hyper ledger Composer includes an object-oriented modeling language that is used to define the domain model for a business network definition. Hyper ledger Composer CTO file is composed of the following elements:1.A single namespace. All resource declarations within the file are implicitly in this namespace. 2.A set of resource definitions, encompassing assets, transactions, participants, and events. 3.Optional import declarations that import resources from other namespaces.

The CTO modeling language is tightly focused with just a few keywords. The model for your business network resides in a file that has a .cto file extension, and contains definitions for the following elements: namespace, resources, imports from other namespaces, as required. *Model the business network* : Hyper ledger Composer only allows working with one model at a time. *Instantiate the model*: The Assets and Participants from

the model appear on screen, saying the registry is empty. when the business network is first created, both the asset and participant registries are empty. You need to create asset and participant instances, and those instances will reside in the registry.

The following steps show how to instantiate and test the model. •Test the business network: Models are great at acting as a sort of blueprint for the application you are building, but a model of a thing is not much good unless it results in an actual thing. For the business model, thing that useful is need to be instantiated. •The Asset and Participant registries: It's time to instantiate the resources, and their instances will live in their respective registries. So, asset instances go in the asset registry, and participant instances go in the participant registry.

The perishable-network model includes a transaction implemented as a JavaScript function in the library module. That you can use to instantiate the model and create entries in the asset and participant registries. It is provided as a way to get the business network from the template up and running more quickly than if you entered the model by hand. It does three things:

1) Creates instances of all the assets and participants from the model.

2) Sets property values on those instances.

3) Stores the instances in their respective registries.

3.2.2 Create use cases for various interactions with the system and endpoints.

Along with the hyper ledger we are also going to create use cases and generate sequence diagrams corresponding to use cases and check if requirements are met. As we know that we can identify the missing requirements/steps easily with the sequence diagram flow, since each activity in life line are related in performing the activities. So, missing requirements can cease to happen the activities and help us to identify and modify use cases.

3.2.3 Testing block-chain related NFRs

Be agile : NFR testing is planned from the beginning of the project. For example, if there is a requirement like application should handle heavy traffic , say 1000 requests per second. Then we use the technology like multi-cast and send data across network to several users at the same time and very the application handling.

Plan, Prioritize : Begin agile itself isn't enough. Planning is important regarding which NFR test should be performed when we get constrained by time or resources.

Setup : It is important to have proper environment. For example, when a system has to handle 100tps with 2 cpu and 4 cores, But in the production we have 4 cpus and 8 cores in these circumstances an test can result two outcomes. 1) you will get good results 2) We will get bad results-which is waste of time and manpower assuming some issue with the app.

Record : Recording NFR is important, because we can prove our system performance and go back in future when we learn something new.

3.3 Phase 3: Test Planning

In this phase, we get a low level view of how every form of testing is to be conducted is decided to have an estimate of number of tests at every level and also the amount of coverage. We check the system availability or we can say testing environment availability. If system is not available then alternative testing strategies need to be planned. Alternative test strategy involves setting up a private blockchain for testing.

There has to be an estimate on coverage and number of tests that will be performed It is mandatory to determine the volume of tests amd test tools and automation. Every level of testing is considered and its Testing Methodology and tools and finalized.

Table 1 gives an example of every testing phase/level along with it's recommended methodology and tools. Table 2 gives an estimate of Volume of tests that are performed at every testing phase or testing level.

Table 1. Testing levels with methodology and tools

Testing Level	Methodology and tools
Unit Testing	Test Driven Development (Mock Stubs)
System Testing	Verifying contracts, blocks and updating, etc. through scripts (Black box)
Integration Testing	TestNet (used primarily for testing Bitcoin-related applications)
Functional/UI Testing	Automated tests for front end (Selenium scripts)

Table 2. Testing Levels with Volume of tests

Testing Level	Volume of Tests
Unit Testing	2500
System Testing	1000
Integration Testing	275
Functional/UI Testing	50

Use Cases that are designed in the second phase are mapped to the tests mentioned above. This makes sure that we have covered all the test scenarios and have included all the user requirements inclusion of user scenario. Output of phase 3 is a Final test strategy and a document of test cases.

3.4 Phase 4: Test Execution and Result Verification

The fourth phase is the last phase of the proposed Blockchain oritented software testing life cycle. It involves executing all the tests at every testing level with the documented methodody and tools from phase 3. Execution can be ideally automated with scrippting which follows a test driven development approach on a suitable framework.

Various key activites that need to be focused in this phase are low level verification, and validation of blocks, Smart Contracts and Transactions. We also need to test all the third party interfaces that are used in the system as well as the user interface and functional flows.

The results then need to be consolidated, analyzed and verfied back to the business side. There has to be a bug report which lists all the defects identifies as well as a detailed test report stating passed and failed test executions.

Table 2. Testing phase with methodology and tools

Testing Phases	Methodology and tools
Unit Testing	Test Driven Development (Mock Stubs)
System Testing	Verifying contracts, blocks and updating, etc. through scripts (Black box)
Integration Testing	TestNet (used primarily for testing Bitcoin-related applications)
Functional/UI Testing	Automated tests for front end (Selenium scripts)

The output of Phase 4 is the test results and defects report which is sent to the team for further processing. This cycle can be carried out until the system works as expected and there are no critical errors found during testing.

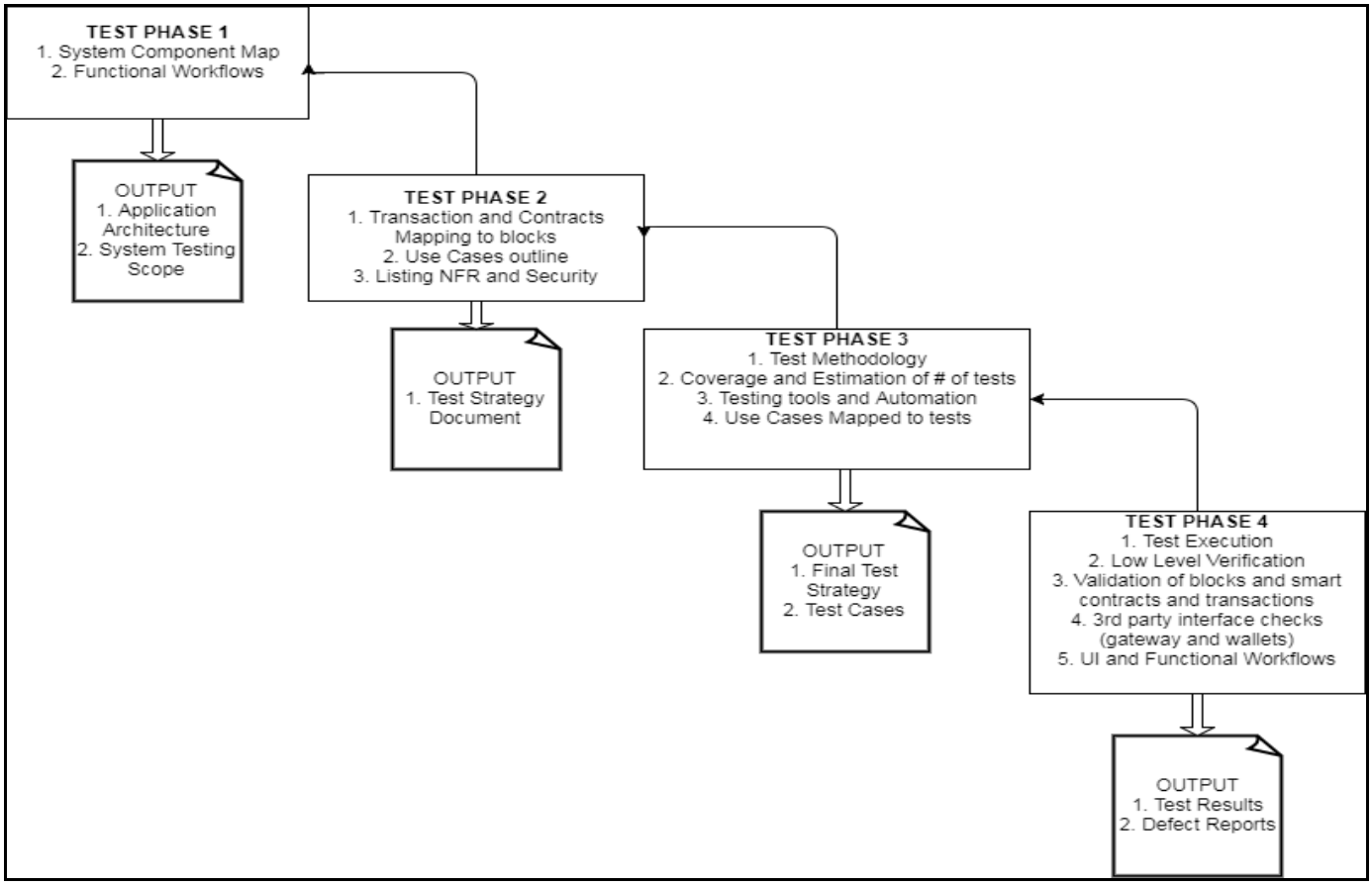

Fig 1: Proposed BOS Testing Lifecycle

4. CONCLUSION AND FUTURE WORK

In the present work, we focused on the most evident issues of state-of-art blockchain-oriented software development. In addition, we read multiple blogs and articles highlighting the issues present in Blockchain oriented software testing. On the basis of the results of the analysis, we proposed new directions for blockchain-oriented software engineering, focusing on collaboration among large teams, testing activities, and specialized tools for the creation of smart contracts. We also advocated the need for new professional roles by having a dedicated software tester from the start of testing lifecycle and also suggested enhanced security and reliability by testing key features of blockchain as well as overall system.

5. ACKNOWLEDGMENTS

We would like to express our sincere gratitude to our professor Dr. Ahmed Salem, for the continuous support of our study and research on this topic, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped us in all the time of research and writing of this paper.

6. REFERENCES

[1] Hiroki Watanabe, Shigeru Fujimura, et.al, "Blockchain Contract: A Complete Consensus using Blockchain", 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE).

[2] The blockchain challenge nobody is talking about: http://usblogs.pwc.com/emerging-technology/the-blockchain-challenge/

[3] Alex Norta et.al, "Conflict-Resolution Lifecycles for Governed Decentralized Autonomous Organization Collaboration", St.Petersburg, Russian Federation © 2015 ACM.

[4] Simone Porru et.al, 2017. Blockchain-oriented Software Engineering: Challenges and New Directions", 39th IEEE international conference on Software Engineering Companion

[5] Ethereum: https://blockgeeks.com/guides/what-is-ethereum/

[6] Testing of Blockchain: http://www.bcs.org/content/conWebDoc/56020

[7] Infosys Whitepaper: https://www.infosys.com/IT-services/validation-solutions/white-apers/Documents/blockchain-implementations-quality-validation.pdf

[8] Kombe, Cleverence & Manyilizu, Majuto & Mvuma, A. (2017). Design of Land Administration and Title Registration Model Based on Blockchain Technology. Journal of Information Engineering and Applications. 7. 8-15.

[9] S. Nakamoto, "Bitocoin: A Peer-toPeer Electronic Cash System," https://bitcoin.org/bitcoin.pdf, 2008.

[10] J. Bonneau et al, "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," in 36th IEEE Symposium on Security and Privacy, May 18-20, 2015.

[11] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating user privacy in bitcoin. In International Conference on Financial Cryptography and Data Security. Springer, 34–51.

[12] Giuseppe Ateniese, Antonio Faonio, Bernardo Magri, and Breno De Medeiros. 2014. Certified bitcoins. In International Conference on Applied Cryptography and Network Security. Springer, 80–96.

[13] Marcella Atzori. 2015. Blockchain technology and decentralized governance: Is the state still necessary? (2015).

[14] Tobias Bamert, Christian Decker, Roger Wattenhofer, and Samuel Welten. 2014. Bluewallet: The secure bitcoin wallet. In International Workshop on Security and Trust Management. Springer, 65–80.

[15] Roman Beck, Jacob Stenum Czepluch, Nikolaj Lollike, and Simon Malone. 2016. Blockchain-the Gateway to Trust-Free Cryptographic Transactions..In ECIS. ResearchPaper153.

[16] Joppe W Bos, J Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. 2014. Elliptic curve cryptography in practice. In International Conference on Financial Cryptography and Data Security. Springer, 157–175.

[17] Christian Decker and Roger Wattenhofer. 2014. Bitcoin transaction malleability and MtGox. In European Symposium on Research in Computer Security. Springer, 313–326.