Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online *Vol 7 Issue 2, July 2025*
(https://fcihib.journals.ekb.eg)

Blockchain and Smart Contract Development: A Survey of Testing Challenges and Approaches

Ramadan Nasr[1,*], Mohamed Ibrahim Marie[1], Ahmed El Sayed Yakoub[1]
[1]Information System , Computers and Artificial Intelligence , Helwan , Cairo ,Egypt
ramadannasr40@fci.helwan.edu.eg, mohamedmarie@yahoo.com, eng_ahmedyakoup@yahoo.com

***Abstract*—The proliferation of blockchain technology signifies a revolutionary shift in the management of online transactions, offering unparalleled security and efficiency through immutable, peer-to-peer decentralized networks. Despite its transformative potential, blockchain introduces significant complexities and testing challenges, particularly within the realm of smart contracts. This paper surveys the landscape of smart contract testing, outlining the critical challenges and the necessity for specialized testing tools and methodologies. Through a comprehensive review of existing literature and practical approaches, this study emphasizes the importance of rigorous testing protocols to ensure the reliability and security of blockchain applications. Key areas of focus include transaction cost optimization, defect identification, and the development of robust testing frameworks, all aimed at maintaining high-quality standards in this rapidly evolving technological domain.**

***Index Terms*— Blockchain, Smart Contract, Smart Contract Testing, DevOps.**

I. Introduction

Digital transactions are rapidly gaining traction globally due to enhanced security, ease of use, and speed. Blockchain technology provides a secure and trustworthy platform for managing online transactions without relying on intermediaries [1]. As more individuals shift towards digital transactions, security becomes a critical concern. Blockchain and smart contracts have attracted significant attention from various sectors, including financial institutions and regulatory bodies. The adoption of blockchain is expected to continue growing exponentially in the coming years. This technology enables secure and transparent transactions, fostering trust among users. Organizations are investing heavily in blockchain to capitalize on its transformative potential.

A blockchain functions as a decentralized ledger distributed across a public or private network of peers where information is shared [2], replicated, and synchronized. This ledger permanently records the history of asset exchanges among network members in a sequential manner. Each transaction is timestamped and assigned a unique cryptographic signature for security. Network nodes collectively agree on a consensus protocol to validate and update ledger records, removing the need for intermediary entities like financial institutions. Once data is stored on the blockchain, it remains immutable. Verified transactions are grouped into blocks and linked to previous blocks, forming a chain. Blockchain technology offers vast potential to revolutionize digital transaction management.

Blockchain technology offers a fresh perspective on software development, as decentralization and anonymity among network nodes create distinct challenges for testing [2]. Traditional testing methods may not be adequate in light of these complexities. Additionally, the unchangeable nature of the blockchain implies that errors in the production system could necessitate extensive code modifications. Consequently, it is essential to employ suitable testing techniques and methodologies. This paper seeks to delineate the hurdles and propose approaches for testing blockchain applications to uphold superior quality standards.

This paper provides a comprehensive survey of the current methodologies, tools, and challenges associated with the testing and analysis of smart contracts in blockchain-based applications. The unique features of blockchain, such as distributed consensus mechanisms and the unchangeable nature of the ledger, introduce complexities that traditional testing methods cannot adequately address. This study highlights the need for specialized testing tools and techniques to ensure high-quality software delivery in blockchain applications. We delve into various testing strategies, including static and dynamic testing, functional and non-functional testing, and mutation testing, specifically tailored for smart contracts. By examining existing tools and methodologies, such as EVMFuzz and Contract Fuzzer, and reviewing recent advancements in the field, our contribution underscores the importance of developing a blockchain-oriented software development lifecycle to meet the evolving demands of this transformative technology.

Table 1. Examples of most popular blockchains

Platform	Type	Smart Contract Language	Consensus Mechanism	Main Application Contexts	Related Projects
Ethereum	Public	Solidity, Vyper	Proof of Work	Financial, Asset trading	DAI Gitcoin Cryptokitties
Quorum	Private	Solidity, Vyper	Proof of Authority	Financial, Supply Chain and Logistics	Liink Komgo Project Ubin
Hyperledger Fabric	Public, Private	Java, Go, JavaScript	Proof of Work	Supply chain, Trade finance, Stock trading	IBM Food Trust Everledger diamond blockchain
Hyperledger Sawtooth	Private	Rust, Go Python, Java	PoET, PBFT, RAFT	Supply chain, Provenance Tracking	Sawtooth Private UTXO Sawtooth Marketplace
NEM	Private	Java	Proof of Importance	Augmented reality, Advertising and marketing, Banking	DigitCoin Bankera Pantos
Stellar	Public	Solidity, JavaScript, Java, Go	Stellar Consensus Protocol	Remittance	StellarX Tempo TillBilly
Corda	Private	DAML, Kotlin, Java	Validity and Uniqueness	Energy trading, Insurance, Retail markets	Energy Block Exchange TradeCloud MonetaGo

The subsequent sections of the paper are structured as follows: Section II provides background material on blockchain smart contracts as well as software testing. In Section III, the focus shifts to smart contract testing. Section IV outlines various challenges associated with smart contract testing. Section V provides approaches to testing, while Section VI offers concluding remarks. Finally, Section VII presents references.

II. BACKGROUND

A. Blockchains And Smart Contracts

Blockchains serve as the foundational infrastructure for digital currencies, featuring blocks secured against tampering or modification, with each block containing data referencing the previous one and a timestamp, ensuring data integrity [3]. Initially, first-generation blockchains like Bitcoin were primarily focused on facilitating cryptocurrency transactions, while second-generation blockchains like Ethereum expanded their scope to enable the development and execution of software by blockchain participants [4].

Blockchain networks are broadly categorized into public and private types: public, or permissionless, blockchains allow all participants to modify the ledger's state, prioritizing transparency but potentially compromising privacy. In contrast, private, or permissioned, blockchains restrict transactional participation to authorized entities, ensuring transactional privacy. In this section, we'll delve into the origins of blockchain and the development of smart contracts, along with highlighting the significance of software testing within the realm of smart contract development. Table 1 illustrates a comparison between the most commonly used blockchain Platforms [5].

The table above illustrates a comparison of the most popular blockchain platforms and their smart contracts, including their primary application context and associated projects.

Blockchain technology provides the distinctive ability to incorporate smart contracts into transactions, facilitating the execution of code in a decentralized and secure setting. Smart contracts, essentially code that executes automatically when certain conditions are met, utilize the permanence and resistance to censorship that are fundamental to blockchain systems.

These contracts allow for the automation of business operations across various organizational borders, ensuring transparency and reliability. Developers encode business rules into smart contracts to govern transaction outcomes, such as payment execution upon successful delivery or fund reversal in case of delays or discrepancies. Smart contracts operate autonomously, executing predefined actions without human intervention, thereby enhancing efficiency and trust in blockchain-based legal agreements.

The figure above outlines the functionality of a prototype smart contract. This contract is designed to automate rent payments, where the tenant initiates fund deposits into the contract. Subsequently, the landlord can withdraw these funds periodically, typically on a monthly basis. Additionally, the contract maintains a comprehensive record of all rent transactions, ensuring transparency and accountability for both parties involved.

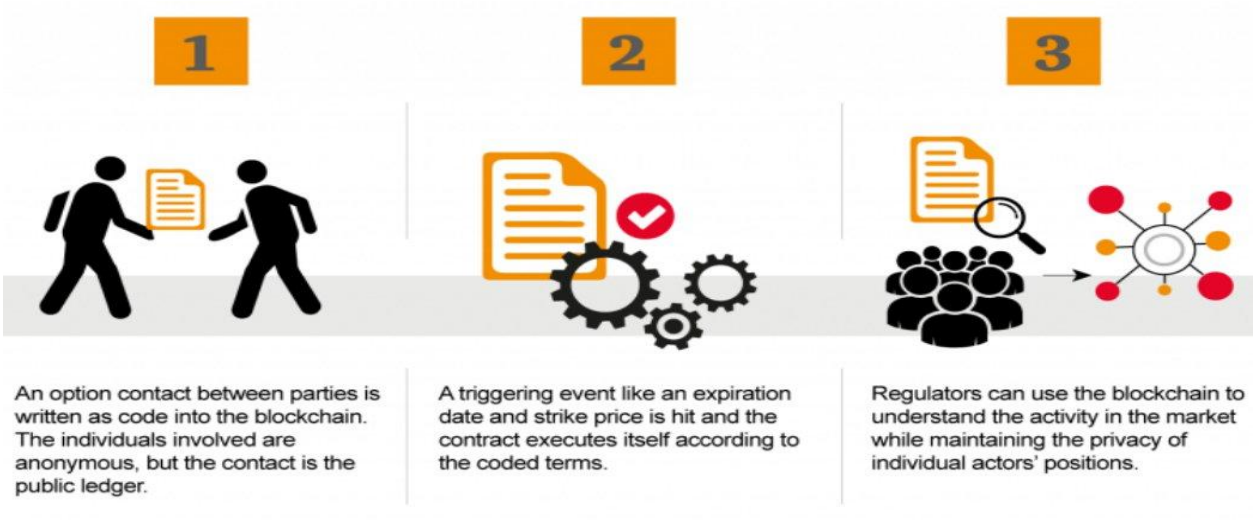

Fig. 1. Example of a Smart Contract.

A. Blockchain As A Service (Baas)

Blockchain as a Service (BaaS) is a cloud-based solution that allows companies to develop, test, and deploy bespoke blockchain applications without the need for initial capital investment. This solution offers services such as consensus mechanisms and validation protocols, allowing businesses to develop their own solutions. BaaS enables businesses to explore blockchain technology and concentrate on developing applications, letting customers and developers create blockchain extensions for current applications across various industries, thereby unlocking the potential of this transformative technology.

Leading enterprise software providers such as IBM, Microsoft, and SAP have initiated the rollout of their cloud-based blockchain offerings. For instance, IBM unveiled IBM Blockchain in March 2017, marking the debut of an enterprise-grade blockchain service built on the Linux Foundation's Hyperledger Fabric version 1.0.

This service empowers developers to swiftly construct and deploy secure blockchain networks on the IBM Cloud, leveraging the robustness of IBM Linux ONE [6], renowned for its unparalleled security features. Similarly, Microsoft introduced the Coco Framework, an open-source solution tailored to create high-scale, confidential blockchain networks meeting enterprise demands [7].

Coco's unique ledger construction approach ensures scalability, distributed governance, and heightened confidentiality without compromising on security and immutability. Furthermore, Microsoft offers Blockchain as a Service (BaaS), a cloud-based platform on Microsoft Azure, providing organizations with a rapid, cost-effective, and low-risk solution. SAP also launched SAP Leonardo, a digital innovation system, in May 2017. This platform seamlessly integrates cutting-edge technologies, including pre-built blockchain capabilities and the SAP Cloud Platform Blockchain service, empowering customers to redefine their business operations effectively.

B. Importance of Software Testing

The main aim of software development is to create products that meet customer needs effectively. As developers, it's crucial to embrace and utilize advanced technologies and programming languages to ensure product quality and value. The software development life cycle (SDLC) offers a standardized process for creating high-quality software, with testing playing a vital role [8]. Testing is essential for delivering quality products to customers. Traditionally, testing occurred towards the end of the coding phase or before product delivery. However, starting testing early in the development cycle can reduce defects and minimize rework costs. Principle 3 of software testing standards [9] emphasizes early testing, stating: “Principle 3: Early testing: Testing activities should start as early as possible in the software or system development life cycle and should focus on defined objectives. Involving the testing team from the beginning, during requirements and analysis phases, promotes a better understanding of the product and significantly reduces the cost of quality by identifying defects early in the development life cycle.”

As mentioned in "Inspecting Requirements" by Wiegers [10], industry data indicates that around 50% of product defects originate from requirement elicitation. Additionally, approximately 80% of coding or design revisions result from prerequisite faults. "Software Testing Techniques" by Beizer [11] offers a comprehensive compilation of testing methodologies, with the author noting that designing tests is highly effective in preventing bugs.

This underscores the significance of early testing, as detecting errors early in the process is crucial. Smart contract testing requires efficient test design to assess and validate contracts before deployment, emphasizing the importance of thorough testing practices.

C. Types and methodologies of Software Testing

Testing seeks to assess different aspects of software to verify that it meets its design and requirements as specified by the client. The software testing process involves a series of activities that include executing the software code and evaluating its various properties to identify any errors present. However, it is limited in its ability to detect the absence of certain features due to the constraints of time and resources, resulting in incomplete testing. In the software development life cycle (SDLC), there are four specific levels of testing: unit testing, integration testing, acceptance testing, and System testing.

- Unit testing ensures the proper functioning of individual software code modules.
- Integration testing aims to evaluate the functionality of units when integrated.
- System testing verifies that the end-to-end system meets its predefined requirements.
- Acceptance testing is conducted to ensure that the system aligns with business requirements and meets the criteria for delivery to end-users.

Additionally, there are various kinds of software testing [12], which can range from straightforward static analysis to dynamic methods, as depicted in Figure 2. Static analysis refers to testing that doesn't involve executing the program but rather examines the structure, syntax, and data flow of the source code.

Common methods for static analysis include inspections, reviews, and step-by-step analysis. Static analysis can be used alongside procedural verification techniques to validate the accuracy of the software. Conversely, dynamic testing involves executing the program, processing inputs, and generating outputs.

Experiments are often conducted to evaluate test data and results, with the aim of confirming whether actual results conform to expected outcomes. Dynamic testing techniques can also encompass both functional and non-functional testing.

Functional testing processes are conducted to verify the accuracy of a product's functional specifications. This type of testing is also referred to as discovery testing. Specific white-box testing techniques, such as interface testing and mutation testing, are employed in this context.

Non-functional testing covers various testing methods designed to evaluate non-functional requirements like usability, performance, security, and compliance. The goal is to verify if system components meet the specified requirements. Non-functional testing is designed to evaluate the system's ability to handle user interactions while meeting nonfunctional parameters that are typically not covered by functional testing approaches.

III. SMART CONTRACT TESTING

Song et al. [13] presented a novel approach to detecting vulnerabilities in Ethereum smart contracts. The authors propose a multi-layer perceptron (MLP) based detection tool that leverages both opcode and control flow graph (CFG) features. This tool integrates static analysis and machine learning to identify various vulnerabilities such as reentrancy, arithmetic issues, unauthorized sends, and timestamp dependencies. The methodology includes extracting opcodes, simplifying them for better feature extraction, and using trigrams for n-gram analysis to capture contextual information.

Driessen et al. [14] introduced SOLAR (Solidity Analyzer), a tool designed to test and verify the correctness of Solidity smart contracts. It emphasizes the importance of security and correctness in smart contracts due to their immutable nature once deployed on the blockchain. The motivation behind developing SOLAR stems from the numerous security vulnerabilities and bugs found in existing smart contracts, which can lead to significant financial losses. The paper highlights notable examples of such vulnerabilities and their consequences.

Tamer et al. [15] introduced a new tool called DLVA , designed to detect vulnerabilities in Ethereum smart contracts using deep learning techniques. DLVA leverages deep learning to analyze Ethereum smart contracts without relying on predefined patterns, manual feature engineering, or expert rules.

In [16] the authors propose a novel tool, SoliDetector, which leverages a knowledge graph to enhance the detection of defects in Solidity smart contracts. This approach outperforms existing tools like Mythril, Slither, and SmartCheck by maintaining higher precision and lower false-negative rates across various defect types. The knowledge graph approach allows SoliDetector to manipulate code information using SPARQL queries, achieving superior defect localization and scalability. Experimental results demonstrate that SoliDetector not only achieves a precision of 97.93% but also operates efficiently, requiring significantly less time to analyze smart contracts compared to other tools.

Sangharatna et al. [17] SmartMuVerf, a novel approach to mutation verification for smart contracts, particularly focusing on Ethereum. The goal is to enhance the testing and verification of smart contracts, which are critical in managing high-value assets and ensuring the correct execution of decentralized applications on blockchain platforms.

Anna Vacca [18] conducted an extensive investigation into different methods, software tools, and obstacles related to blockchain smart contracts development. The comprehensive study encompassed six main areas: testing of smart contracts, evaluation of smart contract code, metrics and security for smart contracts, performance of decentralized applications, and blockchain applications.

Testing serves not only to achieve code coverage but also to identify defects or bugs. Chen et al. [19] assert that smart contracts, being immutable, necessitate being devoid of defects. Through an analysis of various real-world smart contracts and related literature, the authors identified 20 types of malicious defects. The removal of these defects enhances the integrity of smart contracts. Additionally, the study outlines the repercussions stemming from contract defects, thereby aiding developers in recognizing fault indicators and taking appropriate corrective actions.

Zhang et al. [20] discussed the creation of efficient test cases using data flow analysis from a control flow graph. Their strategy employs a genetic algorithm to sequentially optimize dynamic testing for smart contracts. This process begins with the creation of a control flow graph from the source code, which is then examined to determine critical statements and definition-use pairs specific to Solidity programs. Ultimately, they utilize a genetic algorithm combined with a function to generate test cases by calculating these definition-use pairs.

Wang et al. [21] introduced a novel multi-objective approach, incorporating randomness and NSGA-II, aimed at generating cost-effective test suites. While numerous researchers have proposed various testing and verification methods for smart contracts, little emphasis has been placed on factors such as transaction costs, duration (time taken to execute transactions), and the coverage of untested code branches. Addressing these concerns, Wang et al. [21] focused on achieving cost-effective test suite generation for smart contracts through the implementation of the aforementioned multi-objective approaches. Additionally, the authors indicated that maximizing mutation-killed ability during test-suite generation remains an area for future investigation.

Several research studies have proposed testing methodologies for smart contracts using fuzzy approaches to detect faults or security vulnerabilities. Tools like EVMFuzz [22], ContractFuzzer [23], and Fuse [24] have been developed for this purpose. EVMFuzz [22] focuses on identifying weaknesses in Ethereum Virtual Machines (EVM) by employing differential fuzz testing. The core idea involves continuously generating smart contracts and feeding them into the EVM to uncover as many unpredictable outcomes as possible, aiding in vulnerability detection through output cross-referencing. The tool generates smart contracts with predefined mutators and utilizes dynamic priority scheduling. Notably, EVMFuzz has successfully uncovered several previously unknown security bugs.

A tool known as Contract Fuzzer [23] has been designed to test the reliability of smart contracts. It generates fuzzing input based on the specific application binary interface of smart contracts, then analyzes and logs any identified vulnerabilities. Out of approximately 6991 Smart Contracts tested, this tool detected over 459 flaws with high accuracy, including significant issues such as "The DAO" bug and the Parity Wallet bug, which resulted in substantial financial losses amounting to millions of dollars.

The majority of researchers employed Ethereum technology for creating tests for smart contracts. Within each category examined, the authors outlined several unresolved issues. Their analysis on the literature pertaining to smart contracts and blockchain applications indicates a requirement for a development life cycle based on blockchain technology.

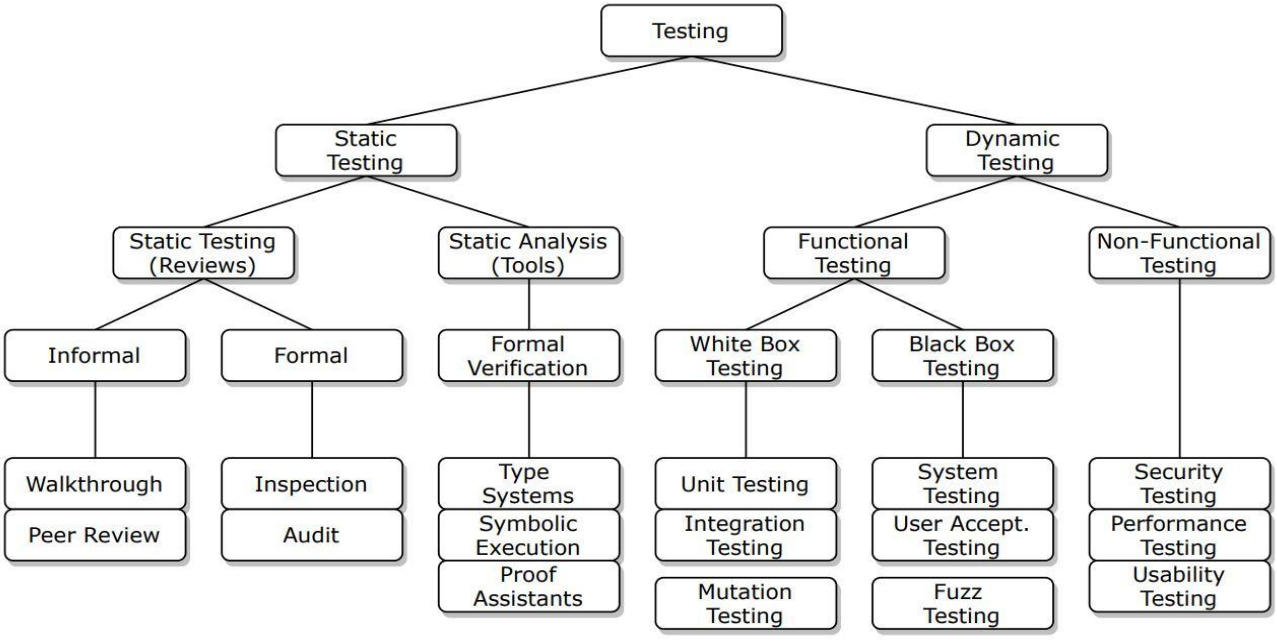

Fig. 2. Overview of Software Testing Types [12]

TABLE 2. The Advantages and
Drawbacks of the research works examined

Paper	Advantages	Drawbacks
[13]	- The use of a Multi-Layer Perceptron (MLP) model allows for the dynamic training of the tool to newer smart contract bugs and Solidity versions. This approach helps in reducing the false positive rate and adapts to the evolving nature of smart contracts.	- The use of synthetic data generation methods like SMOTE for introducing vulnerabilities into the training dataset may not accurately represent real-world scenarios. This can lead to imbalances in the training dataset and affect the overall performance of the model.
[14]	- Automatic generation of high-coverage test suites. - Simulates various blockchain scenarios for thorough testing. - Supports extensions for more algorithms and techniques.	- Techniques like formal verification can be resource-intensive. - Reliance on developer-provided models can limit effectiveness.
[15]	- The authors use training algorithms that are robust enough to cope with problems like mislabeled contracts and increase the analysis capabilities from source code to bytecode in terms of speed and accuracy.	- While it gives speed, there were no such discussions about whether it is scalable in DLVA when deployed and expanded at a large scale across thousands of nodes or whether the tool can manage many contracts at a time. This includes potential bottlenecks in data processing and analysis.
[16]	- The SoliDetector tool achieves a high precision rate of 97.93%, indicating its accuracy in identifying defects without generating a significant number of false positives. - The knowledge graph approach allows for scalable analysis and is capable of handling a large number of smart contracts without compromising performance.	- The authors acknowledge the need for future improvements, such as incorporating execution information to enrich the knowledge graph, which suggests that the current implementation is still evolving and may require further refinement to handle more complex scenarios.
[17]	- Proposes a significant improvement in the testing of smart contracts. The authors represent a novel approach to mutation testing, which could be helpful in ensuring the solidity and security of smart contracts before deploying them onto a blockchain.	- SmartMuVerf focuses on logical and relational operator mutations in the current implementation but does not consider the other types of faults that may cause a substantial contribution to the lack of robustness in smart contracts, such as state management, arithmetic operations, and external calls.
[18]	- The authors identified several open challenges within six principal categories, suggesting the need for further research and development to address these issues effectively.	- There is a demand for a blockchain-oriented software development life cycle. - There is a need to create specialized tools for specific types of testing, such as mutation testing, which are essential for ensuring the quality of smart contracts.
[19]	- Improved robustness of smart contracts.	- The defect dataset can be used in developing smart contract defect detection tools.
[20]	- Proposed ADF-GA for Solidity based Ethereum smart contract programs that generate test cases by using all-uses data flow and genetic algorithms, achieves more coverage and has less number of replays in the genetic algorithm.	- Requires further optimization in selecting and the operation of mutants for generating test cases. - Needs to investigate more suitable methods for the evaluation of test cases. - There are restrictions executing in the environment of smart contracts for performing dynamic testing, which requires a solution.
[21]	- Provides cost efficient test cases	- Requires maximizing the ability of killed mutants during the test-suite generation.

[22]	- EVMFuzz automates the generation of test cases, significantly reducing manual effort and increasing testing efficiency. - The framework successfully identified numerous inconsistencies in gas usage and opcode sequences among different EVM implementations, revealing hidden vulnerabilities that could lead to serious security issues.	- While EVMFuzz is effective in identifying inconsistencies, it may also produce false positives and negatives, necessitating further manual analysis to confirm the findings. - Some generated seed contracts may contain rare situations, such as illegal data types or infinite loops, which certain EVM platforms may not handle correctly, leading to potential crashes or erroneous outputs
[23]	- ContractFuzzer defines new test oracles that can precisely detect real-world vulnerabilities in smart contracts. - ContractFuzzer automatically generates inputs based on the API interfaces of smart contracts, making the testing process more efficient and less prone to human error.	- The immutability of smart contracts makes updating them after deployment difficult, which can delay the application of security fixes.
[24]	- Fuse has shown high true positive rates (96-100%) in detecting various vulnerability classes. - Provides comprehensive test reports and visualizations to aid developers in understanding vulnerabilities. - Encrypted submission of contracts and test reports ensures privacy and security.	- Offline analysis can be inefficient, and online analysis may slow down smart contract execution. - Integration of Dapp testing with fuzz testing of smart contracts is still under study and not fully realized.

Table 2 presents a synthesis of the Advantages and Drawbacks of the research works examined in this literature review. In the table above, we highlighted the evolution and focus of research in blockchain and smart contract development, emphasizing the following:

- Testing methodologies
- Vulnerability detection
- The importance of rigorous software testing practices

IV. Challenges in Testing Blockchain Smart Contract

This section outlines the various triggers inherent in testing smart contract executions. The main challenges are:

A. Public vs Private blockchain

The scope and thoroughness of testing are greatly affected by whether the implementation is carried out on a public platform such as Ethereum or on a customized platform designed specifically for an organization or consortium [25, 26]. Testing private blockchains tends to be more straightforward due to the controlled environment, which facilitates the simulation of different scenarios and allows for internal testing using conventional methods.

A detailed test strategy can be developed due to the customized functionality. However, testing becomes more complex with public platform implementations. Public blockchains lack limits on node participation, face challenges in achieving consensus [27], may experience transaction speed reductions, hard fork occurrences, and other issues, making it challenging to design comprehensive test strategies and cases.

B. Security Vulnerabilities

The widespread adoption of blockchain can be attributed to the inherent security it offers. Alongside significant security benefits, there are also certain challenges that need attention. Failure to address these challenges appropriately could lead to severe consequences, particularly within the finance industry. Whether dealing with public or private blockchains, a key challenge lies in ensuring the security of the network architecture through thorough testing. Nodes within a network may exhibit unresponsiveness or sporadic activity for questionable reasons. Comprehensive testing involving various scenarios is imperative to safeguard the consensus process and maintain ledger consistency.

In the realm of private blockchain systems, there may be occasions where transaction reversibility is essential, perhaps due to attempted theft or other unforeseen circumstances. In such instances, it becomes imperative to conduct thorough validation checks to uphold the integrity of the distributed ledger. Testing procedures must ensure that transactional data remains intact throughout such processes. Private blockchains, like other distributed architectures, rely on network communication for accessing and updating transactional data, rendering them susceptible to denial-of-service and identity-based attacks [28].

Consequently, meticulous testing is indispensable to mitigate these risks and fortify the system against potential vulnerabilities. Despite blockchain technology's overarching

focus on security, it encounters challenges that may not be entirely addressed within existing test frameworks and methodologies.

C. Decentralized Environment

Proper testing in blockchain implementations relies heavily on having a test environment that mirrors the production setup [29]. Access to a test platform resembling the actual deployment is crucial; without it, considerable resources and time may be required to replicate the environment.
Open-source implementations offer distinct test instances that mimic real scenarios, facilitating comprehensive testing of transaction functionalities. Private blockchain setups demand tailored test environments to accommodate customized features effectively.

As blockchain integration becomes more prevalent within existing business frameworks, integration testing assumes greater importance. Rather than standalone development, blockchain is increasingly being integrated into pre-existing applications. Ensuring seamless interaction between blockchain implementations and existing systems poses a significant challenge.

Testing teams must thoroughly understand interface points and communication channels to maintain consistency across processes. Providing access to application programming interfaces (APIs) used for communication enables quality assurance teams to validate proper integration between legacy systems and blockchain components.

D. Lack of Standardization

The lack of understanding of innovation and inadequate skills or experience in designing and developing smart contracts are key factors. Furthermore, the lack of standardization [30] in terminology leads to decreased clarity, quality, and efficiency. Establishing accepted practices for developing test suites through open-source collaboration would greatly benefit smart contract implementation and enhance product quality. Blockchain, being a relatively new technology, requires a deep understanding of its concepts and potential to be effectively applied in various domains, such as healthcare or supply chain management.

Despite being introduced by Satoshi Nakamoto [31] in 2008 as a critical underlying technology in the Bitcoin framework, widespread adoption is hindered by the absence of established best practices.

Both technical and non-technical skills are essential for effective and comprehensive testing. Addressing this challenge is complex, as acquiring additional skills, such as proficiency in the latest smart contract development languages or understanding best practices in implementing blockchain smart contracts, can be costly. Furthermore, even with successful smart contract implementation, testing remains highly specialized and demands experience and a meticulous testing approach.

E. Limited Tooling

The testing tools aim to execute the system being tested, ensuring consistency in performance and thereby validating the effectiveness of the testing process. In this context, the system under test refers to the blockchain application, which demands considerable effort and lacks comprehensive automation support. Choosing the appropriate tool is a critical decision.

To tackle this challenge, there are some tools with non-proprietary software implementations available. While these tools may not perfectly replicate real-world scenarios, they can still effectively test certain high-level transaction functionalities. Ideally, an integrated development environment offering essential modules, compilers, testing frameworks, and analytical tools is required. However, existing variations of these tools fall short of meeting developers' needs [32].

Anna Vacca [18] explores how conventional testing techniques like mutation testing, unit and integration testing, and regression testing need to be tailored to suit the distinctive interaction and architecture of smart contracts. There is a demand for the development of testing tools specifically designed for smart contracts to effectively identify vulnerabilities and bugs in the source code. Specialized tools, particularly for mutation testing, are essential for ensuring the quality of smart contracts.

V. APPROACHES TO TESTING

In the modern era, there is an elevated level of trust worldwide, primarily due to our growing reliance on software to execute a wide range of transactions. Blockchain technology has heightened visibility within these transaction networks, enabling asset information that was once confined to individual owners to be shared among all stakeholders. As a result, testing teams are facing an unprecedented demand to ensure top-notch quality on their first attempt, all while minimizing any negative impact on team productivity during the testing process.

The importance of testing is underscored by the rise of smart contracts in business. Functioning as APIs on blockchain networks, smart contracts are immutable once deployed, making thorough testing crucial for accuracy and reliability. Detecting defects in production systems necessitates creating and deploying new versions, with manual data transfer posing logistical challenges. Furthermore, the immutable nature of smart contracts precludes updates, amplifying the responsibility of the quality assurance team to ensure accuracy from the start.

The rise of DevOps has transformed the role of testers [33], with a greater emphasis on continuous testing. A robust strategy for continuous testing involves testing at the earliest stages in production environments (shift-left), integrating testing into

every deployment process (shift-right), and employing continuous testing throughout the software lifecycle (automation). Given the criticality of software quality, various testing approaches are employed:

A. Consensus

In trusted business networks, transactions are confirmed and recorded using consensus mechanisms such as proof-of-stake, multi-party signatures, and PBFT algorithms [34]. Regardless of the consensus method, validated data serves as the definitive record. Testers must ensure consistency and integrity among stakeholders, considering the dynamic nature of nodes and their impact on data flow and blockchain performance. Service virtualization is valuable for simulating real-world scenarios by controlling component interactions via APIs.

For example, reintegrating a party may require a brief data update, affecting verification speed with more nodes. A thorough testing strategy covering various scenarios is essential to prevent system breakdowns.

B. External Integration

In many contemporary business contexts, blockchain technology is strategically integrated into specific stages of the overall process [35]. This integration necessitates seamless interaction with other interconnected components within the system. For instance, the blockchain may trigger events in external systems, or conversely, external events may trigger actions within the blockchain. Additionally, blockchain operations may rely on external services or routines to execute certain functions effectively. A robust testing strategy entails meticulous examination of every interaction point between the blockchain ecosystem and external systems to ensure the accurate flow of data at each junction.

For instance, when submitting a completed transaction to an API, validation against predefined rules, consensus attainment, and ledger updates must all be thoroughly verified, not solely confirmation receipts. Neglecting validation at any stage poses a risk of ledger corruption and the dissemination of inaccurate information. Given that most blockchains utilize REST-based APIs for interactions, comprehensive testing is imperative to guarantee seamless integration.

C. Functional Testing

Smart contracts are code that automatically executes with each transaction based on predefined rules. It is important to prioritize testing the functionality of the contract over focusing solely on the data. Ensuring functional correctness, along with validating the data, is crucial. A recommended testing strategy emphasizes the importance of functional tests to ensure that terms and conditions are correctly programmed into the contract and activated by appropriate data inputs [36]. It is advisable to perform unit tests on each condition independently, followed by integration testing, to evaluate the complete logic of the smart contract as a whole. Moreover, due to their immutable nature, it is considered best practice to test smart contracts on a simulated blockchain or virtualized service before deployment.

D. Performance Testing

Performance can be evaluated in two primary ways: one approach involves assessing it from the perspective of external end users interacting with a blockchain solution, while the other utilizes feedback obtained from internal system interfaces. It's important to perform performance testing against the services offered by the blockchain ecosystem to assess the impact of potential failures. Monitoring performance during high-risk situations such as data updates across nodes, node synchronization, and consensus processes is crucial, as these scenarios are heavily influenced by data location and latency.

Automated performance testing should be used to evaluate scalability in all these cases [37]. Testers should prioritize speed while ensuring that horizontal and vertical scaling do not compromise performance. In applications like Bitcoin, where individuals freely open digital wallets and engage in exchanges, continuous performance testing and monitoring are essential. Testers should create scenarios that cover all aspects of the blockchain ecosystem, including compound testing

.

E. Security Testing

The rise of blockchain technology has ushered in a new era where all participants have equal roles in securing the data and network. Despite its foundation on security and immutability, blockchain remains vulnerable to security risks [38]. Those who own private blockchains should be cautious about how their existing business solutions interact with blockchain security. Smart contracts pose an even greater challenge for regulators or consensus bodies, as they are deeply integrated into every transaction on the blockchain.

Testers must scrutinize both the implementation code for security deficiencies and the distributed architecture for potential threats that could result in denial-of-service attacks or the exploitation of vulnerabilities. Proper modeling and anticipation of these threats are necessary to develop effective mitigation strategies and detailed fixes. Implementing automated security testing provides a more reliable method for ensuring integrity within diverse applications using ledger systems; fuzz testing, aimed at automating vulnerability discovery by intentionally injecting faults into the system, can also be beneficial. Integrating security testing throughout DevOps processes across implementation deployments will offer comprehensive protection.

VI. CONCLUSION

In the dynamic landscape of blockchain technology and the widespread integration of smart contracts across various sectors, it's imperative to evaluate the current technological status and identify areas ripe for improvement. To this end, our

survey on smart contract testing aims to delve into the methodologies, practices, and tools utilized to address challenges in smart contract development. Despite advancements, certain issues persist, necessitating further research to adapt traditional testing methodologies to blockchain-based development. Moreover, enhancing programming languages for smart contract coding and providing guidelines for streamlined development processes are crucial. Additionally, specialized testing tools and frameworks are essential, regardless of the language or platform used for smart contract development.

Successful adoption of blockchain methodologies requires a meticulous approach to design and validation. While testing mechanisms may mirror those of conventional systems, specific attention must be paid to tailor test strategies in alignment with the principles and implementation constructs of the domain.

VII. REFERENCES

[1] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han and F. -Y. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266-2277, Nov. 2019.

[2] O. Ali, A. Jaradat, A. Kulakli and A. Abuhalimeh, "A Comparative Study: Blockchain Technology Utilization Benefits, Challenges and Functionalities," in *IEEE Access*, vol. 9, pp. 12730-12749, 2021.

[3] Chen, G.; He, M.; Gao, J.; Liu, C.; Yin, Y.; Li, Q. Blockchain-Based Cyber Security and Advanced Distribution in Smart Grid. In Proceedings of the IEEE 4th International Conference on Electronics Technology, Chengdu, China, 7–10 May 2021.

[4] Brandstatter, T.; Schulte, S.; Cito, J.; Borkowski, M. Characterizing Efficiency Optimizations in Solidity Smart Contracts. In Proceedings of the IEEE International Conference on Blockchain, Toronto, ON, Canada, 3–6 May 2020.

[5] Murugan, S.; Kris, S. A Survey on Smart Contract Platforms and Features. In Proceedings of the 7th International Conference on Advanced Computing and Communication Systems, Coimbatore, India, 19–20 May 2021.

[6]] IBM News Room 2017, Available online at https://www-03.ibm.com/press/us/en/pressrelease/51840.wss

[7] Announcing Microsofts Coco Framework for enterprise Blockchain Networks, available online at https://azure.microsoft.com/en-in/blog/announcing-microsoft-s-coco-framework-for-enterprise-blockchain-networks/

[8] N. Sánchez-Gómez, L. Morales-Trujillo, J. J. Gutiérrez and J. Torres-Valderrama, "The Importance of Testing in the Early Stages of Smart Contract Development Life Cycle," in *Journal of Web Engineering*, vol. 19, no. 2, pp. 215-242, March 2020.

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

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

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

[12] IEEE Approved Draft International Standard for Software and Systems Engineering--Software Testing--Part 4: Test Techniques, in *ISO/IEC/IEEE P29119-4-FDIS April 2015* , vol., no., pp.1-147, 8 Dec. 2015.

[13] L. S. H. Colin, P. M. Mohan, J. Pan and P. L. K. Keong, "An Integrated Smart Contract Vulnerability Detection Tool Using Multi-Layer Perceptron on Real-Time Solidity Smart Contracts," IEEE Journals & Magazine | IEEE Xplore, 2024. https://ieeexplore.ieee.org/abstract/document/10430147/

[14] S. W. Driessen, D. Di Nucci, D. A. Tamburri, and W. J. Van Den Heuvel, "SolAR: Automated test-suite generation for solidity smart contracts," Science of Computer Programming, vol. 232, p. 103036, Jan. 2024, doi: 10.1016/j.scico.2023.103036.

[15] Tamer Abdelaziz and Aquinas Hobor. Smart learning to find dumb contracts (extended version). 2023.

[16] T. Hu, B. Li, Z. Pan and C. Qian, "Detect Defects of Solidity Smart Contract Based on the Knowledge Graph," in *IEEE Transactions on Reliability*, vol. 73, no. 1, pp. 186-202, March 2024,10.1109/TR.2023.3233999

[17] Sangharatna Godboley and P. Radha Krishna. Smart Contract Test Case Prioritization based on Frequency and Gas Consumption.IEEE Conference Publication. 2023.

[18] Anna Vacca, Andrea Di Sorbo, Corrado A Visaggio , Gerardo Canfora. 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.

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

[20] Zhang, P., Yu, J., Ji, S. 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, 2020.

[21] Wang, X., Wu, H., Sun, W., Zhao, Y.. 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, 2019.

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

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

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

[25] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic Literature Review of Challenges in Blockchain Scalability," Applied Sciences, vol. 11, no. 20, p. 9372, Oct. 2021, doi: 10.3390/app11209372.

[26] S. Li, Q. Xu, P. Hou, et al., "Exploring the Challenges of Developing and Operating Consortium Blockchains: A Case Study," ACM Int. Conf. Proceeding Ser., pp. 398–404, 2020.

[27] V. Yussupov, G. Falazi, U. Breitenbucher, ¨ et al., "On the serverless nature of blockchains and smart contracts," arXiv, 2020.

[28] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart Contract: Attacks and Protections," IEEE Access, vol. 8, pp. 24 416–24 427, 2020.

[29] P. Sharma, R. Jindal, and M. D. Borah, "A review of smart contract-based platforms, applications, and challenges," *Cluster Computing*, vol. 26, no. 1, pp. 395–421, Jan. 2022, doi: 10.1007/s10586-021-03491-1.

[30] S. D. Kotey *et al.*, "Blockchain interoperability: the state of heterogenous blockchain-to-blockchain communication," *IET Communications*, vol. 17, no. 8, pp. 891–914, Mar. 2023, doi: 10.1049/cmu2.12594.

[31] Satoshi Nakamoto, "Bitcoin: A peer-to-peer electronic cash system",2008 , http://bitcoin.org/bitcoin.pdf

[32] V. Piantadosi, G. Rosa, D. Placella, S. Scalabrino, and R. Oliveto, "Detecting functional and security-related issues in smart contracts: A systematic literature review," *Software, Practice & Experience/Software, Practice and Experience*, vol. 53, no. 2, pp. 465–495, Oct. 2022, doi: 10.1002/spe.3156.

[33] Mikael Krief, Learning DevOps: A comprehensive guide to accelerating DevOps culture adoption with Terraform, Azure DevOps, Kubernetes, and Jenkins , Packt Publishing, 2022.

[34] Z. Hussein, M. A. Salama, and S. A. El-Rahman, "Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms," *Cybersecurity*, vol. 6, no. 1, Nov. 2023, doi: 10.1186/s42400-023-00163-y.

[35] Z. Rasheed and M. Mimirinis, "Integrating Blockchain Technology into a University Graduation System," *Trends in Higher*

Education, vol. 2, no. 3, pp. 514–525, Aug. 2023, doi: 10.3390/higheredu2030031.

[36] H. Chu, P. Zhang, H. Dong, Y. Xiao, S. Ji, and W. Li, “A survey on smart contract vulnerabilities: Data sources, detection and repair,” *Information and Software Technology*, vol. 159, p. 107221, Jul. 2023, doi: 10.1016/j.infsof.2023.107221.

[37] Y. Ucbas, A. Eleyan, M. Hammoudeh, and M. Alohaly, "Performance and Scalability Analysis of Ethereum and Hyperledger Fabric," *IEEE Access*, vol. 11, pp. 67156-67168, 2023, doi: 10.1109/ACCESS.2023.3291618.

[38] F. Jiang *et al.*, “Enhancing Smart-Contract Security through Machine Learning: A Survey of Approaches and Techniques,” *Electronics*, vol. 12, no. 9, p. 2046, Apr. 2023, doi: 10.3390/electronics12092046.