Cluster Computing
https://doi.org/10.1007/s10586-024-04421-7

A systematic mapping on software testing for blockchains

Anıl Elakaş[1] · Hasan Sözer[2] · Ilgın Şafak[1] · Kübra Kalkan[2]

Received: 15 November 2023 / Revised: 3 March 2024 / Accepted: 5 March 2024
© The Author(s) 2024

Abstract
The purpose of this study is to identify and classify studies published on software testing techniques applied to blockchain systems. Previously published reviews in related areas have a narrow focus and/or do not follow a systematic review protocol. We conducted a systematic mapping based on an initial selection of 1025 studies. A rigorous selection process resulted in a final pool of 17 primary studies. These studies are categorized with respect to the employed testing methods, considered quality attributes, and functionality. We observe that most of the publications focus on testing functional correctness or security, whereas the testing of runtime performance attracts less attention. Existing approaches mostly employ fuzz testing or mutation testing. Search-based testing is usually combined with these techniques. The application of model-based testing is rare. The adaptability of fuzz testing and model-based testing techniques to changing blockchain platforms and languages remains a concern. On the other hand, performance and scalability issues are noted for search-based techniques and mutation testing. The use and integration of multiple testing techniques also stand out as a viable research direction.

Keywords Software testing · Blockchain · Testing techniques · Systematic literature review · Survey

1 Introduction

Blockchain stores information digitally as distributed databases shared by nodes on peer-to-peer networks, such as those used by cryptocurrency systems. Blockchain removes the necessity of a central authority, provides immunity to counterfeits, and delivers robust protection through strong and complex algorithms [1]. Its utilization enables users to manage their online secure transactions without the need for an external intermediary or control mechanism, within a peer-to-peer network of computers (nodes) distributed around the world [1]. A blockchain is immutable, which means that the data written on a block cannot be reversed [1]. All these features increase the complexity of such systems, which can be mission-critical and subject to high levels of reliability requirements. As a result of the immutability of the blockchain, a bug that gets into the production system may require a complete rewrite of the code. Due to the decentralized nature of the system and the anonymity of the nodes, ensuring reliability and other quality attributes is difficult.

Effective testing techniques are essential for improving user experience, reducing failure risks, pinpointing vulnerabilities, uncovering performance bottlenecks, and preventing potential financial losses and reputation damage caused by faulty software. There are a variety of testing techniques that can be applied in different contexts. The application of a particular technique may be affected by the quality attribute of interest, the type of fault to be detected, and the type of software system. Software testing is the most commonly applied approach ensuring that required functionality and quality attributes are provisioned by a software system [2]. For blockchain systems to be reliable, the right blend of software testing techniques is necessary.

✉ Anıl Elakaş
anil.elakas@fibabanka.com.tr

Hasan Sözer
hasan.sozer@ozyegin.edu.tr

Ilgın Şafak
ilgin.safak@fibabanka.com.tr

Kübra Kalkan
kubra.kalkan@ozyegin.edu.tr

[1] R&D and Innovation, Fibabanka, Istanbul, Türkiye

[2] Department of Computer Science, Ozyegin University, Istanbul, Türkiye

Published online: 07 April 2024

This need has attracted researchers' attention in recent years.

As a result of the increasing number of publications in the field of blockchain and software testing, practitioners and researchers might find it increasingly difficult to review and understand current state-of-the-art practices in this area. The purpose of this paper is to report the results of a Systematic Mapping (SM) study that we conducted to systematically gain an overview of existing studies related to blockchain system testing and to categorize them. Although there exist several secondary studies (summarized in Sect. 3) that focus on smart contracts' security and reliability, they do not follow a systematic approach in general. A single study [3] applies a Systematic Literature Review (SLR) [4]; however, the scope of the study is limited to only two databases, and the categorization of primary studies is conducted along a single dimension that focuses on quality attributes such as performance. Likewise, other existing surveys focus only on a single quality attribute like security [5] or a single testing technique like mutation testing [6]. They do not provide a comprehensive overview of quality attributes considered and alternative testing techniques employed. In addition, in contrast with this SM, they do not categorize existing primary studies based on both of these aspects.

Our review protocol involves four databases resulting in 1025 primary studies. A rigorous selection process narrows the list to a final pool of 17 primary studies. These studies are categorized based on both the testing methods used and the quality attributes they focus on. We observe that most of the publications focus on testing functional correctness and mutation testing, as well as security and fuzz testing. There are several techniques and tools in this scope that are mature enough for adoption. The testing of runtime performance attracts less attention. Aside from mutation testing and fuzz testing, 11.76% of the studies employ search-based testing, although this technique is usually combined with the other two techniques to improve their efficiency. Only 17.64% of primary studies focus on model-based testing. Hence, there is room for further research to explore the potential applications of search-based and model-based testing in evaluating blockchain systems. There are certain limitations of the existing techniques that are emphasized in primary studies. Low recall and lack of adaptability to changing blockchain platforms and languages are noted for model-based testing techniques. On the other hand, performance and scalability issues stand out for search-based techniques, fuzz testing, and mutation testing. The use and integration of multiple testing techniques also stand out as a viable research direction.

The rest of this paper is organized as follows. Section 2 provides background information about blockchain and testing approaches. Other secondary studies related to blockchain testing are summarized in Sect. 3. Section 4 outlines the research goal and the review protocol, which includes information about databases, keywords, and inclusion/exclusion criteria. Section 5 discusses and classifies these studies. Section 6 further elaborates on the state-of-the-art, limitations, research gaps and potential future trends. Section 7 provides a discussion of validity threats to our results. Finally, Sect. 8 concludes the paper.

2 Blockchain and testing approaches

We first provide background information on blockchains and Smart Contracts (SCs) in Subsect. 2.1. Then, we discuss software testing in Subsect. 2.2. Finally, we summarize the testing techniques used for blockchain in Subsect. 2.3. The set of these techniques will be used as one of the dimensions for categorizing primary studies.

2.1 Blockchains

Blockchains record transactions in a block with a hash known as an unchangeable cryptographic signature. Timestamped blocks group transactions and each block contains a timestamp, a hash of the previous block, and the number of transactions. Trust is decentralized, i.e., there is no central authority that validates transactions. Each node in a blockchain network stores its own blockchain copy locally. Transaction validity is determined by consensus among the nodes.

Permissionless and permissioned blockchains are two different types of blockchain networks that operate with varying levels of access control and decentralization. Permissionless blockchains, also known as public blockchains, are decentralized networks that allow unfettered participation, eliminating the need for prior approval. They offer open access, rely on a decentralized consensus mechanism, and are characterized by their transparency. These make them suitable for applications that require open participation and public validation of transactions. Prominent examples of permissionless blockchains include Bitcoin and Ethereum.

In contrast, permissioned blockchains, often referred to as private or consortium blockchains, feature restricted access and are primarily utilized by organizations or consortiums of trusted entities. These networks hinge on centralized governance, necessitate permission for participation, and often prioritize data privacy and confidentiality. They are commonly applied in enterprise contexts where controlled access and privacy are paramount, and governance can be overseen by a central authority or a consortium. The preference between permissionless and

permissioned blockchains would depend on the specific requirements of the intended use case and the desired degree of decentralization and access control.

2.2 Software testing

Software testing serves a critical role in the Software Development Life Cycle (SDLC) by systematically evaluating whether the developed software adheres to its intended functionality and quality criteria. It reveals potential discrepancies between requirements/specifications and actual functionality, ultimately aiding in the identification of bugs and missing requirements.

Software testing can be divided into two upper-level categories as *functional testing* and *non-functional testing* as shown in Fig. 1. Functional testing examines whether a software system aligns with its predetermined functional requirements and specifications. It involves providing suitable test inputs, anticipating the expected outputs, and conducting a thorough comparison with the actual system outputs

Functional testing branches out into diverse types such as *unit testing*, *integration testing*, and *system testing* that are applied at various phases of the SDLC. Unit testing examines the smallest testable units of a software system, often referred to as modules or components. These units typically encompass individual functions or procedures. This methodical approach aims to ensure the functional correctness and suitability of each independent unit before their integration. It is typically applied by the developers themselves during the development phase, serving as the initial testing stage within the SDLC, preceding integration testing.

Integration testing examines the interplay and data exchange between distinct components or modules within a software application to identify integration-related issues. Its primary objective is to uncover any issues or defects that might surface when individual elements are combined and begin interacting. It is typically applied after unit testing and before system testing.

System testing stands as a comprehensive evaluation point in software development, carefully examining the entire functionality and quality of a fully integrated system. Its primary focus lies in verifying alignment with established requirements and assessing the system's readiness for delivery to end users.

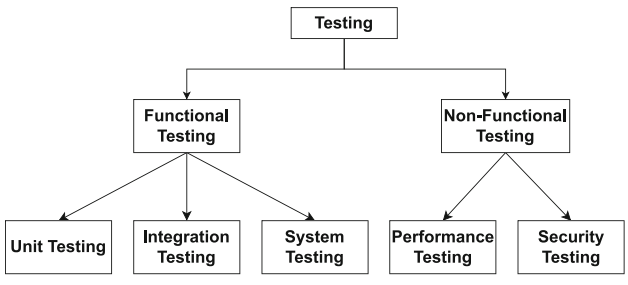

Fig. 1 Types of software testing

Non-functional testing complements functional testing by evaluating the software's adherence to non-functional requirements. These requirements do not specify a particular functionality and as such, they fall out of the scope of functional testing. Non-functional requirements specify constraints regarding the specified functionalities and their quality attributes. Non-functional testing can be categorized into sub-types depending on the quality attribute being considered. We highlight *performance testing* and *security testing* in particular, as the most relevant types of non-functional testing within the scope of our study.

The purpose of security testing is to discover vulnerabilities in the system and determine whether its data and resources are protected from outside intrusions. It ensures that the software system is free from any threats or risks that can cause a loss of information or repute of the organization. Performance testing is crucial for assessing a system's ability to deal with (un)expected workloads. A rigorous approach goes beyond functional correctness to examine speed, responsiveness, and stability under varying pressures. In essence, performance testing acts as a bridge between theoretical capabilities and real-world scenarios, guaranteeing a system's ability to gracefully handle its intended workload and deliver optimal performance in the hands of its users.

A variety of testing techniques are employed to facilitate both functional and non-functional testing. In the following, we review *search-based testing*, *fuzz testing*, *mutation testing*, and *model-based testing*, which turn out to be predominant for testing blockchain systems.

Metaheuristic algorithms are used in search-based testing [2] to generate test inputs and optimize test cases automatically. In particular, genetic algorithms or simulated annealing can be used to improve test coverage by treating testing as a search problem and aiming at the maximization of coverage as the objective. Our study revealed that search-based testing is mainly used for functional testing (See Fig. 5). It is also combined with other testing techniques to improve their effectiveness. The other three techniques explained below are used for both functional and non-functional testing, where security is the most considered quality attribute.

Fuzz testing [7], also known as fuzzing, involves providing invalid, unexpected, or random inputs to a program to identify vulnerabilities and potential software crashes. It is particularly useful for uncovering security vulnerabilities, such as buffer overflows or input validation errors. Therefore, it is mainly used for improving software systems' security and robustness.

Mutation testing [6] involves the injection of intentional faults, known as mutations, into the source code to evaluate the test suite's ability to detect these changes and assess its robustness. It pinpoints areas where tests may be inadequate or need improvement, thereby enhancing the overall quality and effectiveness of the testing process.

Model-based testing [8] uses models of the system under test as a basis for generating and executing test cases. Expected system behavior is abstracted with (semi-)formal models like finite state machines and Unified Modeling Language (UML) diagrams. These models are used for systematically generating test cases, ensuring comprehensive coverage and adherence to system requirements. This approach enhances the effectiveness of the testing process and reduces the amount of manual effort involved in creating and maintaining tests.

2.3 Software testing for blockchain systems

With blockchain technology, the concept of SC emerged to enable decentralized and automated agreement execution. SCs are programs stored on a blockchain that run when predetermined conditions are met. They can be utilized for distributed test case creation and certification purposes. The integration of SCs in software testing introduces a novel approach where the decentralized and automated execution capabilities of smart contracts can be harnessed to streamline and enhance verification and validation processes, thus enabling more efficient and effective software system testing. One can also employ other advanced software testing techniques in assuring the quality of blockchain systems. In the following, we elaborate on the use of various software testing techniques for testing blockchain systems.

Search-based techniques are well suited to addressing the complexity and scalability issues associated with blockchain technology. These techniques can find near-optimal solutions efficiently in large and complex search spaces, which are common in blockchain environments. However, effective fitness functions and search spaces have to be defined in line with the blockchain's unique requirements and characteristics.

Fuzz testing is mainly applied to identifying security vulnerabilities. However, it has some challenges in testing blockchains, including adapting to the decentralized nature of the blockchain and the complexity of SCs. In order to effectively test both the blockchain platform and its associated applications, such as SCs, more blockchain-specific fuzzing tools and techniques are necessary. Although fuzz testing provides insights into the security aspects of blockchain systems, it still needs to evolve to meet the unique requirements of blockchain technologies and keep pace with their rapid development.

Mutation testing for blockchain systems modifies aspects of the blockchain software in order to determine the effectiveness of the test cases. This method helps ensure that complex code, such as SCs, is correct. In order to ensure the high level of security that blockchain requires, it provides a rigorous evaluation of the test suite's ability to detect faults. However, mutation testing for the blockchain can be time-consuming and resource-intensive. It requires considerable programming expertise, making it less accessible to general testing teams. Further research and development are necessary to improve mutation testing so as to effectively handle the unique characteristics of blockchain architectures and smart contracts.

Model-based testing can be used for generating test cases based on models describing the functional features of the blockchain system. As blockchain applications are characterized by intricate interactions and complex business logic, this approach is especially beneficial. However, it is challenging due to the need for highly accurate models, and by the difficulty of setting up and maintaining these models. Continuous refinement of models is necessary to keep up with the rapid development of blockchain technology, especially in modeling blockchain architectures and SC interactions accurately.

As blockchain technologies have become more complex and diversified over the past few years, industrial trends in blockchain testing have evolved significantly. We observed the development of specialized testing frameworks and tools tailored to the blockchain. There has also been an increase in security testing required by the immutable and extremely secure nature of blockchain and a shift toward performance testing in order to evaluate the scalability and efficiency of blockchain networks. In addition, SC testing has become a critical area, with a focus on ensuring that the code that executes contracts automatically is correct and secure [9].

In the following, we summarize literature surveys on testing techniques applied to blockchain systems and SCs.

3 Related work

A number of secondary studies have already been published in the literature on blockchain system verification. The majority of these studies [10–15] provides a survey of formal verification methods rather than testing techniques. There are also surveys [3, 5, 6, 16] that focus on testing. However, some of these [5, 6] do not follow a systematic protocol, and as such they do not present an SM. It appears that there is only one SLR [3] that focuses on the testing of blockchain systems. That review is conducted to identify primary studies on testing of *performance*, *privacy*, and SC issues (i.e., *functionality*) specifically. The search string

adopted in SLR is also aligned with this particular focus, which leads to an initial pool of 82 studies as a result of their search performed on two databases. These studies are categorized based on a single dimension that represents the considered quality attribute for testing, i.e., *performance*, *privacy*, and *functionality*. We perform a categorization along a second (orthogonal) dimension that identifies the types of testing techniques employed.

One of the other existing surveys [5] analyzes and categorizes existing studies with respect to the target blockchain, SC languages, and considered vulnerabilities. Authors identify that existing tools mostly focus on the SC code and do not consider a global view of the blockchain system. A recent SM [16] focuses on SC testing only and performs a categorization along mostly the same aspects, i.e., research method, type of testing, target blockchain, and SC languages. Authors point out the monopolization of research/industry by an attribute or technology in various areas, e.g., Ethereum as a blockchain network. They also point out the lack of extensive evaluations compared to a vast amount of solution proposals. Finally, the survey performed by Sujeetha and Preetha [6] also focuses on SC testing only. Moreover, their study focuses on mutation testing in particular, rather than providing an overview of alternative techniques.

Despite the growing volume of research track records in the area of blockchain-based systems, there are only a few secondary studies. These studies mainly focus on SC testing and they either focus on a particular testing technique or quality attributes considered for testing. They do not provide a comprehensive overview of alternative attributes and techniques, and they do not categorize existing studies based on both of these aspects. Therefore, to address this gap and provide a thorough review, we conducted this study by employing the research goal and protocol provided in the following section.

4 Research goal and the review protocol

The goal of our SM is to provide an overview of the body of knowledge and the state-of-the-art in the intersection of software testing and blockchain technologies. We aim to perform this mapping from two perspectives. First, we want to identify software testing techniques that are applied to testing blockchain systems. Second, we want to identify which quality attributes are verified with these techniques. We formulated the following two research questions (RQs) aligned with these two perspectives:

- *RQ1:* Which testing techniques are employed for testing blockchain systems?
- *RQ2:* Which quality attributes are considered when testing blockchain systems?

In order to conduct an SM, searching for and selecting papers is a crucial step. This process involves searching multiple databases, using specific search terms and inclusion criteria to identify relevant papers, and then carefully screening and selecting them based on their relevance and quality. We followed the steps below to search for and select primary studies:

- Online database source selection
- Defining search keywords
- Application of inclusion and exclusion criteria

This section is organized as follows: Subsect. 4.1 provides information about databases, chosen keywords, and search strings prepared for online databases' search engines. Subsection 4.2 provides detailed information about the inclusion and exclusion criteria, the review process, and the selection phases.

4.1 Online databases and search keywords

In this study, we used online databases by reputable publishers, namely IEEE Xplore, ACM Digital Library, and Science Direct, which are commonly referred to in secondary studies [3]. In addition, we used a generic and inclusive indexing platform, namely Google Scholar, which is known to be the most comprehensive source [17] among the other alternatives like Microsoft Academic, Scopus, and Web of Science.

We composed two keyword sets: Set 1 comprises *blockchain, and smart contract*, while Set 2 consists of the keywords *testing, verification, and validation*. Then, we created search strings by combining these sets with logical expressions, *AND* and *OR*. As a result, we used search strings listed in Table 1 to find the primary studies. We had to adapt these strings according to the online database website search specification formats since each of them has various search query formulation options and search fields. Therefore, we had to use various key combinations. For instance, we had to use two different search strings and combine the results for these to employ ScienceDirect and Google Scholar.

4.2 The review process

Figure 2 depicts the review process, which was executed between March and April 2023. The review protocol and the selection criteria were discussed among all the authors of this paper. The primary study selection was made by the first author and it was reviewed by the second author.

Table 1 Search strings defined for each online database

Database	Search string
IEEE Xplore	(“Document Title“:blockchain AND “Document Title”:test*) OR (“Document Title“:blockchain AND “Document Title”:verification) OR (“Document Title“:blockchain AND “Document Title”:validation) OR (“Document Title“:smart contract AND “Document Title”:test*) OR (“Document Title“:smart contract AND “Document Title”:verification) OR (“Document Title“:smart contract AND “Document Title”:validation)
ACM Digital Library	(blockchain AND test) OR (blockchain AND verification) OR (blockchain AND validation) OR (smart contract AND test) OR (smart contract AND verification) OR (smart contract AND validation)
ScienceDirect	String 1: (blockchain AND test) OR (blockchain AND verification) OR (blockchain AND validation) OR (smart contract AND test) String 2: (smart contract AND verification) OR (smart contract AND validation)
Google Scholar	String 1: allintitle: Test OR Validation OR Verification “smart contract“ String 2: allintitle: Blockchain Test OR Validation OR Verification

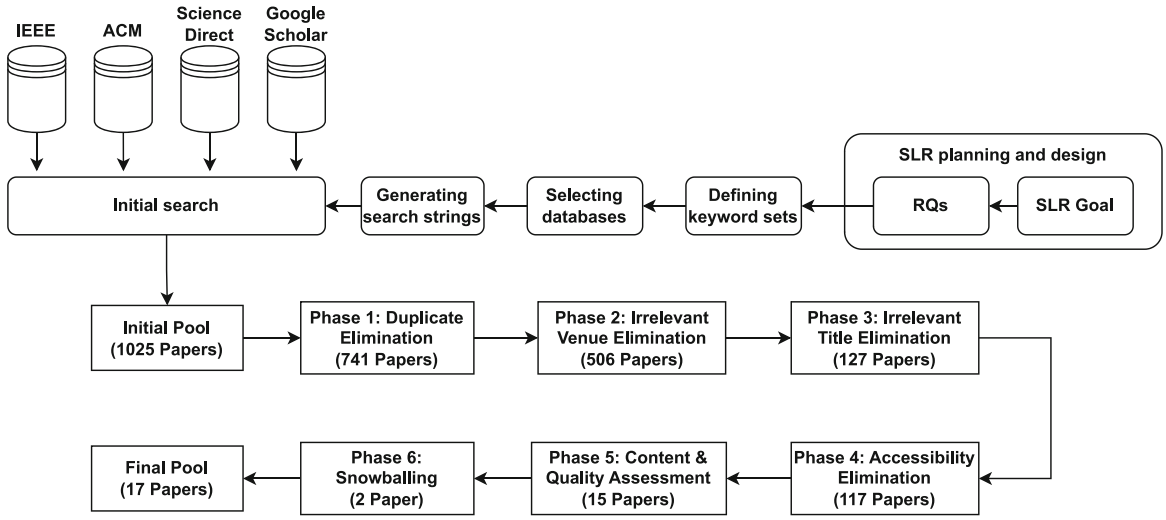

Fig. 2 The review process

In this study, we determined seven main phases as listed in the following.

- Phase 0. Full List: Four different searches were executed in the online databases with relevant search strings listed in Table 1. The search was conducted to cover all the publications between the years 2012 and 2022. A total of 1025 papers were found in the search. 5 publishers that are associated with the highest number of primary studies found in online databases are shown in Table 2.
- Phase 1. Duplicate Elimination: Since Google Scholar online database returns results from different publishers such as IEEE, ACM, Springer, ScienceDirect, Elsevier, etc., there were duplicates in 1025 papers collected from four different online databases in Phase 0. In this phase, we eliminated 284 duplicate papers from the full list. As a result, 741 papers remained after Phase 1 (See Fig. 2).
- Phase 2. Irrelevant Venue Elimination: The papers were evaluated according to the conferences in which they were presented. We eliminated 235 papers published in non-refereed conferences or those with a lack of publisher information. Following this phase, only 506 papers remained.
- Phase 3. Irrelevant Title Elimination: We evaluated the papers by reading their titles. A paper whose title indicates that it is on a different subject than our focus area has been eliminated. After eliminating papers with irrelevant titles, 127 papers remained.

Table 2 The number of primary studies found in online databases and 5 publishers that correspond to the highest number of publications found

Database	The number of papers
IEEE Xplore	252
ACM Digital Library	120
ScienceDirect	40
Google Scholar	613
Total	1025
Publisher	
IEEE	423
ACM	121
Springer	82
ScienceDirect	37
Elsevier	36

- Phase 4. Accessibility elimination: We eliminated papers that are not in English and whose full text is not available online. 10 papers were eliminated in this phase.
- Phase 5. Content & Quality Assessment (QA): We evaluated the papers based on the relevance of their content and quality. The use of a checklist is common for QA; however, there is no commonly agreed list of criteria and checklist items or any commonly accepted standard for conducting QA [18]. The relevant criteria depend on the type and the characteristics of the studies being reviewed. We compiled and applied the checklist shared in Table 3, which is based on a recent tertiary study [18] that systematically reviews commonly used QA instruments. We incorporated the most commonly used checklist items to be as inclusive as possible. We included only those papers that satisfy all the items in this list (See Section 8). 15 papers left after this phase.
- Phase 6. Snowballing: We performed snowballing [19] on the papers already included in our list after Phase 5. Hereby, we first applied backward snowballing, where we examined the references cited within the selected papers to discover additional relevant studies. Then, we applied forward snowballing, where we examined papers that cite the selected papers to ensure that relevant studies were not missed in our search. We used Google Scholar to keep track of the citations. The final pool, extended by two primary studies after this phase, consisted of 17 papers (See Table 4).

Table 3 Content and quality checklist

No.	Question
1.	Is there a statement of the aims of the research?
2.	Is the context of the study clearly described?
3.	Does the study focus on testing blockchain systems?
4.	Are the methods, tools, and techniques appropriate to address the aims of the research?
5.	Do the researchers discuss any limitations of their study?
6.	Does the study provide an experimental evaluation of the proposed technique?
7.	Is the experimental setup fully described?
8.	Is there a clear description of the results?
9.	Does the empirical evidence support the findings?
10.	Does the study introduce a new method, technique and tool?

We present and discuss the results regarding the final pool of primary studies in the following section. We also categorize these studies to identify current trends and research gaps.

5 Results and categorization

After the completion of all 6 phases, we finalized the pool with 17 papers as shown in Table 5. The list of all the primary studies and the list of selected and categorized papers throughout the successive phases of the review protocol is available online (See Sect. 8).

After a comprehensive review of 17 research papers, we identified distinct features and organized them into a structured table (see Table 6). In cases where the papers under review shared a particular feature such as permissionless blockchain, we omitted the inclusion of that feature in our table to avoid redundancy.

We observe that ten of the listed studies support test case generation, and eight of them are introduced for evaluating the effectiveness of a given test suite. Four of the studies support both test case generation and test evaluation. [27] also provides a Web-based interface together with four other studies. All the tools and techniques introduced by primary studies work on permissionless blockchains by default. As an exception, P6 supports both permissioned and permissionless blockchains.

In the following, we first categorize the final pool of primary studies according to the testing method used. Then, we categorize them with respect to the quality attributes being tested. The categorization of papers was performed by the first author in several iterations and results were reviewed by all the authors after each iteration.

Table 4 The selection process through multiple phases

Phase no.	Phase name	The number of remaining papers
0.	Full list	1025
1.	Duplicate Elimination	741
2.	Venue Elimination	506
3.	Irrelevant Title Elimination	127
4.	English & Full-Text Elimination	117
5.	Content & Quality Assessment	15
6.	Snowballing	17

Table 5 The final list of primary studies

References	ID	Title	Publisher	Year
[7]	p1	A Fuzz Testing Service for Assuring Smart Contracts	IEEE	2019
[20]	p2	ADF-GA: Data Flow Criterion Based Test Case Generation for Ethereum Smart Contracts	ACM	2020
[21]	p3	Automated Generation of Test Cases for Smart Contract Security Analyzers	IEEE	2020
[22]	p4	Data Flow Reduction Based Test Case Generation for Smart Contracts	IEEE	2022
[2]	p5	Deviant: A Mutation Testing Tool for Solidity Smart Contracts	IEEE	2019
[8]	p6	ModCon: a model-based testing platform For smart contracts	ACM	2020
[23]	p7	MuSC: A Tool for Mutation Testing of Ethereum Smart Contract	IEEE	2019
[24]	p8	RegularMutator: A Mutation Testing Tool For Solidity Smart Contracts	ScienceDirect	2020
[25]	p9	SolAnalyser: A Framework for Analysing and Testing Smart Contracts	IEEE	2019
[26]	p10	SuMo: A mutation testing approach and tool for The Ethereum blockchain	ScienceDirect	2022
[27]	p11	SynTest-Solidity: Automated Test Case Generation and Fuzzing for Smart Contracts	IEEE	2022
[28]	p12	Testing Smart Contracts Gets Smarter	IEEE	2020
[29]	p13	TestSmart: A Tool for Automated Generation Of Effective Test Cases for Smart Contracts	IEEE	2021
[30]	p14	SMARTGIFT: Learning to Generate Practical Inputs for Testing Smart Contracts	IEEE	2021
[31]	p15	PlaTIBART: a Platform for Transactive IoT Blockchain Applications with Repeatable Testing	ACM	2017
[32]	p16	GasFuzzer: Fuzzing Ethereum Smart Contract Binaries to Expose Gas-Oriented Exception Security Vulnerabilities	IEEE	2020
[33]	p17	sFuzz: an efficient adaptive fuzzer for solidity Smart contracts	ACM	2020

5.1 Categorization according to the testing method

This SM's initial findings shed light on the prevalence of different testing methods in blockchain and software testing. The final pool of 17 papers uses four testing methods as shown in Fig. 3.

In particular, mutation testing emerged as one of the most prominent methods, with six papers focusing on its application in the evaluation and verification of blockchain

Table 6 The features of primary studies

References	ID	Test case generation	Test evaluation	Web based	Permissioned blockchain
[7]	p1	✓		✓	
[20]	p2	✓			
[21]	p3	✓			
[22]	p4	✓			
[2]	p5		✓		
[8]	p6	✓		✓	✓
[23]	p7				
[24]	p8				
[25]	p9		✓		
[26]	p10		✓	✓	
[27]	p11	✓	✓	✓	
[28]	p12				
[29]	p13	✓	✓		
[30]	p14	✓			
[31]	p15	✓	✓		
[32]	p16				
[33]	p17	✓	✓		

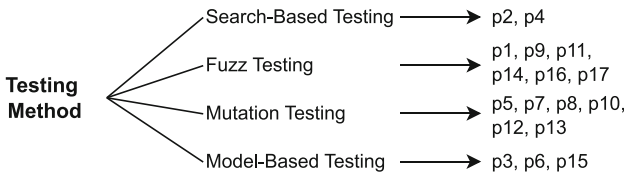

Fig. 3 Testing methods employed by primary studies

systems and software. This illustrates the importance of mutation testing for assessing the fault-resilience and robustness of blockchain systems and aligning them with the decentralized and immutable nature of blockchain transactions. Fuzz testing has also garnered the same attention, with six papers emphasizing its effectiveness in detecting vulnerabilities and ensuring the security of blockchain-based applications. There are three papers dedicated to model-based testing and the other two showcases search-based testing.

The distribution of the studies across testing methods in the final pool suggests varying levels of focus and interest in different aspects of blockchain and software testing. The findings showed that 35.29% of the studies used mutation testing, 17.64% employed model-based testing, 11.76% employed search-based testing, and 35.29% applied fuzz testing. There is a growing concern about security vulnerabilities in blockchain applications, and a dominance of mutation testing and fuzz testing as a means for detecting these vulnerabilities. Additionally, the inclusion of search-based testing and model-based testing papers demonstrates the exploration of new approaches to enhance the testing process in this domain. However, the relative scarcity of papers on search-based and model-based testing compared to mutation and fuzz testing may suggest potential areas for further research. Future studies could delve deeper into the applicability and effectiveness of these less-explored methods in blockchain-based systems and software. Hybrid approaches can also be effective. Indeed, although the number of studies based on search-based testing seems to be low, this technique is actually employed together with fuzz testing and mutation testing in a few studies to improve their effectiveness as discussed further in Sect. 6.

Genetic algorithms in search-based testing have been considered in two approaches, namely ADF-GA [20] and TCG-Re [22]. Both approaches perform control flow and data flow analysis on SCs. The values of input parameters used for tests are encoded on chromosomes. ADF-GA locates *require statements* (i.e., assertions) within the code and aims at covering the execution of these statements. TCG-Re aims at minimizing the number of generated test cases while improving the coverage of these test cases for def-use pairs.

In the final pool, several notable studies employ fuzz testing for vulnerability discovery, using random test generation as the core technique. As part of SynTest-Solidity [27], a genetic algorithm is used to evolve a set of initially randomly generated test cases, ensuring that they meet predefined criteria, such as function, line, and branch coverage. This process involves extracting objectives from the SC, feeding these into the search algorithm, and evaluating the generated test cases using Truffle [34] and

Ganache [35]. SMARTGIFT [30] focuses on improving the generation of suitable inputs for uncovering vulnerabilities. The proposed tool generates practical inputs for testing SCs by learning from the transaction records of real-world SCs. Other studies that employ fuzz testing [7, 25] demonstrate the effectiveness of using random inputs to uncover software weaknesses, showcasing the value of fuzz testing as an effective approach to security analysis. Indeed, this approach has gained significant recognition in the field due to its ability to identify vulnerabilities that may otherwise have remained undetected. GasFuzzer [32] is a tool that helps find security vulnerabilities in SCs by manipulating gas allowance. It works by first creating a pool of seed transactions and then mutating them with two strategies: gas-greedy and gas-leveling. The gas-greedy strategy favors transactions with higher gas consumption, while the gas-leveling strategy tries to level out the gas consumption of all transactions. GasFuzzer has been deployed in FUSE, an online fuzz testing service for Ethereum smart contracts. In [33], sFuzz is an adaptive fuzzer for SCs on the Ethereum platform. sFuzz works by configuring a blockchain network, deploying the smart contract, and generating transactions to call its functions. It monitors the execution of these transactions to collect feedback, such as branch coverage and potential vulnerabilities. Based on this feedback, sFuzz selects test cases for mutation and generation of the next round of test cases. This process repeats until a timeout occurs. It is significantly faster and more efficient than existing fuzzers like ContractFuzzer and Echidna. It is also more effective in achieving high code coverage and discovering vulnerabilities. Unlike symbolic execution engines, sFuzz is a fuzzer and can be combined with them for hybrid fuzzing.

Mutation testing has been conducted in several studies [2, 23, 24, 26, 28, 29]. These studies focus on mutation generation, involving the creation of artificial faults. They have concentrated on test suite execution and evaluation, wherein a collection of test cases is executed against both the original and the mutated code. The test suite's effectiveness is determined by its ability to identify the introduced mutations.

In [2], the proposed tool automatically generates mutants for a given Solidity project and runs all mutants against the given tests to evaluate their effectiveness. In [23], for each SC Under Test (SCUT), the proposed tool first transforms its source files to an Abstract Syntax Tree (AST) model and then performs mutant generation on them. In [24], the authors identified the most prevalent errors. From this set, they specifically chose errors that could be expressed using mutation operators in regular expressions. These selected errors were then considered for replacement in the source code. In the SuMo tool [26], novel mutation operators are introduced with a specific focus on the overloading mechanism, an aspect that previous related studies did not address. Furthermore, SuMo brings forth a set of additional mutation operators designed for various aspects of Solidity SCs. These operators encompass the contract constructor, function modifiers, cryptographic global functions, the SafeMath library, global blockchain variables, enums, return values, and explicit conversions. In [28], a comprehensive list of known bugs in Solidity SCs was analyzed, and ten classes of mutation operators were proposed. The authors report that these operators reproduced actual faults in 10 out of 15 famous buggy SCs. TestSmart [29] incorporates three key modules: one for generating test suites, another for creating mutants, and a third for selecting test cases based on these mutants. Manticore, a robust tool, is employed for test suite generation. An extended version of Universal Mutator, encompassing mutation operators designed for Solidity SCs, is utilized for mutant generation. Additionally, they leverage the Manticore API to analyze test cases compared to mutants.

Three approaches in the model-based testing domain have emerged as promising strategies. The first approach [21], introduces an automated method to evaluate contract analyzers of SCs by generating a diverse set of test cases. The TestBreeder tool comprises two main phases: test set generation and analysis assessment. Initially, it creates test templates for each vulnerability trigger, incorporating code seeds to populate empty slots. This generates test cases covering various complexity levels for a vulnerability trigger. In the second phase, the dispatcher evaluates contract analyzers by sequentially selecting test cases of increasing complexity levels and analyzing test cases one by one within the same vulnerability category. The second approach to model-based testing [8], revolves around the utilization of user-specified models to define test oracles, guide test generation, and measure test adequacy. Incorporating user expertise and domain knowledge enables targeted and reliable test cases. The third approach [31], PlaTIBART is a platform that helps developers build, deploy, and test transactive IoT blockchain applications (ITBAs) reliably and repeatably. It provides tools and techniques for the entire lifecycle of ITBAs, including development, deployment, execution, management, and testing. This ensures consistent and reliable testing through controlled environments, handles increasing numbers of clients with predictable performance, and encourages the adoption of ITBAs by demonstrating their reliable testing and deployment.

The categorization of the 17 papers in the final pool based on testing method and year, as shown in Fig. 4 provides an insight into the temporal progression of research in the field of blockchain and software testing. In 2017, there was a single study which is focused on model-

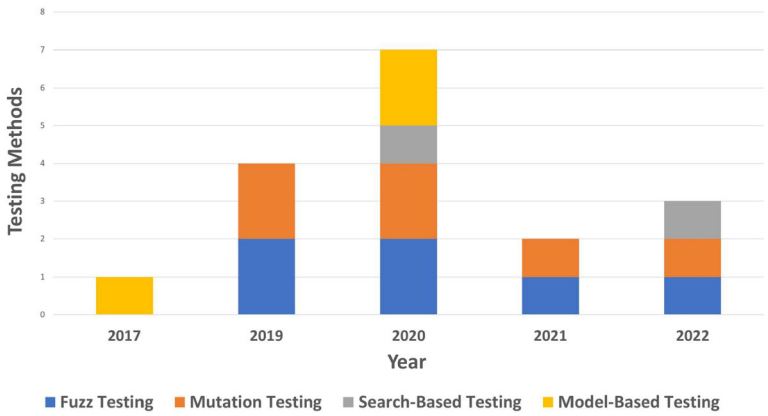

Fig. 4 The distribution of primary studies and the corresponding testing methods studied over the years

based testing. In 2019, the focus was on fuzz testing and mutation testing, with two papers dedicated to each method. This indicates an early exploration of these testing approaches in blockchain applications. The research landscape changed as we moved into 2020. Mutation testing and fuzz testing continued to receive attention with two papers each, demonstrating their ongoing relevance to evaluating blockchain systems. Additionally, model-based testing emerged with two papers and search-based testing emerged with one paper, signifying a growing interest in innovative techniques for testing blockchain applications.

Following the developments in previous years, in the year 2021 the focus was on fuzz testing and mutation testing, with one paper dedicated to each method. The other testing methods were not considered by any other publication during this year, indicating a potential research gap. In 2022, the publications on various testing methods were more balanced. Fuzz testing, mutation testing, and search-based testing are each studied in one paper, highlighting ongoing interest in these methods. However, model-based testing has not attracted researchers' attention for the last couple of years, suggesting a potential area for further exploration or an acknowledgment of limited applicability.

5.2 Categorization according to the quality factor being tested

Figure 5 shows a bubble plot that visually depicts the concentration of testing methods and quality factors under consideration. The categorization of the 17 papers in Fig. 5 provides an insight into the testing methods employed and the quality factors being tested in the context of blockchain and software systems. Regarding functionality, search-based testing, and fuzz testing highlight the applicability to assessing the functional aspects of these systems, with two papers each. Mutation testing emerges as a prominent method, representing the biggest bubble, with four papers dedicated to exploring its effectiveness in evaluating functionality. Model-based testing also contributes to assessing functional correctness, with one paper.

Shifting the focus to security, fuzz testing stands out as a prominent method. Four papers examine its effectiveness in detecting security vulnerabilities and ensuring the robustness of blockchain and software systems. Two papers examined the impact of mutation testing and one paper examined the impact of model-based testing on security evaluation. There are, however, no papers specifically addressing security in the final pool in search-based testing, indicating an unexplored area in state-of-the-art research. These findings highlight the importance of fuzz testing, and mutation testing in security assessment. This also suggests potential opportunities for exploring the applicability of search-based testing in enhancing the security of these systems.

Model-based testing is discussed in a paper that explores its impact on performance evaluation. Interestingly, no papers in the final pool specifically address performance in relation to search-based, fuzz, and mutation testing. Hence, there is a research gap on the application of search-based, fuzz, and mutation testing in assessing the performance of blockchain systems as well. Overall, the results provide valuable insights into the testing methods employed and the gaps in research concerning the functionality, security, and performance aspects of these systems.

We further elaborate on existing techniques, research gaps, and potential future trends in the following.

6 Discussion

In this section, we elaborate on each type of testing technique, representative studies that stand out, their advantages and limitations, hybrid approaches, and opportunities for further exploration considering the research gaps. We first discuss the main findings for each testing type. Then, we compare various techniques and discuss (potential) synergies among them.

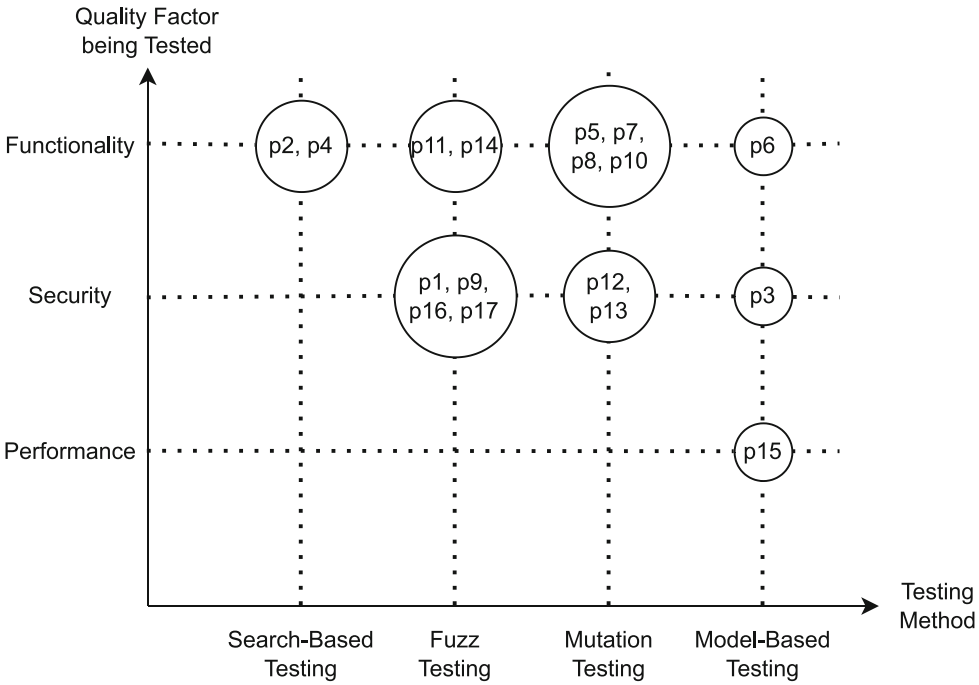

Fig. 5 The categorization of primary studies in two dimensions: the employed testing technique and the evaluated quality attribute

Fuzz Testing (FT): FT techniques and tools like sFuzz [33] are efficient in general and applicable for cases where symbolic execution cannot scale due to the limitations of the underlying constraint solvers. On the downside, recall can be low, and important vulnerabilities can be missed. Empirical results show that the number of true positives can be reduced up to 50% in comparison with symbolic execution tools like Oyente [36]. Feedback-guided fuzzing is used to increase test coverage as much as possible. Another issue is the lack of standardized benchmark datasets [33] and realistic test environments for evaluating FT. Simulation of a real-world blockchain environment can replicate the conditions found in live blockchain networks, such as network delays, consensus mechanisms, and transaction ordering.

The use of large language models is shown to be promising for fuzzers. SMARTGIFT [30] relies on such a model that is trained with real-world transaction records. In comparison with baseline fuzzing, SMARTGIFT effectively generates test inputs for approximately 77% of SC functions. Future efforts in this line of research could focus on expanding datasets, addressing decoding challenges, and retraining models concerning the adaptability to a variety of SC languages and resilience to emerging security threats [30].

Search-Based Testing (SBT): These testing approaches mainly utilize genetic algorithms for dynamic test case generation for SCs. In SBT, there is a need for refining selection and mutation operations, exploring enhanced fitness functions, and addressing execution environment constraints to improve its performance. These considerations relate to broader challenges in the field of SBT for SCs, where the optimization of improved mutation operations and the evaluation of fitness are among the most pressing issues.

Model-Based Testing (MBT): As a result of the dynamic nature of SC development, test suites must be constantly updated along with contract analyzers. In order to keep pace with the evolving complexity of SCs and to identify and address vulnerabilities efficiently, MBT approaches with automated tools become increasingly important. As an example, TestBreeder [21] employs templates to automatically build a test suite by exploiting various code elements of vulnerabilities, structures, and bugs. PlaTIBART [31] uses the specification of a distributed system that is based on the actor model to set up the corresponding system, and get it up and running. The system is simulated while the expected set of operations like the creation and integration of clients, and the execution of pre-defined scenarios are verified.

The effort and error-proneness related to the manual specification of models are the most forthcoming limitations for this approach and for MBT in general. The use of formal verification to check the consistency of models and tool support for model construction is proposed as future work directions to address these limitations [8, 31].

Mutation Testing (MT): MT tools like Deviant [2], MuSC [23], and SuMo [26] offer novel mutation operators that can be used for simulating various types of Solidity faults. Empirical evaluations show that achieving high mutation scores is challenging even when test suites meet the statement and branch coverage criteria [2, 24, 26]. The

majority of faults can be left undetected despite high coverage rates. Tests generally fail to detect mutants generated by mutation operators that affect the visibility of functions, state variables, modifiers, and exception handling [2, 26]. These mutants can mimic issues related to reentrancy, delegated calls, arithmetic errors, and time stamp-dependent faults [28].

Scalability and efficiency are the major concerns for MT. Mutants should be evaluated for their ability to expose faults in SCs to ensure that the costs of MT are paid off. The average time to generate mutants is short (e.g., a few seconds), but the execution time can be very long [23]. It can take up to 50 h to test 871 mutants, where 736 of them did not even compile [24]. Moreover, not all surviving mutants contribute to quality improvement or turn out to be useful. One needs to manually analyze surviving mutants and this manual analysis effort further increases the cost.

Synergies: There are also a number of hybrid approaches, where a type of technique is used to support and improve the other. SynTest-Solidity [27] is an example of the use of FT and SBT together. It is an FT tool that automates the generation of Solidity test cases by using genetic algorithms. Experimental results suggest that the use of SBT significantly increases the coverage of FT. Another example of a hybrid approach is TCG-Re [22] which brings MBT and SBT together. It employs data flow models for test case generation. However, it also uses a genetic algorithm, which helps to increase the coverage of definition-use pairs in the data flow. An interesting observation is that search-based techniques are central in all these hybrid approaches as depicted in Fig. 6.

The only intersection, for which we could not find a primary study for testing blockchain systems is the intersection of MT and SBT. There is an interesting research gap for exploiting promising synergies in this intersection. Search-based mutation testing is well-explored in other domains, where search-based techniques are utilized for optimizing test data generation, operator selection, and mutant generation [37]. Therefore it can be used for improving MT in several aspects.

Our literature review is subject to a number of validity threats, which are discussed in the following.

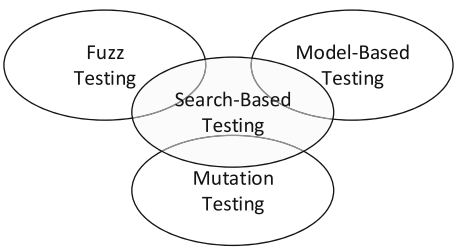

Fig. 6 The common use of search-based techniques, intersecting with other techniques in hybrid approaches

7 Threats to validity

Some of the relevant studies might have been missed as an *external validity* threat due to the time period when the search was performed as well as the employed search strings. We had to adapt the search string for each database, each of which has various search query formulation options and search fields. Hence, we might have missed a related study as a result of inevitable changes made to the search query. We discussed the query design among the authors and employed multiple databases to mitigate this threat. In particular, we used Google Scholar to attain an extensive coverage of the search space.

A major contribution of our review is the categorization of primary studies along two dimensions. However, this categorization is subject to an *internal validity* threat that is related to the reliability and sufficiency of the considered categories for analyzing and classifying all the primary studies. To mitigate this threat, we analyzed the primary study in several iterations and reviewed the results with all the authors after each iteration. Newly emerged categories in each iteration were introduced to the taxonomy and the necessary adaptations were made respectively.

There exists a *construct validity* threat regarding the selection of primary studies, which might be subject to bias. We tried to mitigate this threat by applying a QA. The selection criteria were discussed among all the authors to ensure their quality. The selection was made by the first author and reviewed by the second author.

We applied a systematic review protocol and shared a replication package to facilitate reproducibility. Selection of primary studies were performed by the first two authors, whereas the review process as well as the categorization of these studies were discussed among all the authors to mitigate *conclusion validity* threats.

8 Conclusion

This systematic mapping study provides a comprehensive review of software testing methods for blockchain systems. Our analysis focused on two aspects in that respect: employed state-of-the-art testing techniques and quality factors addressed in the existing literature. We reduced a large pool of primary studies to a final selection of 17 papers. The majority of these papers focus on ensuring functional correctness and detection of security vulnerabilities. Mostly either fuzz testing or mutation testing is employed to this aim. The adoption of model-based testing is rare. It has been used for performance testing, which have not attracted much attention in general. The adoption of search-based testing on its own is not very common

either. However, it is usually employed together with fuzz testing and mutation testing to improve their efficiency. Generally, the use and integration of multiple testing techniques stand out as a viable research direction.

There are certain limitations of the existing techniques that are emphasized in primary studies. Low recall and lack of adaptability to changing blockchain platforms and languages are noted for model-based testing techniques. On the other hand, performance and scalability issues stand out for search-based techniques, fuzz testing, and mutation testing. Finally, there is a need for larger datasets to support empirical evaluations.

Author Contributions All authors contributed to the study's conception and design. Material preparation, data collection, and analysis were performed by Anıl Elakaş. The review protocol and the selection criteria were discussed among all the authors. The primary study selection is made by Anıl Elakaş and it was reviewed by Hasan Sözer. The categorization of papers is performed in several iterations and results are reviewed by all the authors after each iteration. The discussion section was written by Ilgın Şafak. The first draft of the manuscript was written by Anıl Elakaş and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK). The work is supported by the Scientific and Research Council of Turkey under Grant No. 119C111.

Data Availability The list of all the primary studies and the list of selected and categorized papers throughout the successive phases of the review protocol are available as part of our replication package, which is shared at https://github.com/anilelakas/blockchain-softwareTesting.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Guo, H., Yu, X.: A survey on blockchain technology and its security. Blockchain: Res. Appl. **3**, 544–545 (2022). https://doi.org/10.1016/j.bcra.2022.100067
2. Chapman, P., Xu, D., Deng, L., Xiong, Y.: Deviant: A mutation testing tool for solidity smart contracts. IEEE Int. Conf. Blockchain (2019). https://doi.org/10.1109/Blockchain.2019.00050
3. Arsat, N., Bakar, N.S.A.A., Yahya, N.: Testing in blockchain-based systems: a systematic review. Int. Conf. Cyber IT Serv. Manag. (CITSM) (2022). https://doi.org/10.1109/CITSM56380.2022.9935846
4. Kitchenham, B., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering. University and Durham University, Keele (2007)
5. Benabbou, C., Gurcan, O.: A survey of verification, validation and testing solutions for smart contracts. In: 2021 Third International Conference on Blockchain Computing and Applications (BCCA), pp. 57–64 (2021). https://doi.org/10.1109/BCCA53669.2021.9657040
6. Sujeetha, R., Deiva Preetha, C.A.S.: A literature survey on smart contract testing and analysis for smart contract based blockchain application development. In: 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), pp. 378–385 (2021). https://doi.org/10.1109/ICOSEC51865.2021.9591750
7. Mei, X., Ashraf, I., Jiang, B., Chan, W.K.: A fuzz testing service for assuring smart contracts. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security, pp. 544–545 (2019). https://doi.org/10.1109/QRS-C.2019.00116
8. Liu, Y., Li, Y., Lin, S.-W., Yan, Q.: Modcon: A model-based testing platform for smart contracts. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2020, pp. 1601–1605. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3368089.3417939
9. Smetanin, S., Ometov, A., Komarov, M., Masek, P., Koucheryavy, Y.: Blockchain evaluation approaches: state-of-the-art and future perspective. Sensors **20**(12), 3358 (2020)
10. Tolmach, P., Li, Y., Lin, S.-W., Liu, Y., Li, Z.: A survey of smart contract formal specification and verification. ACM Comput. Surv. (2021). https://doi.org/10.1145/3464421
11. Garfatta, I., Klai, K., Gaaloul, W., Graiet, M.: A survey on formal verification for solidity smart contracts. In: Proceedings of the 2021 Australasian Computer Science Week Multiconference. ACSW '21. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3437378.3437879
12. Liu, J., Liu, Z.: A survey on security verification of blockchain smart contracts. IEEE Access **7**, 77894–77904 (2019). https://doi.org/10.1109/ACCESS.2019.2921624
13. Krichen, M., Lahami, M., Al-Haija, Q.A.: Formal methods for the verification of smart contracts: a review. In: 2022 15th International Conference on Security of Information and Networks (SIN), pp. 01–08 (2022). https://doi.org/10.1109/SIN56466.2022.9970534
14. Ilgi, G.S., Kayali, D., Olawale, P., Demir Erdem, B., Dimililer, K., Kirsal-Ever, Y.: Formal verification for security technologies in the blockchain with artificial intelligence: A survey. In: 2022 Innovations in Intelligent Systems and Applications Conference (ASYU), pp. 1–6 (2022). https://doi.org/10.1109/ASYU56188.2022.9925532
15. Murray, Y., Anisi, D.A.: Survey of formal verification methods for smart contracts on blockchain. In: 2019 10th IFIP International Conference on New Technologies, Mobility and Security

(NTMS), pp. 1–6 (2019). https://doi.org/10.1109/NTMS.2019.8763832
16. Imperius, N.P., Alahmar, A.D.: Systematic mapping of testing smart contracts for blockchain applications. IEEE Access **10**, 112845–112857 (2022)
17. Martín-Martín, A., Thelwall, M., Orduna-Malea, E., López-Cózar, E.D.: Google scholar, microsoft academic, scopus, dimensions, web of science, and OpenCitations' COCI: a multidisciplinary comparison of coverage via citations. Scientometrics **126**, 871–906 (2021)
18. Yang, L., Zhang, H., Shen, H., Huang, X., Zhou, X., Rong, G., Shao, D.: Quality assessment in systematic literature reviews: a software engineering perspective. Inform. Softw. Technol. **130**, 106397 (2021)
19. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10 (2014)
20. Zhang, P., Yu, J., Ji, S.: Adf-ga: Data flow criterion based test case generation for ethereum smart contracts. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. ICSEW'20, pp. 754–761. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3387940.3391499
21. Kim, K.B., Lee, J.: Automated generation of test cases for smart contract security analyzers. IEEE Access **8**, 209377–209392 (2020). https://doi.org/10.1109/ACCESS.2020.3039990
22. Ji, S., Zhu, S., Zhang, P., Dong, H.: Data flow reduction based test case generation for smart contracts. In: 2022 29th Asia-Pacific Software Engineering Conference (APSEC), pp. 149–158 (2022). https://doi.org/10.1109/APSEC57359.2022.00027
23. Li, Z., Wu, H., Xu, J., Wang, X., Zhang, L., Chen, Z.: Musc: A tool for mutation testing of ethereum smart contract. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1198–1201 (2019). https://doi.org/10.1109/ASE.2019.00136
24. Ivanova, Y., Khritankov, A.: Regularmutator: a mutation testing tool for solidity smart contracts. Procedia Comput. Sci. **178**, 75–83 (2020).
25. Akca, S., Rajan, A., Peng, C.: Solanalyser: A framework for analysing and testing smart contracts. In: 2019 26th Asia-Pacific Software Engineering Conference (APSEC), pp. 482–489 (2019). https://doi.org/10.1109/APSEC48747.2019.00071
26. Barboni, M., Morichetta, A., Polini, A.: Sumo: a mutation testing approach and tool for the ethereum blockchain. J. Syst. Softw. **193**, 111445 (2022). https://doi.org/10.1016/j.jss.2022.111445
27. Olsthoorn, M., Stallenberg, D., Van Deursen, A., Panichella, A.: Syntest-solidity: Automated test case generation and fuzzing for smart contracts. In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 202–206 (2022). https://doi.org/10.1145/3510454.3516869
28. Andesta, E., Faghih, F., Fooladgar, M.: Testing smart contracts gets smarter. In: 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 405–412 (2020). https://doi.org/10.1109/ICCKE50421.2020.9303670
29. Fooladgar, M., Arefzadeh, A., Faghih, F.: Testsmart: A tool for automated generation of effective test cases for smart contracts. In: 2021 11th International Conference on Computer Engineering and Knowledge (ICCKE), pp. 476–481 (2021). https://doi.org/10.1109/ICCKE54056.2021.9721448
30. Zhou, T., Liu, K., Li, L., Liu, Z., Klein, J., Bissyandé, T.F.: Smartgift: Learning to generate practical inputs for testing smart contracts. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 23–34 (2021). https://doi.org/10.1109/ICSME52107.2021.00009
31. Walker, M., Dubey, A., Laszka, A., Schmidt, D.: Platibart: a platform for transactive iot blockchain applications with repeatable testing, pp. 17–22 (2017). https://doi.org/10.1145/3152141.3152392
32. Ashraf, I., Ma, X., Jiang, B., Chan, W.K.: Gasfuzzer: fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities. IEEE Access **8**, 99552–99564 (2020). https://doi.org/10.1109/ACCESS.2020.2995183
33. Nguyen, T.D., Pham, L.H., Sun, J., Lin, Y., Minh, Q.T.: sFuzz: an efficient adaptive fuzzer for solidity smart contracts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 778–788 (2020)
34. Truffle Suite. https://trufflesuite.com/. Accessed 31 Oct 2023
35. Ganache. https://trufflesuite.com/ganache/. Accessed 31 Oct 2023
36. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 254–269 (2016)
37. Silva, R.A., Senger de Souza, S.R., Lopes de Souza, P.S.: A systematic review on search based mutation testing. Inform. Softw. Technol. **81**, 19–35 (2017)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Anıl Elakaş is a Senior Software Engineer at Fibabanka's R&D Center in Istanbul, Turkey. He received his MSc in 2020 in Computer Science from Sabancı University. He is currently a PhD student in Computer Science department in Ozyegin University. His current research interests include Software Testing, Blockchain, Machine Learning.

Hasan Sözer received his B.Sc. and M.Sc. degrees in computer engineering from Bilkent University, Turkey, in 2002 and 2004, respectively. He received his Ph.D. degree in 2009 from the University of Twente, The Netherlands. From 2002 until 2005, he worked as a software engineer at Aselsan Inc. in Turkey. From 2009 until 2011, he worked as a post-doctoral researcher at the University of Twente. In 2011, he joined the department of computer science at Ozyegin University, where he is currently working as a full professor.

Ilgın Şafak is an Expert Researcher at Fibabanka's R&D Center in Istanbul, Turkey. She received her MSc and PhD degrees in 2006 and 2013 respectively, in Electrical and Electronics Engineering from Hacettepe University in Turkey. She worked as a Product Manager in Product Development and Innovation at Mastercard Payment Transaction Services Turkey in Istanbul, Turkey between 2013 and 2016. She worked as a Manager in Product Development and Innovation at Mastercard in Purchase, NY from 2016 until 2018. She worked as a Solution Engineer from 2019 until 2020 at HAVELSAN, Turkey. She has been working at Fibabanka since 2020. She published 23 articles and has filed 11 patent applications. Her research interests are in Distributed Ledger Technology, Artificial Intelligence/Machine Learning, the Internet of Things, and Cybersecurity.

Kübra Kalkan is a faculty member in the Computer Science Department of Özyeğin University in Turkey. She received B.Sc. and M.Sc. degrees from Sabancı University in 2009 and 2011 respectively and Ph.D. degree from Boğaziçi University in 2016. She was a researcher with various institutions, such as EPFL, Microsoft Redmond, and Northeastern University. She worked as Post-Doctoral Researcher in the University of Oxford after the PhD. Her current research interests include Blockchain, IoT, and P2P networks.