Proceedings of the Second International Conference on Smart Electronics and Communication (ICOSEC).
IEEE Xplore Part Number: CFP21V90-ART; ISBN: 978-1-6654-3368-6

A Literature Survey on Smart Contract Testing and Analysis for Smart Contract Based Blockchain Application Development

R Sujeetha
Computer Science and Engineering SRMIST,Vadapalani
Chennai, India
sr7092@srmist.edu.in

C.A.S Deiva Preetha
Computer Science and Engineering SRMIST,Vadapalani
Chennai, India
deivaprc@srmist.edu.in

2021 2nd International Conference on Smart Electronics and Communication (ICOSEC) | 978-1-6654-3368-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICOSEC51865.2021.9591750

***Abstract*— Smart Contracts have noninheritable huge prominence within the recent years. After the Ethereum came into existence in 2015, the execution of smart contract had a great development in blockchain technology. The smart contracts execution faces number of issues to name few reliability, scalability, security. Development of Smart Contracts are not followed by standard software development life cycle. This causes the applications with smart contract(s) cannot perform exhaustive testing as well as it is expensive. The most prominent fields like smart contract testing and analysis of the code for vulnerability attracts many researchers. Targeted on reviewing the techniques and approaches discussed in various selected related papers from IEEE, science direct etc. For each of the selected research work identified open challenges that require further research. Hence this literature review on smart contract testing and analysis of smart contract code intends to emphasize the merits and demerits in smart contracts development process.**

Keywords*— *Smart Contract, Testing, Existing Tools

I. INTRODUCTION

Blockchain technology is the new technology of the future online. Blockchain technology is initially developed in the financial sector. Use Cases of blockchain technology enable other domains such as Health, Insurance, Supply chain Management, Education and many more to use technology for their better product. The first live blockchain was Bitcoin Blockchain [1] in 2009. Presenting the idea, frameworks used to get an arrangement between a deceitful group. Bitcoin exchanges are dependent upon the effective end of a total, non-intrusive framework that ensures things like proprietorship and accessibility of cryptographic money. One of the essential components of the Ethereum is smart contracts. Smart contracts are turning out to be progressively mainstream nowadays. It was first presented in early 1990s by computer researcher and cryptographer Nick Szabo as a sort of computerized transaction protocols. Clarifies by what means clients can enter information or worth and discover something restricted to the machine (example- soft drinks vending machine). To make it clear, smart contracts are usually user programs or algorithms or protocols that can be used to verify, validate or make irreversible transactions. This is a clear paradigm shift, dishonest organizations want to be trusted in their agreement to make the right computer system. A well-executed smart contract makes it possible to make crowdfunding without the need for a third party. Third-party involvement puts the system centralized where all the trusts are in one group. As smart contracts deployed in a blockchain are unchanged, public and empowered. Disrupting smart contracts is almost impossible. The largest blockchain Ethereum is currently supporting smart contracts specially designed in 2014 [2] as an extended model of blockchain. Solidity is one of the programming language used to develop smart contracts specifically and popularly in Ethereum. Over the most recent couple of years across the line, unsafe programming wrongdoing, which has brought about gigantic monetary misfortunes and social deterioration, has made an issue for appropriate development, approval and the execution of smart contracts.

II. BACKGROUND

A. *Importance of Testing*

The objective of programming advancement is to expand an item that will carry worth to our clients. As a computer programmer, we should have gained notoriety for uncovering and accepting advanced innovations and programming dialects. This will bring success in guaranteeing the product quality and provides worth to clients. Software development life cycle(SDLC) provides a standard procedure and helps to develop quality software which comprises testing as a fundamental part[5]. Testing plays a major role in the delivery of quality product to clients. Generally, the thought and procedure initially were done at the end of the coding of the product or before the delivery of the product. A promising beginning of the test, assist with diminishing the number of imperfections and, at last, the expense of revamping toward the end. The standards of software testing [6] provides various guidelines, in which guideline 3 emphasis on early testing which states as follows, "Guideline 3: Early testing: To discover early, testing exercises will be begun as ahead of schedule as conceivable in the product or framework advancement life cycle, and will be center around characterized targets. In the event that the testing group is included right from the start of the requirements and examination stage they have better agreement and knowledge into the item and besides the expense of value will be considerably less if the deformities are found as ahead of schedule as conceivable as opposed to later in the development life cycle".

As noted in "Inspecting Requirements" of Wiegers [7], "Industry data suggests that approximately 50% of the product defects are originated in the requirements elicitation". 80% of the revamp of coding or design happens with prerequisites faults.

"Programming Testing Techniques" Beizer [8], provides a complete index of testing procedures. The author expressed that "the act of designing tests is one of the most effective bug preventers known", which gives the meaning that mistake discovery is testing. This prompted an exemplary knowledge into the force of early testing. The smart contract testing needs effectiveness in designing test, such methods would permit the assessment and approval of the Smart Contract before it's deployed.

B. *Testing types and techniques*

Testing expects to assess distinctive programming perspectives to establish that the product meets its plan and necessities according to the particulars given by the client. The product testing measure follows a bunch of exercises, executing the product code and assesses the different properties of the product a work in progress with the plan to discover the mistakes. It can just distinguish the errors that is available and can't recognize the shortfall of any highlights. This happens in light of no thorough testing is performed because of time limitation. There are four degrees of testing occurring in SDLC, perceived as levels: unit testing, integration testing, system testing and acceptance testing as given in figure 1.

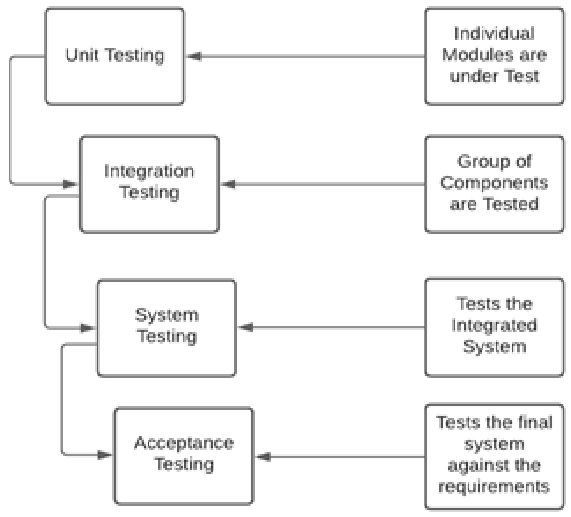
Unit Testing
Individual Modules are under Test
Integration Testing
Group of Components are Tested
System Testing
Tests the Integrated System
Acceptance Testing
Tests the final system against the requirements

Figure1: Levels of Testing

- The unit testing guarantees that singular programming code modules are functioning effectively.
- The objective of integration testing is to assess if the units are effectively functioning while incorporated.
- System testing confirms that end to end system's predefined prerequisites is evaluated.
- At last, acceptance testing is performed to ensure that the system complies with the business prerequisites and meets the criteria for delivery to end-users.

The intricacy and testing costs increments as progress from the bottom level of testing i.e. "unit testing" to the top level of testing i.e. "acceptance testing". This is on the grounds that unit tests are effectively robotized contrasted with different kinds of tests.

Furthermore, there are various types of software testing, which can be as simple as static or dynamic given in figure 2. Testing that doesn't include programming execution yet plays out the checking of source code construction, punctuation and information stream is named as static investigation. Commonplace ways to deal with static investigation incorporate examinations, audits, and step by step analysis. For demonstrating the accuracy of programming, static investigation can be used in addition to procedural confirmation techniques. Dynamic testing, then again, includes programming execution, while taking care of data sources and creating yields.

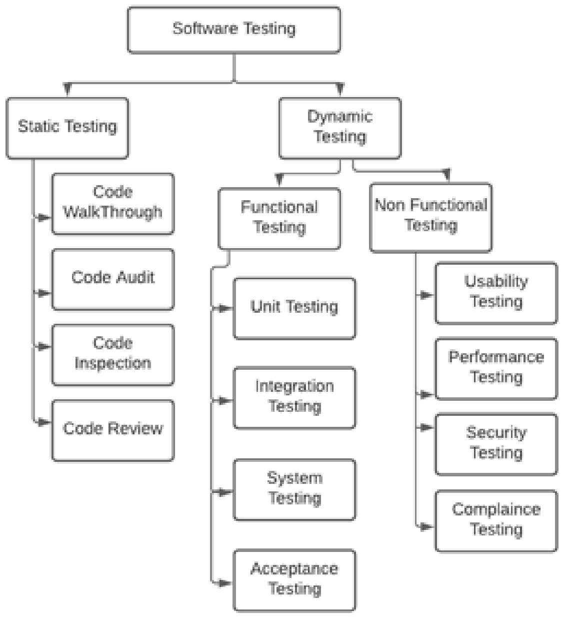

Figure2: Software Testing Types

Experiments are commonly evolved indicating test sources of info and yields, and the objective of testing is to check whether the real yields adjust to the normal yields. Dynamic testing methods can be additionally delegated functional and non-functional testing.

Functional testing procedures are performed to validate the functional specifications of the product. Testing can be additionally named discovery testing. Explicit white-box testing methods incorporate interface testing and mutation testing.

Non-functional testing is a group of testing types performed against non-functional requirements like usability, performance, security and compliance. Evaluates whether a component of the system complies with the requirements. Non-functional testing design tests the system availability to usage by clients meeting the nonfunctional parameters which are not generally addressed by functional testing methods.

C. *Research Motivation:*

Testing is a significant aspect of software development that guarantees quality and viability. To quantify the adequacy, an estimation of test suites is required. The quality test suite includes tests that distinguish and discover flaws. If experiments cannot identify flaws, it will be difficult to build good software. The quality of the test suite or testing can be improved or estimated by using certain fault-based testing techniques like code coverage, error seeding and mutation analysis. Mamdouh Alenezi et. al., [3] suggests "When comparing the line code coverage and mutation testing, line code coverage does not provide the quality test suites whereas mutation testing provides a better picture of coverage by injecting faults into the source and comparing it with the original code". Also suggested that "both methods can be used in conjunction which can provide a comparative and actual picture of where the test suites are lacking and for that area a new test case can be developed". This motivated the research to be carried out on mutation analysis and other testing techniques for the smart contract can help to provide the guidelines in producing effective test suites used for smart contract testing which can help in deploying defect free smart contract.

D. Structure of the paper:

The goal of this review is to highlight various tests approaches available for smart contract and tools for smart contract analysis. The following have been adapted to post the open challenges which can direct us for future research to be carried out. Explored a number of literature review papers, related works that had been issued in reputable conferences and journals on testing techniques used for smart contracts, how mutation testing is applied for testing smart contracts and smart contract code analysis. The following is how the paper is structured.

- Section III provides the various tests approaches used for smart contract.
- Section IV Existing mutation testing tools.
- Section V Challenges in testing smart contract for blockchain-based applications.
- Section VI Tools used for the analysis of smart contract code as well as the current challenges are discussed.
- Section VII concludes the paper.

III. SMART CONTRACT TESTING

Wang et al., [10] proposed random and NSGA-II based multi-objective approach to provide cost-effective test suites. Many of the researchers proposed a number of testing and verifying methods for a smart contract where they have not much concentrated on the cost to perform a transaction, duration cost (the time taken to transact) and coverage of uncovered code branches. Wang et al., [10] encountered those as their objectives to achieve the cost-effective generation of the test suite for smart contract and achieved the same by implementing above mentioned multi-objective approaches. The author(s) have also mentioned as "maximizing mutation killed ability during the test-suite generation are future work".

The importance of testing is not only code coverage, but also to detect defects or bugs. Chen et al [11] due to the immutable nature of smart contracts, it should be defect-free. The author had analyzed various real-world smart contracts and posts related to smart contracts and defined 20 kinds of defects that are malicious and getting rid of them improves the goodness of smart contracts. The impacts caused by contract defects are summarized and thus help developers to better understand the indicator of faults and removing them.

Zhang et al. [12] addressed generation of effective test cases based on data flow analysis from the constructed Control flow graph. Proposed method includes genetic algorithm for optimizing the dynamic test of smart contracts. The test cases are generated in stages. Firstly constructed the control flow graph for the source code, next the graph is analyzed to obtain the required statements, to get the definition-use pairs for the Solidity programs. Finally, to generate the test case, a genetic algorithm with a function was used to calculate the definition-use pairs.

Some studies proposed smart contract testing methods using fuzzing approaches to identify faults or security flaws. Tools of kind EVMFuzz [13], ContractFuzzer [14] and Fuse [15] was introduced. EVMFuzz [13] targets recognizing weaknesses of Ethereum Virtual Machines (EVM) using differential fuzz testing. The central idea is to generate smart contracts on a continuous basis and feed them to EVM in order to find as many unpredictable in results as possible, which helps in discovering susceptible with output cross-referencing. The tool produces smart contracts with predefined mutators before applying dynamic priority scheduling. The tool discovered several previously unknown security bugs.

Contract Fuzzer [14] is a tool for testing smart contracts for reliability. During execution, fuzzing input is created on basis of Application Binary Interface specifics of smart contracts and afterwards, they are characterized to identify flaws and are logged. Of about 6991Smart Contracts, the tool announced in excess of 459 flaws with high precision, particularly those of "The DAO" bug and the Parity Wallet bug that prompted millions of dollars in misfortunes.

Anna Vacca [22] published a detailed study on various techniques, tools and challenges for blockchain smart contract-based development. The review study was carried out on six major categories like testing smart contracts, analysis of smart contract code, metrics, and security for smart contract, Decentralized applications performance, and blockchain applications. Most of the researchers used Ethereum technology on test generation for smart contracts. The authors had suggested some open challenges in each of the six categories they have considered. The result of their literature review on the smart contract and blockchain applications reveals that there is a need for a blockchain-based software development life cycle.

IV. EXISTING MUTATION TESTING TOOLS

The traditional mutation testing when compared with mutation testing for smart contracts the methodology remains same where the mutants are selected and injected into the source code. The mutated source code is then given as input program to perform the mutation testing using the test suite. The mutation score is then computed to evaluate the test suite's quality and the effectiveness of the test case. Figure 3 prescribes the process of conventional mutation testing. This section provides an overview of the existing mutation testing tools for the smart contracts.

Zixin Li [4] MuSC was the introductory mutation testing tool in the Ethereum platform in solidity language for smart contract based blockchain applications. The author preferred mutation testing to assess the testing's adequacy. The paper discusses in detail the workflow of the mutation testing tool. The smart contract source code written in solidity is first converted into an abstract syntax tree format. The input to the tool is the smart contracts and the test cases. The users select the mutation operators to be used in the tool. Once the operators are selected, the source code is transformed into AST format. The mutants are introduced into the AST formatted source code. During the execution of mutants, the tool records each variant's information as well as the corresponding results of execution. Mutants causing compilation errors are not considered in testing. The mutation score and test results for every mutant are generated as resultant test report. The tool is effective in error injection.

"When comparing test results related to the quality of the test suite, the efficiency of mutant generation and execution should be prioritized" as suggested by the author.

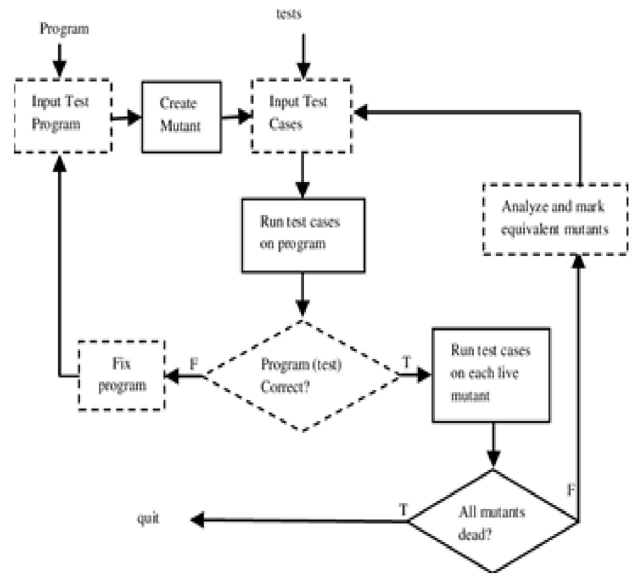

Figure 3: Conventional Mutation Testing

Y.Ivanova A.Khritankov [25], developed regular mutator that improves the reliability of the smart contract. The author considered mutation analysis to enhance the test suites quality. They have developed the tool with the mutation operators that are very specific to the solidity language. Finally, the developed methodology is compared with existing methods like code coverage metric in terms of error detection. The common errors occurring in the coding activity of a product are identified to prepare possible mutation operators. The selected mutation operators are represented as regular expressions which are substituted in the source code as mutants. For example, the arithmetic operators are called "arithmetic operator replacement" where the arithmetic operators are modified like "+" as "-" are injected as mutants. Similarly, relational operators, conditional operators, solidity specific operators are injected as mutants. By using the identified mutation operators the tool is developed in python language and is tested on large projects containing smart contracts and test suites in the blockchain. Finally, the code coverage metric and mutation score are compared. On careful examination of the output, one of the following status is allocated for the mutant:

1. "SURVIVED", on successful execution of all test cases,
2. "KILLED", on finding a mistake and failed,
3. "COMPILATION ERROR", on the non-successful compilation.

The quantitative relation between the total killed mutants to the total number of generated mutants excluding compilation error status gives the mutation score, which is compared with code coverage metrics for assessment of the tool. The author suggested the following as future directions,

1. "To replace the method of inserting mutations using regular expressions with building an abstract syntax tree and making mutations that will not cause compilation errors.
2. Selection of mutants in order to use the most effective ones."

Hartel P, Schumi R [26] proposed a set of mutation operators and evaluated these operators at scale and mutant killing condition-based gas limits for smart contract and achieved novelty in this killing condition. The proposed concepts are accomplished by picking the most precise smart contract-specific also called as solidity language-specific mutation operators, examines its viability with mutant killability and feature extreme vulnerabilities that can be infused with the transformations. The existing mutation methods are enhanced by the addition of a killing condition capable of detecting a change in gas consumption. The authors have suggested the following as future directions in which they have mentioned as "there is a need to study the errors made by Solidity developers at scale to validate the mutation operators and to use the gas limit on transactions to detect equivalent mutants".

Honig J.J, Everts M.H, Huisman M [27] proposes a mutation testing framework called vertigo. As discussed in above papers related to mutation testing here also the author considers mutation testing rather than code coverage metric as relating to general strategies for the nature of the test suite, it estimates the suite's viability at distinguishing deficiencies in the source code straightforwardly. It accomplishes this by presenting small changes in the source code such as from ">" to "=", and then running the test suite for each of such changes. Once all the injected mutations are tested, the mutation score is calculated used as the metric to assess the test suite quality. The author also discusses the use of mutation testing in the development lifecycle of a smart contract. They provided a definite plan and execution of a mutation testing structure that is independent of its test environment in this paper. Also proposes new mutation operators for the Solidity language, which is used in smart contract development which was not considered by other researchers in previous works related to mutation testing for smart contract.

In vertigo, mutant sampling proposed by [9] is used to choose a random sample from accessible transformations executed as an early sifting. This sifting can be utilized to execute others like mutant grouping or equivalent mutant detection. Vertigo executes the whole of test suite for every mutant in a random or pre-determined order, to name a few flaws. The author suggests that research to be conducted on assessing available mutation operators and design new operators specific to solidity. There is a need for optimization technique in mutant's assessment in terms of mutation score. The outcome suggests that mutation testing can be used to enhance contracts for the language specific like solidity. Also there was a less count of equivalent mutants produced proves the efficiency of the operators been considered and reduces the manual process of investigating equivalents. The author suggests, "Including the characteristics of arithmetic mutation operators to the safe math function" as future work. These available existing tools helps to understand the need for using the mutation testing helps in improving the smart contract effectively and to develop efficient test suites to avoid defects.

The following table provides a summary of merits and remarks of research works considered in this literature survey.

Paper Title &Year	Authors	Merits	Remarks
"Towards generating cost-effective test-suite for Ethereum smart contract", 2019	Wang, X., Wu, H., Sun, W., Zhao, Y	Provides cost efficient test cases.	Requires maximizing ability of killed mutants during the test-suite generation.
"Defining smart contract defects on Ethereum", 2020	Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., Chen, T	Improved robustness of smart contracts.	The defects dataset can be used in developing the smart contract defect detection tools.
ADF-GA: Data flow criterion based test case generation for Ethereum smart contracts, 2020	Zhang, P., Yu, J., Ji, S	Proposed ADF-GA for Solidity based Ethereum smart contract programs that generate test cases by using All-Uses Data Flow and genetic algorithm, achieves more coverage and has less number of replay in genetic algorithm.	Requires further optimization in selecting and operating of mutants for generating test cases. Needs to investigate better suitable methods for evaluation of test cases. There is restrictions in executing environment of smart contracts for performing dynamic testing which requires a solution.
A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges, 2021	Anna Vacca, Andrea Di Sorbo, Corrado A Visaggio , Gerardo Canfora	Review was carried out on six major categories like testing smart contracts, analysis of smart contract code, metrics, security for smart contract, Dapp performance, and blockchain applications. Authors had suggested some open challenges in each of the six categories	There is a need for a blockchain-based software development life cycle. There is a need to develop specialized tools for specific testing like mutation testing which helps to provide a quality smart contract
MuSC: A Tool for Mutation Testing of Ethereum Smart Contract,2019	Zixin Li, Haoran Wu, Jiehui Xu, Xingya Wang, Lingming Zhang, Zhenyu Chen	The author prefered mutation testing to assess the testing's adequacy and is the first mutation testing tool for Ethereum smart contract. The tool is effective in error injection.	Mutants causing compilation errors are not considered in testing. Some features like error handling has to be improved. The efficiency of mutant generation and execution should be prioritized.
Regular Mutator: A Mutation Testing Tool for Solidity Smart Contracts,2020	Y.Ivanova A.Khritankov	Improves the reliability of the smart contract. It involves automatic adjustments to the programme syntax and the production of semantic versions of the programme as an alternative to line metrics.	Needs developing alternate method for mutation insertion such that they does not cause compilation errors.
Practical Mutation Testing for Smart Contracts, 2019	Honig J.J., Everts M.H., Huisman M	Proposes a mutation testing framework called vertigo. The author shows that mutation testing can be used rather than code coverage metric as relating to general strategies for the nature of the test suite, it estimates the suite's viability at distinguishing deficiencies in the source code straightforwardly.	Vertigo executes the entire test suite for each mutant and executes the test suite in a random or pre-determined order, to name a few flaws. The author suggests that research be conducted on how to assess existing mutation operators and design new mutation operators for smart contract specifics. There is a need for optimization technique in mutant assessment in terms of mutation score.
Mutation Testing of Smart Contracts at Scale, 2020	Hartel P., Schumi R	Mutation methods are enhanced by the addition of a killing condition capable of detecting a change in gas consumption.	There is a need to study the errors made by Solidity developers at scale to validate the mutation operators and to use the gas limit on transactions to detect equivalent mutants.
Blockchain enabled smart contract based applications: Deficiencies with the software development life cycle models, 2020,	M. H. Miraz and M. Ali	Explored about six traditional SDLC models analyzed them in comparison with the issues faced in software development using blockchain smart contract technology.	Advocates there is an immediate need to develop a new standard model(s). Traditional SDLC models are not suitable for Block chain based development.
Testing, verification and validation of distributed ledger systems,2020	K. N. Pankov	Identified the inadequacy of best practices in this domain by listing available technologies for evaluation and assessing blockchain enabled smart contracts.	The process of establishing a regulatory and technical foundation does not yet include testing, verification, and validation of distributed ledger systems. Conducting research in this area and developing the appropriate standards would help to accelerate the development of blockchain-based distributed ledger technology while also increasing its security.

Table:1

V. Challenges in Testing Smart Contract for Blockchain-Based Applications

This part presents the different provokes intrinsic to testing the Smart Contract executions. The distinguished key difficulties are as per the following,

A. Lack of best practices in creating Smart Contract:

The absence of innovation understanding, and the absence of abilities or involvement with planning and creating Smart Contract are incorporated. Additionally, the absence of normalization in the use of ideas and phrasings prompts a decline in lucidity, quality, and efficiency. To help the implementation of smart contract and upgrade the product quality, explicitly accepted procedures between the framework and the local area running after open source in creating test-suites would be profoundly gainful. Relatively, BlockChain is another innovation, and to now make it available as an application (e.g., medical care or production network), one must have a thorough understanding of its concepts and potential. In spite of the fact that it was presented in “2008 by Satoshi Nakamoto”[1] as a significant hidden innovation in the “Bitcoin framework”, because of the absence of the previously mentioned best practices, it is yet far from possible selection.

Also, the specialized and non-specialized ability is basic for viable and comprehensive testing. With the consciousness of this test, tending to its anything but a simple undertaking since getting the necessary extra abilities e.g., knowledge of latest smart contract developing languages or grasp the prescribed procedures in executing blockchain smart contract are costly. Besides, even with the effective execution of the smart contract, their testing is exceptionally specific that requires experience and a thorough testing approach.

B. Lack of specialized tools

The testing tools look forward to carrying out the system being tested which reflects the execution with good constancy thus guarantee the effectiveness of testing being performed. Here the system under test is the blockchain application that requires colossal exertion and there is also a lack of automation support. One of the most important decisions is choosing the right tool for the application.

To address this issue, there are few tools with non-proprietary software executions that produce experiments that don't generally mirror the experiments of a real situation, yet they can still be successful in testing some of the high-level exchange functionalities. Specifically, it needs an incorporated improvement environment that offers the required modules, compiler, framework to test, tools for investigating and recording. Few variants of these tools exist, they are although there are a small number of variations of these tools, and they are insufficient to meet the needs of engineers.

Anna Vacca [22], Examining how conventional testing procedures similar to the unit and integration testing, mutation testing, test case generation, and regression testing, should be adjusted to the improved interaction and construction of a smart contract. There is a need for the development of testing tools that are generally used for smart contracts that helps to detect vulnerabilities as well as bugs in source code. There is a need to develop specialized tools for specific testing like mutation testing which helps to provide a quality smart contract.

VI. Tools Used for the Analysis of Smart Contract Code and Current Challenges

The following section discusses various tools available for smart contract analysis for vulnerability detection. Mutation testing of the smart contract not only helps in assessing the test case suite also helps to improve vulnerability detection. Smart Contract analysis concentrates on both static analyses as well as dynamic analysis of code.

A. Smart Contract Analysis Tools:

SmartCheck is a static evaluation tool for smart contract Tikhomirov [17]. SmartCheck executes lexical and syntactical analysis on Solidity smart contract code. It generates intermediate representation as XML parse tree by using ANother Tool Language Recognition and a custom Solidity grammar. It detected suspicious samples by using XPath queries on the intermediate representation. The device attempted on various checked smart contracts referenced from Etherscan. SmartCheck can recognize security, usefulness, operational and improvement issues. “Examination shows that 99.9% of agreements have issues, 63.2% of agreements have basic weaknesses” provided by the author.

Securify, security analyzer tool by Tsankov et al.,[16] is versatile, completely computerized, and set to demonstrate contract practices being protected/unprotected regarding a given property. Securify's examination comprises of two stages. To start with, it symbolically examines the contract's reliance chart to separate exact semantic data from the code. At that point, it checks consistency and infringement designs that catch adequate constraints to demonstrate whether or not a property holds. To empower expandability, samples are indicated in an assigned domain-specific dialect. The tool had investigated more than eighteen thousand smart contracts created by the clients, systematically it is been adopted by the specialists to perform vulnerability reviews.

VERX was introduced to confirm the useful features of smart contracts by Permenev et al[18]. The tool is designed based on the blend of three steps like decrease of time property confirmation, symbolic execution and the postponed predicate that utilizes representative running during exchanges and deliberation at the boundaries of exchanges. A trial assessment led on 83 transient properties and 12 genuine ventures showed the adequacy of the device.

In other researches, the “Solidity Instrumentation Framework” was introduced for the improvement of smart contracts developed using Solidity [19]. The framework helps to easily query and modify syntactic information as well as can generate source code from the Abstract Syntax Tree, mirroring progressions created by the client. It also aims to screen, dissect, upgrade and produce code. The framework was evaluated on genuine smart contracts to verify with the declarations and fault-based testing by injecting faults. The assessment also shows that the framework is prepared to do consequently examining the intermediate representation and addressing it through the use of object-oriented classes like c++, provides a graphical user interface for data recovery and modifies the intermediate representation and generates solidity code from it.

Di Angelo and Salzer [20] has reviewed large number of publications on available tools for the investigation of Ethereum smart contracts on thinking about accessibility, development, techniques utilized and the level of safety. Finally, a short list of tools that can be considered useful for detecting security issues was provided. According to the author, FSolidM takes an interesting approach by constructing source code from a state-oriented requirement. Due to its intellect, KEVM seems to be the consensus pick for verification, with the downside that its use necessitates competence. Securify is the utmost sophisticated tool for validation. MAIAN is based on a specific description of the security flaws to be discovered as well as, in comparison to several other tools, goes above and beyond to eliminate random errors by checking the vulnerabilities. To end up the list, Mythril is a perfect illustration of a tool for collaborative contract analysis with a focus on usefulness.

Zou et al. [21] Provided an overview to emphasize the actual issues that exist in the advancement of smart contracts. Based on the results achieved, some of the issues faced by the developers while developing smart contract like, there is no fruitful method to ensure the vulnerability issues available in smart contract code, development tools are yet fundamental, coding dialects and virtual machines further have a lot of weaknesses, functional issues are complex to tackle in a resource limited execution, and information resources are not much available.

B. Challenges in Smart Contract Analysis

Here we specify few of the identified challenges in smart analysis to further direct our research towards the improved analysis tools for smart contract. To recognize costly pattern and enhancement mechanisms for smart contracts; develop smart contracts explicit measurements for assessing code properties identified with quality; produce code examination techniques that are autonomous from the programming language or that could cover different languages other than Solidity and Go.

Nicolas et,al.,[28] have suggested that the traditional SDLC is not suitable for smart contract based blockchain development process. The distinct approach for assessing and affirming the excellence of a whole system development is not available. Pankov [24] identified the inadequacy of best practices in this domain by listing available technologies for evaluation and assessing blockchain enabled smart contracts. Miraz, Ali [23] explored about six traditional SDLC models analyzed them in comparison with the issues faced in software development using blockchain smart contract technology and advocates an immediate need to propose a new model that best suites the development process of smart contract. These are still open for future research.

VII. CONCLUSION

In the developing expansion of blockchain based innovations and the utilization of smart contracts in various areas, it is fundamental to possess an image of present technology status as well as dissect whereupon areas such as innovation carries advancement. Nonetheless, this innovation can be improved by presenting software rehearses for testing just as advancement of smart contracts. In view of these inspirations, we conducted the survey on two fields like smart contract testing and analysis of its code in order to comprehend the procedures, practices, and tools explicitly considered in dealing with the issues presented in smart contract development, as well as recognizes that the issues are still open. Furthermore extensive research on how to apply conventional testing methods to blockchain technology based development is required as well as engineers should be provided with set of rules and procedures to avoid confusions in process used for development. There is also a need to improve languages used for coding smart contract in order to work on the creation and comprehension of smart contracts. Finally, specialized tools and frameworks are required for testing the smart contract which is independent of the language and platform used for developing the smart contract.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] “Ethereum foundation. The solidity contract-oriented language”. https://github.com/ethereum/solidity., 2014.

[3] M. Alenezi, M. Akour, A. Hussien and M. Z. Al-Saad, "Test suite effectiveness: an indicator for open source software quality," 2016 2nd International Conference on Open Source Software Computing (OSSCOM), 2016, pp. 1-5, doi: 10.1109/OSSCOM.2016.7863677

[4] Z. Li, H. Wu, J. Xu, X. Wang, L. Zhang and Z. Chen, "MuSC: A Tool for Mutation Testing of Ethereum Smart Contract," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1198-1201, doi: 10.1109/ASE.2019.00136

[5] N. Sánchez-Gómez , L. Morales-Trujillo , J. Torres-Valderrama ,“Towards an Approach for Applying Early Testing to Smart Contracts” , 2019, - 4th International Special Session on Advanced practices in Model-Driven Web Engineering, Science Direct.

[6] Graham D., Van Veenendaal E., Evans I., Black R., 2015. Foundations of Software Testing: ISTQB Certification Cengage Learning Emea; Revised edition.

[7] Wiegers K.E., 2001. Inspecting Requirements. StickyMinds.com Weekly Column.

[8] Beizer B., 1990. Software Testing Techniques. Van Nostrand Reinhold Company Limited.

[9] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," in IEEE Transactions on Software Engineering, vol. 37, no. 5, pp. 649-678, Sept.-Oct. 2011, doi: 10.1109/TSE.2010.62.

[10] Wang, X., Wu, H., Sun, W., Zhao, Y., 2019. Towards generating cost-effective test-suite for Ethereum smart contract. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 549–553.

[11] Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., Chen, T., 2020b. Defining smart contract defects on Ethereum. IEEE Trans. Softw. Eng.

[12] Zhang, P., Yu, J., Ji, S., 2020. ADF-GA: Data flow criterion based test case generation for Ethereum smart contracts. In ICSEW'20: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp 754–761.

[13] Fu, Y., Ren, M., Ma, F., Jiang, Y., Shi, H., Sun, J., 2019. EVMFuzz: Differential fuzz testing of Ethereum virtual machine. arXiv preprint arXiv:1903.08483.

[14] Jiang, B., Liu, Y., Chan, W., 2018. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 259–269.

[15] Chan, W., Jiang, B., 2018. Fuse: An architecture for smart contract fuzz testing service. In:2018 25th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 707–708.

[16] Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.,2018. Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 67–82.

[17] Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E.,Alexandrov, Y., 2018. Smartcheck: Static analysis of Ethereum smart contracts.In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 9–16.

[18] Permenev, A., Dimitrov, D., Tsankov, P., Drachsler-Cohen, D., Vechev, M., 2020. VerX: Safety verification of smart contracts. In: 2020 IEEE Symposium on Security and Privacy. SP, pp. 18–20.

[19] Peng, C., Akca, S., Rajan, A., 2019. SIF: A framework for solidity contract instrumentation and analysis. In: 2019 26th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp.466–473.

[20] Di Angelo, M., Salzer, G., 2019. A survey of tools for analyzing Ethereum smart contracts. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures. DAPPCON, IEEE, pp. 69–78.

[21] Zou, W., Lo, D., Kochhar, P.S., Le, X.-B.D., Xia, X., Feng, Y., Chen, Z., Xu, B., 2019. Smart contract development: Challenges and opportunities. IEEE Trans. Software Engineering.

[22] Anna Vacca, Andrea Di Sorbo, Corrado A Visaggio , Gerardo Canfora, 2021. A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. In: 2021 The Journal of Systems & Software Volume 174, https://doi.org/10.1016/j.jss.2020.110891.

[23] M. H. Miraz and M. Ali, "Blockchain enabled smart contract based applications: Deficiencies with the software development life cycle models," 2020, arXiv:2001.10589. [Online]. Available: http://arxiv.org/abs/2001.10589

[24] K. N. Pankov, "Testing, verification and validation of distributed ledger systems," in Proc. Syst. Signals Generating Process. Field Board Commun., Mar. 2020, pp. 1 to 9.

[25] Y.Ivanova A.Khritankov, "Regular Mutator: A Mutation Testing Tool for Solidity Smart Contracts", 2020, Science Direct, Procedia Computer Science, Volume 178, Page No. 75-83.

[26] Hartel P., Schumi R., "Mutation Testing of Smart Contracts at Scale". (2020) In: Ahrendt W., Wehrheim H. (eds) Tests and Proofs. TAP 2020. Lecture Notes in Computer Science, volume 12165. Springer, Cham. https://doi.org/10.1007/978-3-030-50995-8_2.

[27] Honig J.J., Everts M.H., Huisman M. "Practical Mutation Testing for Smart Contracts",2019. In: Pérez-Solà C., Navarro-Arribas G., Biryukov A., Garcia-Alfaro J. (eds) Data Privacy Management, Cryptocurrencies and Blockchain Technology. DPM 2019, CBT 2019. Lecture Notes in Computer Science, vol 11737. Springer, Cham. https://doi.org/10.1007/978-3-030-31500-9_19.

[28] N. Sánchez-Gómez, J. Torres-Valderrama, J. A. García-García, J. J. Gutiérrez and M. J. Escalona, "Model-Based Software Design and Testing in Blockchain Smart Contracts: A Systematic Literature Review," in IEEE Access, vol. 8, pp. 164556-164569, 2020, doi: 10.1109/ACCESS.2020.3021502