The Journal of Systems & Software 174 (2021) 110891

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges☆

Anna Vacca *, Andrea Di Sorbo, Corrado A. Visaggio, Gerardo Canfora

Department of Engineering, University of Sannio, Italy

ARTICLE INFO

Article history:
Received 21 August 2020
Received in revised form 21 November 2020
Accepted 22 December 2020
Available online 28 December 2020

Keywords:
Software engineering for blockchain technologies
Software quality
Software metrics
Empirical study
Ethereum
Smart contract

ABSTRACT

Blockchain platforms and languages for writing smart contracts are becoming increasingly popular. However, smart contracts and blockchain applications are developed through non-standard software life-cycles, in which, for instance, delivered applications can hardly be updated or bugs resolved by releasing a new version of the software. Therefore, this systematic literature review oriented to software engineering aims at highlighting current problems and possible solutions concerning smart contracts and blockchain applications development. In this paper, we analyze 96 articles (written from 2016 to 2020) presenting solutions to tackle software engineering-specific challenges related to the development, test, and security assessment of blockchain-oriented software. In particular, we review papers (that appeared in international journals and conferences) relating to six specific topics: smart contract testing, smart contract code analysis, smart contract metrics, smart contract security, Dapp performance, and blockchain applications. Beyond the systematic review of the techniques, tools, and approaches that have been proposed in the literature to address the issues posed by the development of blockchain-based software, for each of the six aforementioned topics, we identify open challenges that require further research.

1. Introduction

The Bitcoin whitepaper (Nakamoto, 2008) was published in 2008, and the related blockchain was started in 2009. Since then, blockchain technology has continued to show several advantages (e.g., decentralization, trust, immutability, transparency Golosova and Romanovs, 2018) and different areas of applicability. One of the most relevant applications of blockchain is smart contracts (i.e., self-executing contracts containing the terms of the agreement between the parts). Smart contracts allow trusted transactions and agreements to be carried out among different anonymous parties.

The result of a smart contract execution on the nodes of a blockchain corresponds to a change of status in the blockchain itself. Such a change of status is triggered by a transaction posted to the blockchain. Transactions are aggregated in blocks, and nodes must reach a consensus to add a new block to the blockchain. In particular, due to the absence of a central authority, blockchain technologies make use of consensus algorithms to validate and verify the transactions. To reach such consensus, well-known algorithms are typically used (e.g., the Proof of Work and the Proof of Stake algorithms Bach et al., 2018).

Blockchain applications and smart contracts can be used in various fields: from insurance refunds to financial transactions, from corporate operations to the traceability of goods and the protection of intellectual property. Thus, the number of companies using blockchain and smart contract applications continues to grow. However, for satisfying high performance, scalability, and security requirements, blockchain applications need to be well-designed and thoroughly tested. This is especially true if we consider that smart contracts are developed through non-standard software life-cycles, in which delivered applications can hardly be updated or bugs resolved by releasing a new version of the software (Destefanis et al., 2018). In general, the development of smart contracts is very different from traditional software development, raising new problems and challenges (Marchesi et al., 2020). For instance, on the one hand, (i) developers must ensure code security for smart contracts due to the immutability of blockchain and the sensitiveness of digital information often managed, on the other hand (ii) they must pay special attention to gas consumption, as the execution of smart contracts in blockchain platforms like Ethereum is implemented through the gas mechanism (Zou et al., 2019). It should be considered that the blockchain technology requires specific constraints and characteristics of the applications that will run on it, the product

☆ Editor: [BURAK TURHAN].

* Corresponding author.
E-mail addresses: avacca@unisannio.it (A. Vacca), disorbo@unisannio.it (A. Di Sorbo), visaggio@unisannio.it (C.A. Visaggio), canfora@unisannio.it (G. Canfora).

https://doi.org/10.1016/j.jss.2020.110891
0164-1212/

lifecycle and the development software process of smart contracts and blockchain-based applications. All these changes to the software product and process must be identified and studied in order to develop a comprehensive body of knowledge of blockchain based software engineering.

Although the increasing popularity of smart contracts and the research efforts that have been posed to tackle some of these issues, the approaches proposed are still far from being consolidated. For these reasons, it urges to identify the software engineering tools, techniques, best practices, as well as testing approaches, specially designed to address the novel features introduced by decentralized programming on the blockchain (Chakraborty et al., 2018). To fill this gap and have a clearer picture of the research efforts that have been carried out to improve the implementation, the security, and the reliability of this kind of applications, in this paper we perform a systematic review of the literature concerning the software engineering tools, techniques, and practices envisioned for dealing with the peculiarities posed by smart contracts and blockchain development. Moreover, we also try to identify possible future research directions and open issues that need to be addressed. In particular, we pose the following general research question:

What is the current state of the art in research and which are the main challenges faced in the development, testing, and quality assessment of blockchain-oriented software?

In particular, this research question aims at reviewing the specific methods, techniques, and tools proposed for improving the design, construction, testing, maintenance, and quality of smart contracts and decentralized apps. Additionally, our goal is to also identify the specific challenges of blockchain-oriented software engineering that are still open and need further research. To answer our research question, several aspects of software engineering have been considered, such as testing, security, and source code quality of blockchain-based applications.

During the life cycle of software, the testing phase is useful to verify if the software produced corresponds to the expected specifications and, above all, if it does not present bugs, errors, or other defects. Concerning software testing, we specifically analyze (i) approaches and frameworks for generating test cases (such as Zeus Kalra et al., 2018), (ii) approaches to identify code smells present in the source code (Chen et al., 2020b), and (iii) tools based on fuzz testing (such as Fuse Chan and Jiang, 2018, Contractfuzzer Jiang et al., 2018, Evmfuzz Fu et al., 2019 and ReGuard Liu et al., 2018).

Another aspect to consider in software engineering is the analysis of the source code with the aim of evaluating its quality. For this purpose, we discuss the existing solutions aimed at detecting problems in the code, (e.g., Smartcheck Tikhomirov et al., 2018, Slither Feist et al., 2019, KEVM Hildenbrandt et al., 2018). In particular, we analyze the proposed solutions to identify which of them is more useful for identifying specific problems.

Software metrics are useful measurement units for quantifying, measuring, and evaluating the different aspects of software development. Such metrics can be divided into different categories according to their use, including metrics related to software products, metrics related to software development processes, metrics related to software quality, Object-oriented metrics. We analyze the literature concerning the adoption/proposition of metrics to deal with issues specifically posed by blockchain-oriented development.

Another very important aspect is the security of the source code. Since the most popular and successful applications of smart contracts are tokens and decentralized exchanges (i.e., peer-to-peer trading of cryptocurrencies) moving money for tens of billions of dollars (Oliva et al., 2020), there is a need for improved security practices. Source code might contain critical security defects or vulnerabilities, which, if exploited, in the case of blockchains, smart contracts, and virtual currencies, can lead to financial losses. Thus, we consider both techniques and tools aimed at identifying fraudulent behaviors and security flaws in smart contracts (e.g., TEETHER Krupp and Rossow, 2018, MAIAN Nikolić et al., 2018, Oyente Luu et al., 2016).

We also discuss solutions aimed at measuring Dapp (decentralized application) performance. In particular, benchmarks for performing such evaluations are analyzed.

In addition, to better understand the specific applications that use the blockchain beyond cryptocurrency, we also consider the Dapps, which have been created by developers for being used in a plethora of different scenarios (e.g., social platforms, casino, and financial exchange, etc.). In particular, analyzing such applications, we are interested in better comprehending which specific problems are addressed through the adoption of the blockchain technology.

Results of our literature review highlight that most of the techniques investigated and tools/frameworks implemented have a focus on the Ethereum platform, with few exceptions as Bitcoin and Hyperledger. However, many of these techniques and tools only deal with specific aspects and issues of blockchain-oriented software engineering. For these reasons, further research is needed for more comprehensively targeting the different constraints and challenges posed by blockchain-based software development, testing, and quality assessment.

In summary, the contribution of the current paper is twofold:

- we provide the community with a common knowledge base enumerating the approaches for improving smart contract and blockchain application development that have been proposed so far; and
- we identify the specific blockchain software engineering challenges that are still open and need further research.

Paper structure. The paper is organized as follows. Section 2 compares existing literature review in the field of blockchain and our proposal regarding software engineering. Section 3 presents the research methodology used. Section 4 discusses the findings of our literature review, while Section 5 highlights future research directions and open issues that have emerged from the analysis of the collected papers. Section 6 deals with the main threats that could affect our findings and, finally, Section 7 concludes the paper.

2. Related work

The interest of the research community towards issues concerning applications based on the blockchain technology is increasingly growing. Unfortunately, a literature review systematically examining the specific aspects of Software Engineering is still lacking. In fact, the literature reviews realized till now regards mainly the application of the blockchain technology to specific domains such as security, biomedical, intellectual property.

Among the topics covered in the identified literature reviews, Taylor et al. (2019) deal with blockchain cyber security. By identifying peer-reviewed literature, they feature applications that have used blockchain security, such as Internet of Things (IoT), using Personally Identifier Information (PII), public key cryptography, machine visualization. In their study, they pose three research questions related to the latest applications that focus on security, how much blockchain is used to improve cyber security and what are the methods used. Regarding blockchains for Iot security, the analysis shows that blockchain technology is useful even if there are no guidelines on how to integrate it. Regarding

blockchains for AI data security, blockchain is useful to reduce the risks related to transaction and financial fraud.

Conoscenti et al. (2016) aimed at understanding how blockchain and peer-to-peer approaches can be used in applications with decentralized and intensive data, guaranteeing the user privacy. Based on the examined literature, the study found integrity problems (only pseudonymity can be guaranteed), anonymity and adaptability, which depend on the implementation of proof of work and miners.

Instead, Casino et al. (2019) analyzed the current state of blockchain technology and its applications. They propose a taxonomy to classify blockchain-based applications into finance, integrity verification, governance (including citizen and user services, public sector, voting), IoT, healthcare management, privacy and security, business and industry (which includes supply chain management and energy sector), education, data management, and finally miscellaneous. This analysis highlights limitations of blockchain technology and usability in different application domains.

While the previous study focuses on blockchain applications, Porru et al. (2017) highlight the state of the art of Blockchain-oriented software (BOS) development by identifying current problems and new directions to be taken. For this purpose, they examined around 1000 repositories on GitHub.[1] Then, they extracted information such as programming language, age, popularity, number of contributors. The most popular projects are Bitcoin and Ethereum, the average age of the projects is four years and there are several open issues. By comparing the software systems, the authors suggest increasing testing, collaborations, improving testing and debugging for a specific programming language and creating tools for smart contracts.

Drosatos and Kaldoudi (2019) carried out a review of the scientific literature related to blockchain applications in the biomedical field. The purpose is to identify for which biomedical problems blockchain technology is used, what is the maturity level of the respective approaches, which types of biomedical data are considered and which blockchain features and functionalities are used. The study shows that the field is still in its early stages. In fact, most of the studies are in the conceptual or architectural design phase. Research focuses mainly on health access integration, integrity and control records and relative patient data, while a smaller part of applications concern medical research, clinical trials, supply chain of medicines and medical insurance.

Wang et al. (2019a) investigated the blockchain applications in the field of intellectual property.

Crowdfunding sources are collected for new businesses and innovative projects by several supportive people using online platforms. Hartmann et al. (2019) provided a comparison between traditional crowdfunding and blockchain-based crowdfunding which is still at the beginning. The success factors characterizing traditional crowdfunding are different from the blockchain-based one. Understanding success factors is useful to facilitate potential investors to look for suitable projects to invest in. Regulatory frameworks should reinterpret the requirements to allow effective enforcement of regulations. For this, they analyzed existing publications on traditional and blockchain-based crowdfunding to better understand similarities and differences between the two.

It emerges a clear lack of a systematic literature review concerning the software engineering issues in the designing, development, maintenance, and testing of applications based on the blockchain technology, which motivates our analysis.

[1] https://github.com/

3. Research methodology

Our systematic literature review on blockchain based Software Engineering has been carried out following the guidelines of Kitchenham and Charters (2007). In this section we will introduce how the examined papers have been selected, prepared for the analysis and an overview of the produced research (who are the main authors, how the papers are distributed over time and on the types of publication).

3.1. Selection of primary study

To search for primary studies, we have identified the following keywords:

("Blockchain" OR "smart contract" OR "Ethereum")

and set as a filter the years from 2016 to 2020.

The platforms used for the search were Google Scholar,[2] IEEE Xplore Digital Library,[3] ACM Digital Library,[4] Science Direct.[5]

For each paper found, the title was analyzed, then abstract, introduction, and conclusions to determine whether it pertained to the established inclusion criteria. Then we went on to analyze the rest of each document to find significant contributions and open issues.

3.2. Inclusion and exclusion criteria

Papers published in scientific international journals and proceedings of international conferences were considered. All other kinds of work like books, technical reports or master theses were not included. Papers dealing with topics not belonging to the corpus of knowledge of Software Engineering have been excluded, like: mining algorithm, cryptography, ICO (Initial Coin Offer), and architectural issues of platforms enabling Distributed Ledgers. In addition, articles not written in English and articles not indexed were not taken into account.

3.3. Selection results

Considering the keywords, the following results have been obtained, divided by platform:

- Google Scholar: 130,000 results.
- IEEE Xplore Digital Library: 5810 results.
- ACM Digital Library: 33,187 results.
- Science Direct: 12,835 results.

Subsequently, the duplicated studies were removed. Then the papers concerning the exclusion criteria were removed. Finally, 96 papers remained on which analysis was conducted.

3.4. Quality assessment

In addition to the inclusion/exclusion criteria, the quality of the studies found was subsequently checked. The studies were divided into the following six categories:

- Smart contract testing: it includes the approaches to verify and validate diverse aspects of blockchain, the automatizing of the testing process.

[2] https://scholar.google.com/ - last access: 28th July 2020.
[3] https://ieeexplore.ieee.org/Xplore/home.jsp - last access: 28th July 2020.
[4] https://dl.acm.org/ - last access: 28th July 2020.
[5] https://www.sciencedirect.com/ - last access: 28th July 2020.

Table 1
Summary of articles by years.

Year	#Articles	%Article
2016	4	4.16%
2017	11	11.46%
2018	41	42.71%
2019	25	26.04%
2020	15	15.63%

- Smart contract code analysis: it comprehends frameworks, methods and tools for static analysis.
- Smart contract code metrics: it gathers measurement tools and metrics bunch for evaluating structural properties of smart contracts.
- Smart contract security: it includes the techniques and the approaches to enhance security of smart contracts.
- Dapp performance measurement: it comprehends the measurement of a Dapp efficiency, which is a critical issue for blockchain based applications.
- Blockchain Application: it includes papers investigating the application of the Blockchain technology to different domains.

3.5. Data extraction and analysis

For each paper that passed the quality assessment, we examined the following further aspects: the motivations for which it was written, which contributions it gave to research, which results it has achieved and which challenges it posed for the future.

Finally, a spreadsheet was filled in with the following information for each paper: (i) Article Name; (ii) Type of study; (iii) Classifications Study; (iv)item Keyword; (v) Case study; (vi) Experimentation; (vii) Type of experimentation; (viii) Qualitative/quantitative study; (ix) Experimental samples sizes; (x) Analysis method; (xi) Software sample/replication package availability; (xii) Number of citations over time; (xiii) Empirical evidence.

This information was useful to structure the relevant aspects of each paper necessary to our systematic literature review.

4. Results

The different studies have been classified into six categories. The first four concern smart contracts and consider the aspects of testing, security, analysis and metrics relating to the source code. The rest concerns Dapps and blockchain applications. Table 1 reports the distribution of analyzed articles by year, while in Table 2 the considered papers are divided by publication types. Based on the 96 papers considered, Table 3 indicates the 10 authors who authored (or co-authored) the largest number of them. Instead, considering the collected data (see Section 3.5), to provide a complete overview of the inspected articles, in Tables 4, 5, 6, and 7, we report information about the (i) tools implemented (see Table 4), as well as (ii) the empirical validations (see Table 5), (iii) the case studies (see Table 6), and (iv) the surveys (see Table 7) carried out. The fields in each table are sorted according to the technology considered. It is worth pointing out that in the blockchain landscape there is a high percentage of smart contracts that are very similar to each other or are multiple versions of the same contract (Kondo et al., 2020). For this reason, the findings reported in Table 4 should be taken with caution.

Table 2
Summary of articles by publication type.

Sites	#Articles	%Article
Conference	49	51.04%
Workshop	21	21.87%
Journal	18	18.75%
Preprint	8	8.34%

Table 3
Top 10 blockchain authors for software engineering.

Surname, Name	Affiliation	#Article
Bartoletti, Massimo	University of Cagliari, Italy	5
Marchesi, Michele	University of Cagliari, Italy	4
Tonelli, Roberto	University of Cagliari, Italy	4
Chen, Ting	Center for Cybersecurity, University of Electronic Science and Technology of China, China	4
Destefanis, Giuseppe	Brunel University London, England	4
Rocha, Henrique	Inria Lille - Nord Europe Villeneuve D'ascq, France	4
Chen, Jiachi	Monash University, Australia	3
Ducasse, Stéphane	Inria Lille - Nord Europe Villeneuve D'ascq, France	3
Li, Xiaoqi	Department of Computing, The Hong Kong Polytechnic University, China	3
Luo, Xiapu	Department of Computing, The Hong Kong Polytechnic University, China	3

4.1. Smart contract testing

Research in this area mainly focused on: test generation (Wang et al., 2019b; Chepurnoy and Rathee, 2018; Fu et al., 2019; Jiang et al., 2018; Liu et al., 2018; Chan and Jiang, 2018; Liu et al., 2018), contract defects identification (Chen et al., 2020b), and approaches to verify and validate diverse aspects of blockchain (Kalra et al., 2018; Ellul and Pace, 2018; Wu et al., 2020). In this section most of the literature uses Ethereum technology, only (Chepurnoy and Rathee, 2018) refers to Bitcoin technology.

Unlike traditional software, Ethereum Smart Contracts (ESCs) are gas-driven programs: the choices of developers are conditioned by the aim of keeping the levels of gas consumption under acceptable thresholds. For these reasons, it is necessary to have test suites that maximize effectiveness while minimizing the running costs. To this aim, Wang et al. (2019b) considered ESC test generation as a Pareto minimization problem, to minimize (i) uncovered branch coverage, (ii) time cost, and (iii) gas cost. The proposed approach demonstrates to reduce time and gas costs while maintaining the ability to cover branches. Instead, Zhang et al. (2020) propose a new approach to generating test case called ADF-GA (All-uses Data Flow criterion based test case generation using Genetic Algorithm) for smart contracts written in Solidity language. The test cases generated in this way are able to have wide coverage and a reduced number of iterations using genetic algorithms.

A topic that received a great interest in the literature is the detection of bugs occurring in smart contracts. The bugs in a smart contract are difficult to fix after it has been deployed and this could cause a loss of money. The Zeus (Kalra et al., 2018) framework was proposed to check the correctness and confirm the fairness of smart contracts. By correctness authors mean using secure programming practices, instead fairness indicates compliance with high-level business logic. ZEUS simultaneously

Table 4
Tools and features.

Tool/Framework	Technology	Technique	Description	Experimental Dataset	Findings
CONTRACTLA-RVA (Ellul and Pace, 2018)	Ethereum	Standard techniques for runtime verification.	A tool for runtime verification of smart contracts to ensure safety.	Parity Multisig Wallet smart contracts.	Useful to ensure that smart contracts have certain specifications.
KEVM (Hildenbrandt et al., 2018)	Ethereum	Formal analysis.	A formal specification executable EVM's bytecode stack-based language.	Official Ethereum test suite.	KEVM is complete, performing and can be used in an environment of continuous integration.
SmartCheck (Tikhomirov et al., 2018)	Ethereum	Static code analysis.	Static analysis tool to detect code issue related to security, operations, development and functionality.	4,600 verified smart contracts.	The tool identified 99.9% contracts with issues, 63.2% contracts with vulnerability.
MadMax (Grech et al., 2018)	Ethereum	Static code analysis.	Static analysis program to detect gas-vulnerabilities.	91,800 smart contracts.	81% of the samples considered present warnings.
RA (Re-entrancy Analyzer) (Chinen et al., 2020)	Ethereum	Static code analysis.	A static analysis tool to find re-entrancy attacks.	Known re-entrancy vulnerabilities.	The tool precisely identifies vulnerable smart contracts.
Samreen & Alalfi (Samreen and Alalfi, 2020)	Ethereum	Combination of static and dynamic analysis.	A framework to detect Re-entrancy vulnerabilities.	5 modified smart contracts.	The combination of the two techniques improves the performance and decreases the false positives.
SolMet (Hegedűs, 2019)	Ethereum	Code parsing, static code analysis.	A tool to measures static OO metrics in Solidity smart contracts.	10,206 Solidity source code file with nearly 45,000 smart contracts.	The tool is immature at the current state.
PASO (Pierro and Tonelli, 2020)	Ethereum	Code parsing, static code analysis.	A web-based parser for Solidity language analysis.	No validation has been provided.	Calculate metrics of smart contract and can be used with different versions of the Solidity language.
VerX (Permenev et al., 2020)	Ethereum	Abstraction and symbolic execution engine.	A symbolic execution engine to find functional properties of Ethereum smart contracts.	12 real world Ethereum projects.	Verx is practical and useful to verify functional properties of smart contracts.
Gasper (Chen et al., 2017)	Ethereum	Symbolic execution.	A tool to locate gas-costly programming pattern (Dead Code, Opaque Predicates,Expensive Operations in a Loop).	4,240 smart contracts.	93.5%, 90.1% and 80% smart contracts suffer from three different gas-costly pattern patterns.
GasChecker (Chen et al., 2020a)	Ethereum	Symbolic execution.	A tool for automatic identification of inefficient code for gas consumption in smart contracts.	1,500 representative smart contracts from 599,934 contracts.	Most real contracts contain inefficient code.
MAIAN (Nikolić et al., 2018)	Ethereum	Symbolic execution.	A tool to trace properties of smart contracts to find exploit.	970,898 smart contracts.	Most of the smart contracts identified by the tool are true positives.
SmartEmbed (Gao et al., 2020)	Ethereum	Code parsing.	A prototype to identify code clones.	More than 22,000 smart contracts.	The tool identified 90% of the clones.
SIF (Peng et al., 2019)	Ethereum	Code instrumentation and analysis.	A framework for monitoring, instrumenting, and generating code.	51 smart contracts.	The tool can analyze the AST, retrieve information and generating Solidity code.
Oyente (Luu et al., 2016)	Ethereum	Symbolic execution.	An analysis tool for smart contracts to detect security bugs.	19,366 Ethereum contracts	The tool identifies vulnerable smart contracts.
TEETHER (Krupp and Rossow, 2018)	Ethereum	Symbolic execution.	The tool creates exploits for contracts given its binary bytecode.	38,757 Ethereum contracts.	The tool finds exploits for 815 on 38,757 smart contracts.
Slither (Feist et al., 2019)	Ethereum	Static code analysis.	A tool to find vulnerabilities, code optimization, code review, bug detection.	1,000 smart contracts.	The tool is better than other in terms of speed, robustness and false positives.
Securify (Tsankov et al., 2018)	Ethereum	Symbolic analysis.	Security analyzer for Ethereum smart contracts.	18,000 smart contracts.	The tool demonstrates the correctness of smart contracts and discovers violations.
Contract-fuzzer (Jiang et al., 2018)	Ethereum	Static code analysis, fuzz testing.	A fuzzer to test Ethereum smart contracts for security vulnerabilities.	6,991 smart contracts.	The tool detects 459 vulnerabilities on 6,991 smart contracts tested.
EVMFuzz (Fu et al., 2019)	Ethereum	Static code analysis, differential fuzz testing.	Differential fuzz testing to detect vulnerabilities of EVMs.	36,295 smart contracts.	Most of the smart contracts generated have different performance.

(*continued on next page*)

Table 4 (*continued*).

Tool/Framework	Technology	Technique	Description	Experimental Dataset	Findings
ReGuard (Liu et al., 2018)	Ethereum	Code parsing, fuzz testing.	A fuzzing-based analyzer to detect re-entrancy bugs in smart contracts.	5 modified smart contracts.	The tool detects re-entrancy vulnerabilities dynamically.
Kaya (Wu et al., 2020)	Ethereum	Test cases specification.	A testing framework to write and execute test cases.	Two case study.	The framework is useful in testing Dapps.
SmartInspect (Bragagnolo et al., 2018a)	Ethereum	Code parsing, static code analysis, decompilation techniques.	Inspector of smart contracts written in Solidity focusing on debugging properties.	An example of a smart contract written in Solidity.	Reflective approach allows users to see the content of contracts without redeploy them.
FSolidityM (Mavridou and Laszka, 2018)	Ethereum	Finite state machine based model.	A framework to create safer smart contracts, enhancing the security and functionality of contracts.	A smart contract with several functions.	They provide a graphical editor and a set of plugins to help developers.
Zeus (Kalra et al., 2018)	Ethereum, Hyperledger Fabric	Abstract interpretation, symbolic model checking.	A framework to verify correctness of smart contracts.	22,400 smart contracts	94.6% contracts are vulnerable.
Blockbench (Dinh et al., 2018, 2017)	Ethereum, Parity and Hyperledger Fabric.	Benchmarking.	A benchmarking framework for private blockchain systems.	Platforms: Ethereum, Parity and Hyperledger Fabric.	Different design compromises and performance gaps between blockchain and database systems.

uses abstract interpretation, the symbolic model checking and the horn clauses (with semantic rules, then translate into assertion) to speedily check the security of smart contracts. Zeus has been tested on over 22,000 smart contracts and it has shown that around 94.6% of them are vulnerable.

For runtime verification of smart contracts, standard techniques can be used, including a novel stake-based instrumentation technique to ensure that the participating parties comply with the contracts. A proof-of-concept tool CONTRACTLARVA (Ellul and Pace, 2018) implements this technique. CONTRACTLARVA receives the combination of smart contract and its specifications as input and automatically transforms it into a safe contract with the same behaviors as the original. The approach could be used to ensure that smart contracts adhere to a particular specification. For example, if the smart contract has bugs that can lead to financial losses, these specifications can avoid it.

With regards to the problem of correcting flawed programs, the problem of identifying contract defects has been faced. Chen et al. (2020b) proposed an empirical study to describe contract defects. From their analysis, they identified 20 contract defects related to security, architecture, and usability problems, and validated the emerging contract defects through an online survey aimed at collecting opinions from real-world smart contract developers. Considering the existing tools (such as Oyente Luu et al., 2016, Zeus Kalra et al., 2018 and Contractfuzzer Jiang et al., 2018), they also showed that it is possible to detect only 7 of the defined code smells, while 13 of them are not covered by the existing tools, highlighting the need of developing techniques aimed at detecting such smells.

Another issue felt by the developers community is how to improve transactions to add a block in the blockchain. A block of the chain consists of multiple transactions of smart contracts that are added by a miner. Specifically, the miner firstly executes these transactions sequentially. To guarantee that a correct block is added to the blockchain, the validators serially re-execute the same transactions. The block can be added to the blockchain only if the validators agree with the final state of the block as recorded by the miner. Performing smart contract transactions in series implies low performance. Instead, adding concurrency to smart contract execution can increase efficiency. Anjana et al. (2019) proposed a new way of performing smart contract transactions concurrently using Software Transactional Memory systems (STM). To achieve this, they use the Basic Time stamp Ordering (BTO) and Multi-Version Time protocols stamp Ordering (MVTO) for SMT and propose a library to generate block graphs (BG). Using a concurrent validator and BG, miners can efficiently re-execute smart contract transactions in order to satisfy the correctness criterion. However, no real implementations of this strategy are provided.

Mavridou and Laszka (2018) proposed FSolidM, a framework for secure creation of smart contracts. This framework is rooted in rigorous semantics for the design of contracts such as finite state machines (FSM) to help developers create safer smart contracts and for enhancing the security and functionality of contracts. Based on this FSM-based model, they implemented a graphic editor for contract design and automatic code generation. Then they implemented a set of plugins that developers can add to their contracts. Among these plugins, two of them, *locking* and *transition counter*, implement security features to prevent common vulnerabilities (i.e. return was unpredictable). The other two plugins, *automatic timed transitions* and *access control*, implement common design models for facilitating the development of correct contracts with complex functionalities.

Since also protocols must be validated, testing techniques have been developed for dealing with this problem. Chepurnoy and Rathee (2018) aimed to improve the quality of blockchain protocol implementations. They tested abstract properties to check if a blockchain system respects certain properties that each blockchain system should respect. They test history, minimal state and memory pool components separately generating random samples. The suite is composed of 59 tests that check the different properties of a blockchain system. To run the suite on a concrete blockchain protocol client, developers generate random objects used by the protocol. The test suite then checks the satisfaction of the properties over many random cases.

Other studies have proposed fuzz testing, to detect software errors or security flaws, offering tools such as EVMFuzz (Fu et al., 2019), ContractFuzzer (Jiang et al., 2018), ReGuard (Liu et al., 2018) and Fuse (Chan and Jiang, 2018). The differential fuzz test continuously provides invalid, unexpected or random data as input for different programs with the same functions. These programs are then monitored intercepting different behaviors from the usual defined as "different act" on some inputs, in this case, they could find some program bugs. EVMFuzz (Fu et al., 2019) aims at detecting vulnerabilities of EVMs with differential fuzz

Table 5
Empirical validations.

Author	Technology	Description	Dataset components	Findings
Hegedűs (2019)	Ethereum	Usage of static OO metrics to the smart contracts written in the Solidity contract-oriented language.	More than 45,000 contracts from 10,206 Solidity smart contract source code files.	Smart contracts are short, uncomplicated and little commented.
Aldweesh et al. (2018a)	Ethereum	Benchmark approach to assess the CPU usage required per EVM opcode; comparison with the set gas cost.	EVM opcodes are classified into eleven categories.	This study provides some interesting insights about reward for invested CPU resources, with respect to opcodes and computer platform.
Aldweesh et al. (2018b)	Ethereum	Benchmarking approach to assess whether the fees miners receive from creating and executing smart contracts is proportional to the cost as expressed in terms of CPU usage.	80 smart contracts.	There are discrepancies result in misaligned incentives that impact the dependable operation of the blockchain.
Wang et al. (2019b)	Ethereum	Generate cost-effective test-suites using two multi-objective test generation approaches.	8 Dapps.	The proposed approach could reduce the gas and time cost while retaining the ability to cover branches.
Tonelli et al. (2018)	Ethereum	Evaluate if there are statistical differences between metrics for smart contract and traditional ones.	12,000 smart contracts.	Smart contracts metrics have ranges more restricted than the corresponding metrics in traditional software systems.
Anjana et al. (2019)	Ethereum	Efficient framework for concurrent execution of smart contracts.	A set of benchmarks generated for Ballot, Simple Auction, and Coin contracts.	The proposed protocols are faster than the previous solution in series.
Bez et al. (2019)	Ethereum	To assess the current implementation of blockchain and the associated improvement proposals.	Thousands of Ethereum nodes.	Ethereum is safe, decentralized but not very scalable.
Wohrer and Zdun (2018)	Ethereum	Security patterns in Solidity.	Data from different sources.	Six design patterns related to security issues in the code.
Norvill et al. (2017)	Ethereum	A framework for grouping similar contracts.	998 smart contracts.	Identification of the purpose of a contract with clustering .
Ye et al. (2019)	Ethereum	Analysis of the different tools to detect bugs in smart contracts.	1,010 smart contracts in 200 files.	Most smart contracts lack security.
Chen et al. (2018b)	Ethereum	Identify the main Ethereum activities through graph analysis.	28,502,131 external transaction and 19,759,821 internal transaction.	Results help to better understand Ethereum.
Tian et al. (2020)	Ethereum	Smart Contract Classification With a Bi-LSTM Based Approach.	15,213 smart contracts.	The proposed approach is effective.
Chen et al. (2019)	Ethereum	Detecting Ponzi Scheme in smart contracts.	3,000 smart contracts.	The proposed model works better than the traditional ones.
Marchesi et al. (2020)	Ethereum	Pattern to optimize gas consumption.	24 design pattern divided in 5 categories.	Applying these patterns, the gas consumption can be mitigated.
Chepurnoy and Rathee (2018)	Bitcoin	Exhaustive testing of an abstract blockchain protocol implementation.	Random samples.	Test suite checks the satisfaction of certain blockchain properties.
Ortu et al. (2019)	Bitcoin-core, Ethereum-go, Monero, Dodgecoi, Ripple.	Detecting differences between the two categories of projects, and in the statistical distribution of 10 software metrics.	5 C++ open source blockchain-oriented and 5 traditional java software systems.	Metrics analysis could identify differences in programming and revealing meaningful differences between the two projects domain.
Wessling et al. (2019)	Hybrid software architectures.	Impact of design patterns in smart contracts on transaction costs.	Cost comparison to perform transactions by simulating three usage scenarios.	Creating a hybrid application with the right balance of the elements is still difficult.
Thakkar et al. (2018)	Hyperledger Fabric	Performance and optimization of Hyperledger Fabric.	Variation of configurable parameters like block size, channels, resource allocation.	Six guidelines on configuring parameters and identified performance bottlenecks.

testing. The idea behind EVMFuzz consists in continuously generating seed contracts for several EVM executions to find inconsistencies between possible results, even with cross references. They defined a EVM fuzz testing with two metrics (gasUsed and opcode sequence), implemented in different programming languages to evaluate the differences between the different EVM platforms. They have mutated 36,295 smart contracts in the real world and generated 253,153 smart contracts. Based on the results, 66.2% showed differential performance, including 1596 variant contracts showed inconsistent output. Five unknown security bugs were found and have been included in the CVE (Common Vulnerabilities and Exposures) database.

ContractFuzzer (Jiang et al., 2018) is a fuzzer to test Ethereum smart contracts for security vulnerability. During its execution, input fuzzing is generated according to ABI specifications of smart contracts and then they are defined test oracles to detect security vulnerabilities. Later, logs are produced with the vulnerabilities identified. By considering 6991 smart contracts, the tool reported more than 459 vulnerabilities with high accuracy, especially those of "The DAO" bug[6] and the Parity Wallet bug[7] leading to millions of dollars in losses.

[6] DAO (Decentralized Autonomous Organization) operates as support for an organization through rules defined in smart contract. "The DAO" consists of complex smart contracts running on the Ethereum blockchain.

[7] The Parity wallet is used to integrate tokens and manage Ether transfers.

Table 6
Case studies.

Author	Technology	Subject	Description	Findings
Aung and Tantidham (2017)	Ethereum	Smart home case study using blockchain.	The authors show how to apply Ethereum private blockchain for Smart home System (SHS).	Home owner can: check transaction history done, set up the policies for handling transactions.
Manzoor et al. (2018)	Ethereum	A Cash-less payment scheme using blockchain.	The scheme uses private Ethereum blockchain for transaction processing within the remote village.	Demonstration of the feasibility of the system design with a prototype implementation.
Hinckeldeyn and Jochen (2018)	Ethereum	Smart Storage Container for a blockchain-based Supply Chain Application.	A prototype of smart storage containers to assess risks and maturity of blockchain and Internet of Things for logistical processes.	Smart storage containers and smart contracts could be coupled. Developments are needed in order to make it a use case.
Zinca and Negrean (2018)	Ethereum	A road tax payment application using the Ethereum platform.	The development of a road tax payment application using the Ethereum platform.	The proposed solution has low costs, processing speed and safety.
Chan and Jiang (2018)	Ethereum	Fuse, a fuzz testing service.	Architecture for Smart Contract Fuzz Testing Service.	The techniques are able to detect about 96% of security problems.
Marchesi et al. (2018)	Ethereum	BOSE application.	Agile practices for the software development process.	Software engineering practices in blockchain applications can be helpful to have better results.
Zhang et al. (2020)	Ethereum	ADF-GA (All-uses Data Flow criterion based test case generation using Genetic Algorithm).	Test case generation approach for Solidity.	Ability to generate test case with wide coverage.
Rouhani and Deters (2017)	Ethereum	Transaction in Parity and Geth.	Ethereum transactions performance using Ethereum blockchain and two clients, Parity and Geth, separately.	With the same configuration, transactions are faster by 89.8% in Parity client than Geth client.
Zhang et al. (2017)	Ethereum	Metrics for assessing blockchain-based healthcare apps.	Metrics to evaluate Dapps in terms of feasibility, intended capability, and compliance in the healthcare domain.	Initial guide for creating future apps in this domain.
Delgado-Mohatar et al. (2019)	Ethereum	Biometric Template Storage with blockchain.	In the analysis were analyzed costs and performance for a biometric system based on blockchain.	Integration between biometric system and public blockchain is possible.
Nizamuddin et al. (2019)	Ethereum	Document sharing and version control based on blockchain.	A blockchain-based solution and framework for document sharing and version control.	The system is decentralized, secure, and resilient.
Ranganthan et al. (2018)	Ethereum	Marketplace Application.	Simulation of Marketplace Application on The Ethereum blockchain.	Newly emerging field.
Payette et al. (2017)	Ethereum	Ethereum address space.	Segmentation of the Ethereum address space into distinct behavior groups.	Identify cluster characteristics and behavior.
Chen et al. (2018a)	Ethereum	Ponzi Schemes identification in blockchain.	Using data mining and machine learning to identify Ponzi schemes.	The approach is accurate.
Huang (2018)	Ethereum	Potential attacks in smart contracts.	Optimize parameters, network structure and analysis tools with convolutional neural network (CNN).	The proposed system assesses safety with little effort.
Martens and Maalej (2018)	Ethereum	ReviewChain, a decentralized review approach.	To resolve the problem of central authorities by using the public Ethereum blockchain.	Through the application, users can access reviews on the blockchain in a usable way.
Hukkinen et al. (2019)	Ethereum	Ethereum Transaction Costs.	Reducing Ethereum Transaction Costs in a Blockchain Electricity Market Application.	Useful guidelines to optimize smart contract efficiency for Ethereum.
Wöhrer and Zdun (2020)	Ethereum	Domain-specific smart contract language.	Language based on a high level of abstraction.	Useful to reduce complexity and increase comprehensibility.
Bragagnolo et al. (2018b)	Ethereum	Ethereum Query Language.	Facilitate the way the data extracted from the blockchain database are recovered, formatted and presented.	Useful for users who need to search for information in the database.
Tonelli et al. (2019)	Ethereum	Microservices System.	Microservices-based system using a set of Smart Contracts written in Solidity.	It is possible to realize such application maintaining the same functionalities.
Kfoury and Khoury (2018)	Ethereum	Voice over Long Term Evolution (VoLTE) technology.	Approach for securing End-to-End (e2e) VoLTE media based the Ethereum Blockchain	Solution has little impact on existing IMS (IP Multimedia Subsystem) network.
Destefanis et al. (2018)	Parity	Bug discovered in a Parity smart contract library.	Parity source code and library analysis.	Library vulnerability due to negligent programming.
Atzei et al. (2019) and Bartoletti and Zunino (2018)	Bitcoin	BitML, a domain-specific language for smart contracts.	Toolchain for developing and verifying smart contracts executed on Bitcoin	BitML has advantages and limitations.

(*continued on next page*)

Table 6 (*continued*).

Author	Technology	Subject	Description	Findings
Gencer et al. (2018)	Bitcoin, Ethereum	Bitcoin and Ethereum Networks.	Measurement study on decentralization metrics.	Bitcoin has a larger capacity network than Ethereum, but with more cluster nodes in the data centers.
Pongnumkul et al. (2017)	Hyperledger Fabric, Ethereum.	Performance analysis of Hyperledger Fabric and Ethereum.	Evaluate the performance and limits of the two platforms varying number of transactions.	Varying the workload, Hyperledger fabric has a higher throughput and lower latency than Ethereum.
Coblenz et al. (2019)	Hyperledger Fabric	Obsidian, a new blockchain programming language.	The language should improve safety, security, and developer effectiveness.	Useful to take into account the needs of users, to ensure security and detect bugs.
Chang et al. (2019)	Blockchain-based process	Smart contract based tracking process.	Conceptual guidelines for practical business system design and implementation.	Paying using digital currency is possible and saves time.
Yasaweerasinghe-lage et al. (2017)	Blockchain-based systems.	Predicting Latency of Blockchain-Based Systems.	Use of consolidated tools and techniques but also analysis of possible new problems.	Use architectural performance modeling and simulation tools to predict blockchain latency is feasible.
Rocha and Ducasse (2018)	Blockchain-oriented software (BOS)	Steps Towards Modeling Blockchain Oriented Software (BOS).	Deal with specialized blockchain modeling notations.	The study highlights a lack of modeling notation for BOS systems.
Amoordon and Rocha (2019)	Tendermint	Tendermint, an application-based blockchain.	Advantages and disadvantages of Tendermint platform compared to other platforms.	Tendermint is a solid project.
Xu et al. (2019)	OriginChain	OriginChain, a blockchain-based traceability system.	The application restructures the current system by replacing the central database with the blockchain.	The structural design of the smart contract affects the quality of the system.
Meng and Qian (2018a,b)	DelivChain	DelivChain, blockchain Application in Supply Chain Management.	Metric for Predictive Delivery Performance in Supply Chain Management.	Potential benefits in using the tool.
Suankaewmanee et al. (2018)	MobiChain	MobiChain, m-commerce application using blockchain.	The application ensures secure transactions in m-commerce.	Mining processes can be performed on an Android mobile device.
Pradeepkumar et al. (2018)	Blockchain Application Design.	To measure the digitizability complexity of a regulation document.	Modeling approach that supports the automated analysis of human-readable regulation.	Help developers measure document readability.
Liu (2018)	Blockchain concepts.	ChainTutor, java application for learning basic blockchain concepts.	Creating a tool with compact GUI-based learning useful to teachers and students.	Users can change the parameters in the blockchain to better understand how it works.
Shrivas et al. (2020)	Any kind of blockchain platform.	Hybrid security framework.	Reference model for identifying security treats in blockchain.	Usable with any blockchain platform.
Clack et al. (2016)	Axoni Core, Corda, Digital Asset Platform, Ethereum, Hyperledger Fabric.	Smart Contract Templates.	Support for the life cycle management of a smart contract in the legal field.	Useful to financial service industry.
Jiang et al. (2019)	Ethereum Quorum and SERO.	A Privacy-Preserving E-Commerce System.	Implementation of a corporate protocol to preserve privacy through smart contracts in the negotiation phase.	The proposed model is feasible.
Hamida et al. (2017)	Multi-chain,Quorum, Hyperledger Fabric Open-Chain,Chain Core,Corda, Monax.	Enterprise Blockchain.	Analysis of technological components and applications.	Taxonomy of applications, use cases.
Zhang et al. (2019)	Hybrid-storage blockchain	GEM^2-Tree, Authenticated data structure (ADS) gas efficient.	Design of an authenticated data structure that can be adequately managed by the blockchain.	Useful to support authenticated queries.

Fuse (Chan and Jiang, 2018) is a fuzz testing service. Compared to ContractFuzzer, it found about 50% less true positive cases in the experiment.

ReGuard (Liu et al., 2018) uses the fuzz testing to detect re-entrancy bugs into Ethereum smart contracts.

Beyond such tools, Samreen and Alalfi (2020) have proposed a framework that combines static analysis to detect re-entrancy vulnerabilities and dynamic analysis to confirm such vulnerabilities. To test the framework, authors modified 5 smart contracts, observing that the framework was able to detect the vulnerabilities introduced in the code.

Because Dapps are exposed to serious vulnerabilities, Wu et al. (2020) proposed a framework called Kaya. Kaya provides a descriptive Dapp behavior language (DBDL) to help write test cases, which are then executed automatically. In addition, Kaya modifies incomprehensible addresses into readable variables to help in understanding. These functions are useful to test Dapp more easily.

Table 7
Surveys.

Author	Technology	Subject	Quantitative Description	Findings
Zou et al. (2019)	Ethereum	Challenges and opportunities in the development of smart contract.	Interviews with 20 developers and survey on 232 practitioners to validate the findings from the interviews.	Identified different limits in the programming language and available tools.
Atzei et al. (2016)	Ethereum	Attacks which exploit vulnerabilities.	Analyze the security vulnerabilities of Ethereum smart contracts and common programming pitfalls	The difficulty of detecting mismatches between intended behavior and the actual one.
Chen et al. (2020b)	Ethereum	Contract defects in Smart Contracts.	Online survey to validate contract defects identified by the study.	Help developers decide defects removal priority.
Ranganthan et al. (2018)	Ethereum	Marketplace applications on the Ethereum blockchain.	Analysis of the most popular startups in the field of Marketplace Applications.	There is no mature application for a decentralized marketplace yet.
Di Angelo and Salzer (2019)	Ethereum	Tools for smart contract analysis.	Analysis of existing tools through installation and testing.	Five tools are particularly useful (FSolidM, KEVM, MAIAN,Securify,Mythril).
Bartoletti et al. (2020)	Ethereum	Ponzi schemes detection.	Behavior and impact of the Ponzi scheme on Ethereum.	10% of the 1384 verified contracts are Ponzi schemes.
Bartoletti and Pompianu (2017)	Bitcoin, Ethereum	Concept of smart contract in Bitcoin and Ethereum.	They quantify the usage of smart contracts in relation to their application domain.	Useful information for developers to develop a new language for smart contracts.
Wang et al. (2019c)	Bitcoin, Ethereum	Internet of Things Applications.	Analysis of existing blockchains with a focus on those based on IoT applications.	To integrate Iot and blockchain, blockchain should improve blockchain capacity, security, and scalability.
Yamashita et al. (2019)	Hyperledger Fabric	Risks of Hyperledger Fabric Smart Contracts.	Risks related with chaincodes using Go language.	13 potential risks.
Dinh et al. (2018)	Ethereum, Parity, and Hyperledger Fabric	Blockchain technologies.	State of the art analysis and blockchain classification based on distributed ledger, cryptography, consensus protocol and smart contract.	Comparison of findings found in each area of analysis.
Sayeed et al. (2020)	Ethereum, Rootstock (RSK), EOS, The Real-time Operating system Nucleus (TRON), Steller, Hyperledger Fabric, Cardano, and Corda	Smart contract security vulnerabilities.	Review regarding the integration of artificial intelligence techniques to maintain privacy in smart contracts.	Complex smart contracts does not lessen privacy and security issues.
Wessling et al. (2018)	Blockchain technologies	Building hybrid Dapp architectures.	A possible approach to assess in which elements of an 'application architecture' blockchain technology could be used.	Lack of a holistic engineering approach to creating hybrid systems.

Open challenges in Smart Contract Testing: *investigating how traditional testing techniques, like unit and integration testing, mutation testing, test case generation, and regression testing, must be adapted to the development process and structure of a smart contract, also considering the immutability of blockchain and the need to test smart contracts on testnets, before actually deploying them.*

4.2. Smart contract code analysis

Research in this area mainly focused on: static analysis framework (Hildenbrandt et al., 2018), gas-costly bytecodes (Aldweesh et al., 2018a,b; Chen et al., 2017, 2020a; Marchesi et al., 2020), tools for smart contract analysis (Di Angelo and Salzer, 2019; Ye et al., 2019; Tikhomirov et al., 2018; Tsankov et al., 2018; Feist et al., 2019; Permenev et al., 2020), tools for clone detection (Gao et al., 2020).

4.2.1. Tools for smart contract analysis

In this first subsection, static and dynamic analysis tools for Ethereum smart contracts are examined.

An extensible static analysis tool is Smartcheck (Tikhomirov et al., 2018). It translates the source code into an intermediate XML-based representation and then compares it with XPath models. The tool has been tested on 4600 verified smart contracts downloaded from Etherscan.[8] SmartCheck is able to detect security, functionality, operational and development problems that can make the code difficult to understand. Analysis shows that 99.9% of contracts have issues, 63.2% of contracts have critical vulnerabilities. Securify (Tsankov et al., 2018) aims at verifying secure/unsafe contractual behavior with respect to certain properties. For example, a property states that writing to owner field is limited, so not all users can do a transaction that writes in this field.

Another tool proposed in literature is Slither (Feist et al., 2019). It provides detailed information on smart contracts written in Solidity by converting them into an intermediate representation called SlithIR which preserves the semantic information that would instead be lost using the bytecode. This tool is used to detect automatically vulnerabilities, identify ways to optimize the code, improve understanding of smart contracts and assist in code review.

SmartInspect (Bragagnolo et al., 2018a) uses decompilation techniques for visualizing the current state of a smart contract instance without redeploying it nor developing additional code

[8] https://etherscan.io/ - Etherscan is a Block Explorer, Search, API and Analytics Platform for Ethereum.

for decoding. Since the status of a deployed contract is read-only and opaque, SmartInspect can be used to introspect the contents of any deployed contract, with the aim of identifying defects, such as incomplete or inconsistent specifications. Another way to understand the characteristics of smart contracts is proposed by SmartEmbed (Gao et al., 2020). Understanding the features can be useful for detecting clones and bugs in the code. Smart contracts are converted to word stream with structural information, then converted to Vectors and compared with others. The results show that the tool is able to detect 90% of the clones present.

KEVM (Hildenbrandt et al., 2018) was developed for supporting program analysis and verification. It is a formal executable specification of the language based on the stack bytecode of the Ethereum Virtual Machine (EVM) built with the K Framework used for formal analysis.

Another tool to automatically verify functional properties of smart contracts is VERX (Permenev et al., 2020). This tool is based on a combination of three techniques: (i) reduction of time property verification to reachability control, (ii) a symbolic execution engine for the precise and efficient Ethereum Virtual Machine, and (iii) the abstraction of the delayed predicate which uses symbolic execution during transactions and abstraction at the borders of transactions. An experimental evaluation conducted on 83 temporal properties and 12 real world projects demonstrated the effectiveness of the tool.

On the contrary, other authors proposed support tools for the development of Solidity smart contracts as SIF (Solidity Instrumentation Framework) (Peng et al., 2019). SIF provides a way to query the AST (Abstract Syntax Tree) of the code; to insert, modify and delete nodes in the AST; to generate code from AST which reflects the changes done by the user. SIF aims to monitor, analyze, optimize and generate Solidity code. The tests were carried out on 51 real smart contracts to check assertions and mutation tests by inserting errors. The empirical evaluation shows that SIF is capable of automatically analyzing the AST and representing it using C++ classes, provides a user interface to retrieve information and make changes to the AST and generate Solidity code from the instrumented AST.

To evaluate the quality of the different tools in the literature, authors in Di Angelo and Salzer (2019) and Ye et al. (2019) have compared existing tools, including Slither, Smartcheck, to guide developers in analyzing already deployed code. 27 tools for the analysis of Ethereum smart contracts have been analyzed considering availability, maturity, methods used and the degree of security. Finally, authors suggest using 5 tools they deem most useful: FSolidM to generate source code from specifications, KEVM for formal verification; Securify regarding formal guarantees; MAIAN for finding vulnerabilities and Mythril for analyzing contracts interactively. Zou et al. (2019) proposed a survey to highlight the challenges still open in the development of smart contracts in Ethereum. The main emerged issues were: it is difficult to ensure the safety for smart contracts, the existing tools are at a basic level, the programming language and the online resources are not enough mature.

Unlike previous work mainly targeting Ethereum, some researchers focused on the Bitcoin platform,[9] which does not have a high-level contract language, nor related development and verification tools. In Atzei et al. (2019) and Bartoletti and Zunino (2018), the authors propose a toolchain for the development and verification of smart contracts that can be executed on Bitcoin platform. The toolchain is based on BitML, a recent domain specific language for smart contracts with a computationally integrated sound in Bitcoin. BitML has several limitations, including the fact that is not complete. Indeed, some smart contracts can be expressed in Bitcoin but not in BitML. BitML could be extended to express contingent payments contracts, where participants solve a class of NP problems.

Another problem concerns the presence of bugs after the code has been released. Destefanis et al. (2018) highlight the need for a discipline of blockchain software engineering that addresses such problems. They analyze an attack to Parity, a wallet application, due to a bug discovered in a smart contract library. Bad programming practices allowed an anonymous user to accidentally freeze around 500 K Ether ($ 150 million in November 2017). This vulnerability in the library was caused by negligent programming rather than a problem in the Solidity language. This is also due to existing approaches to software development of smart contracts that are insufficient.

Besides technologies like Ethereum and Bitcoin, Hyperledger Fabric is increasingly established as a blockchain platform. Hyperledger Fabric (Yamashita et al., 2019) is a permissioned blockchain framework that uses generic programming languages, like Go and Java, to implement smart contracts (called chaincode). These languages have advantages and disadvantages. Among the disadvantages, there may be risks that developers did not need to consider in traditional systems since these languages were not originally created to write smart contracts. For this reason, the authors analyzed the Go language and its tools. Analyzing the risks associated with chaincodes, they highlighted 14 potential risks. Then they analyzed how many of them can be covered by existing tools. The results show that some risks are not covered. They have implemented a static analysis tool to cover these risks and have evaluated their usefulness. This study could be extended to other programming languages.

4.2.2. Smart contract classification

Concerning smart contract classification the majority of research efforts targeted Ethereum, Bitcoin and blockchain technologies in general. Sometimes it is difficult to communicate without errors between developers and stakeholders. To reduce this gap, Wöhrer and Zdun (2020) have created a programming language for smart contract with higher abstraction level that can be automatically transformed into an implementation. Such language is closer to the natural one and can reduce the complexity of the project to make it more understandable and less prone to errors. Instead, Norvill et al. (2017) create a framework to group similar contracts within the Ethereum network using the publicly compiled codes available for contracts. The clustering techniques used were: Affinity Propagation and K-medoids. The results confirmed that clusters based on bytecode hash similarity can be created to identify the purpose of a contract by adding a label to each cluster. However, clusters with similar purposes have been identified. In order to avoid this, in the future, supervised learning methods that allow grouping them could be used.

Another way to group and classify smart contract is by using Bi-LSTM model and Gaussian LDA (Tian et al., 2020). Information such as source code, comments, and tags are used as input for the template. Then Bi-LSTM captures grammatical rules, while Gaussian LDA generates comments. The results show that the model has good performance.

While previous studies have analyzed smart contracts for a given platform, such as Ethereum, other studies have compared smart contracts across different platforms. Bartoletti and Pompianu (2017) have examined a dataset of 6 platforms for smart contracts, showing technical differences between them. For the two most popular ones, Bitcoin and Ethereum, they examined a sample of 834 contracts, classifying them according to their application domain and measuring the relevance of each of these categories. This study can provide valuable information to developers new domain specific languages for smart contracts. In addition, measuring the most common use cases is useful to understand which domains deserve more investment.

[9] https://bitcoin.org

4.2.3. Gas consumption

In addition to the analysis of the source code, other authors have dealt with the gas consumption for smart contracts execution. This issue is specific to Ethereum, and other blockchains using the gas concept. Indeed, In this third subsection the Ethereum platform is mainly considered.

There is a cost in terms of gas for each operation code (opcode) associated with smart contracts. The authors try to assess whether the commissions that miners receive from the creation and execution of smart contracts are proportional to the cost expressed in terms of CPU usage (Aldweesh et al., 2018a) and whether the CPU required is proportional to the gas required (Aldweesh et al., 2018b). Based on the results, the gas used is not proportional to the computational effort for the creation and execution of smart contracts. Payette et al. (2017) group together the Ethereum address space into clusters using algorithms like k-means. Then, they report the quantitative characteristics of each group and make qualitative inferences on the behavioral traits of the clusters.

Another aspect of the gas to be analyzed is how it can be reduced. In 2017, Chen et al. (2017) propose to replace source code with a more efficient code to optimize gas consumption. They implement GASPER to automatically discover gas-costly programming patterns from the bytecode of smart contracts. They identify 2 categories: Useless Code Related Patterns and Loop Related Patterns. In this categories, they identified 7 gas-costly patterns. GASPER manages to identify 3 types of gas-costly patterns: Dead Code, Opaque Predicates and Expensive Operations in a Loop. Analyzing a dataset of 4240 smart contracts, the tool identified 93.5%, 90.1% and 80% respectively of smart contracts that present these patterns.

Subsequently, the same authors, in 2020, proposed a tool called GasChecker (Chen et al., 2020a) that allows to automatically find inefficient gas code in smart contracts. This tool, using a Mapreduce programming model, is scalable and produces few false positives.

Finally, Marchesi et al. (2020) have identified 24 design patterns that can affect gas consumption. These patterns have been divided into 5 categories: external transaction, storage, saving space, operations and miscellaneous. For each of them they identified the problem and a possible solution.

Open challenges in Smart Contract Analysis: *identify expensive pattern and optimization mechanisms for smart contracts; develop smart contracts specific metrics for evaluating code properties related to quality; produce code analysis methods that are independent from the development language or that could cover other languages besides Solidity and Go.*

4.3. Smart contract code metrics

Researchers investigated similarities and differences between traditional software and blockchain-oriented software (BOS) from different perspectives (Marchesi et al., 2018; Ortu et al., 2019; Rocha and Ducasse, 2018) and software metrics, collected on Solidity smart contracts, represent the premier means to explore BOS peculiarities (Hegedűs, 2019; Tonelli et al., 2018; Gencer et al., 2018; Pierro and Tonelli, 2020).

4.3.1. Blockchain-oriented software (BOS)

As blockchain technologies become increasingly popular, also blockchain-oriented software (BOS) continues growing uncontrollably, without taking into account the quality of produced software and the basic concepts of software engineering. To tackle this problem, researchers in Marchesi et al. (2018) and (Rocha and Ducasse, 2018) proposed software development processes, based on agile methods and UML diagrams that take into account the specific peculiarities of this type of software. This allows blockchain companies and ICO startups to benefit from software engineering best practices. However, both studies highlighted the need for conceiving tools for BOS modeling to help developers writing high-quality code.

To take full advantage of blockchain technologies, it is also important to understand the structure and the behavior of BOS systems, and how they differ from traditional software systems. In this context, Ortu et al. (2019) provided a statistical characterization of BOS, by comparing blockchain-oriented and traditional Java software systems, relying on 10 software metrics. They found that significant differences between the two sets can be observed in the distributions of Average Cyclomatic, Ration Comment To Code, and Number Of Statements metrics, highlighting differences in programming practices.

From an architectural perspective, blockchain-based applications have centralized elements, such as Web servers, connected to decentralized elements such as smart contracts. Finding the right balance between the different elements is a challenge and this has an impact on quality aspects of software such as safety, maintainability, performance, or cost. The study by Wessling et al. (2019) tackled the problem of analyzing the architectural blockchain tactics. In particular, the authors showed that common software design patterns could be not always beneficial and that the expected usage scenarios have a strong impact on the operational costs, claiming the need for further research to allow developers to make more informed architectural design decisions. Few measurements concern the level of decentralization that is obtained in practice. Gencer et al. (2018) proposed a measurement research on different metrics of decentralization related to Bitcoin and Ethereum. They estimated the network resources of nodes building a Falcon relay network[10] and the interconnection among them, the protocol requirements influencing the operation of nodes, and the robustness of the two systems against attacks. By using existing Internet measurement techniques, they concluded that neither Bitcoin nor Ethereum has rigorously better properties than the other.

4.3.2. Smart contract-specific metrics

While the first part of this section refers to several blockchain applications, subsequent studies refer to the Ethereum technology. In addition to studies that have been focused on calculating blockchain-specific metrics, other cases have focused on smart contract-specific metrics.

Smart contracts are programs executed on the blockchain. Since such smart contracts handle money, it is useful to have few failures and vulnerabilities, so it is important to provide developers with useful tools to measure the quality of smart contracts. Object-oriented (OO) metrics have been proposed for addressing this problem.

For instance, Hegedűs (2019) used OO metrics for estimating properties of the smart contracts written in Solidity. More specifically, a prototype tool called SolMet to measure OO metrics on Solidity source code files is implemented. Besides, SolMet has been used to characterize the complexity and size of 10,000 Solidity smart contracts, finding that smart contracts are usually short, uncomplicated and present few comments. As many libraries defined in Solidity files present similar functionalities, the author suggests exploiting external libraries and dependency management mechanisms for enabling code reuse. In a similar effort, Tonelli et al. (2018) compared the software metrics measured on Smart contracts (SC) and metrics extracted from

[10] https://www.falcon-net.org/

traditional software systems. They implemented a code parser to compute metrics like total lines of code to a specific blockchain address, the number of smart contracts inside a single address code, blank lines, the comment lines, the number of static calls to events, the number of modifiers, the number of functions, number of payable functions, the cyclomatic complexity as the simplest McCabe definition, the number of mappings to addresses for the dataset considered. Then they compare metrics that can also be calculated in traditional software like total lines of code, blank lines, comments and LOC. The authors found that smart contract lines of code is the metric that is closest at the statistical distribution of the corresponding metric in traditional software system. Also Pierro and Tonelli (2020) implement a parser called PASO to calculate metrics measuring properties of smart contracts. However, the metrics are Solidity language-specific, such as count of mapping, events, contracts, addresses. The advantage of this tool compared to the previous ones is that it does not require to install any additional software as it is web based and it is resilient to the updates of the Solidity language.

In addition to the source code metrics, in other cases, metrics of blockchain applications have been calculated. Blockchain could also be used in the health field to solve problems such as delayed communications, inefficient delivery of clinical reports and fragmented medical records. In existing literature, there are few guidelines for creating blockchain-based health apps. For this reason, Zhang et al. (2017) described a series of evaluation parameters from a technical point of view to address healthcare interoperability issues. Examples of metrics are scalability over large populations, interoperability, authentication.

Defining metrics is important both in the case of blockchain platforms and in the case of smart contracts. In particular, software metrics have been leveraged in software engineering research to compare blockchain and traditional systems, from different perspectives. Moreover, OO metrics have been computed on smart contracts to identify similarities and differences from more traditional programming languages. As the languages for writing smart contracts are new and constantly updated, there is still much to analyze.

Open challenges in Smart Contract Code Metrics: *guidelines for software developers;use of agile methods to develop software; a unique tool to calculate different metrics; identifying new language-specific metrics for writing smart contracts.*

4.4. Smart contract security

The aspects of the security that have been dealt with so far are: static analysis of smart contract code (Chinen et al., 2020), testing techniques for finding security flaws (Wohrer and Zdun, 2018; Atzei et al., 2016), and tools for identifying vulnerable smart contracts (Krupp and Rossow, 2018; Nikolić et al., 2018; Luu et al., 2016; Grech et al., 2018).

The identification of Ponzi schemes is a recurring topic in many analyzed papers (Bartoletti et al., 2020; Chen et al., 2019, 2018a) concerning the Ethereum technology. Ponzi schemes are not security issues related to smart contract implementation, but they are deliberate frauds made by scammers. The term Ponzi scheme indicates a form of financial fraud where, behind the promise of high profits, users deposit their money. Thus, victims believe that profits come from product sales or other financial revenue, while the source of funds is other victims. Thus, the business stands as long as new investors contribute with new funds. This pattern was invented by Charles Ponzi about a hundred years ago. Smart contract platforms like Ethereum offer new opportunities to scammers who can create "reliable" fraud through smart contracts. For instance, Bartoletti et al. (2020) proposed a complete survey of Ponzi schemes on Ethereum to evaluate behaviors and impacts from several perspectives. The analysis shows that the economic impact of the Ponzi schemes in smart contracts is limited, as only a small part of transactions is involved. However, the authors suggest a set of recommendations for users, such as advertisement checks, or contract's code and transaction logs inspection.

By collecting samples from real applications, Chen et al. (2019, 2018a) proposed an approach based on machine learning methods to detect smart contracts implementing Ponzi schemes (i.e., smart Ponzi schemes) deployed on the blockchain. In an analysis conducted on over 3000 smart contracts of Ethereum, they found 200 smart Ponzi schemes.

Given that smart contracts manage and transfer valuable assets, strong research efforts have been posed on investigating vulnerability problems in smart contracts (Krupp and Rossow, 2018; Wohrer and Zdun, 2018; Atzei et al., 2016; Nikolić et al., 2018; Luu et al., 2016). In this context, Atzei et al. (2016) surveyed the security vulnerabilities of Ethereum smart contracts and provided a taxonomy of common programming pitfalls leading to vulnerabilities. By using Grounded Theory, Wohrer and Zdun (2018) identified six design patterns to address the security issues of Solidity smart contracts. An example is the Checks-Effects-Interaction pattern, which describes how a function code should be structured to avoid side effects and unwanted execution behavior. Other patterns deal with problems related to lack of control of the smart contract after it has been distributed in Ethereum.

Shrivas et al. (2020) proposed a hybrid framework that protect against different threat types such as blockchain runtime environment, communication protocol, smart contract, consensus protocol. This framework can be applied to any type of blockchain platform.

Other researchers proposed tools aimed at identifying vulnerable Ethereum smart contracts, such as TEETHER (Krupp and Rossow, 2018), MAIAN (Nikolić et al., 2018), Oyente (Luu et al., 2016), MadMax (Grech et al., 2018). Bugs occurrence can cause even huge monetary losses, so it is important to find and fix them. TEETHER automatically creates exploits for a contract by inspecting its binary bytecode. An analysis performed on 38,757 unique Ethereum contracts, allowed researchers to find 815 vulnerable contracts. MAIAN support the specification of properties to be traced, and leverages interprocedural symbolic analysis and concrete validator to exhibit real exploits. Analyzing nearly one million contracts, MAIAN flagged thousands of contracts vulnerable at a high true positive rate. Similarly, Oyente is a symbolic execution tool for finding potential security bugs. Among nearly 20,000 existing Ethereum contracts, Oyente flagged 8833 of them as vulnerable.

Instead, MadMax uses static programming techniques to automatically detect gas-based vulnerabilities. Gas-based vulnerabilities exploit the unwanted behavior that occurs when a contract terminates the gas available to run. The results show that 81% of the samples analyzed have vulnerabilities.

Considering the increasing diffusion of Ethereum and the increasing value of capitalization, very little is known about the relationship between users and smart contracts. To fill this gap, Chen et al. (2018b) analyzed three main activities on Ethereum: the transfer of money, the creation of smart contracts, and the invocation of smart contracts, by using a graphical analysis. They proposed a new approach based on cross-graph analysis to address security problems, like attack forensics and anomaly detection.

While techniques based on static or dynamic analysis are usually applied to find security bugs in smart contracts, Huang

(2018) proposed a solution aimed at reducing the labor costs of the experts by exploiting image recognition for Ethereum smart contract.

Another type of attack is re-entrancy, which has the objective of stealing Ether, cryptocurrency used by Ethereum. Chinen et al. (2020) propose RA (Re-entrancy Analyzer), a static analysis tool that combines symbolic execution and equivalence control to analyze smart contract vulnerable to re-entrancy attacks.

Considering the different techniques of blockchain exploitation, Sayeed et al. (2020) classify them into 4 categories according to the logic of attack (consensus protocols, bugs in the smart contract, malware running in the operating system and fraudulent users). Then they analyze the vulnerabilities of smart contracts to determine the real impact. Analysis shows that the most frequent attacks are: DAO attack, Parity attack, king of Ether Trone, Govern Mental, and Dynamic libraries. The most used testing tools are: Fuse, Bug framework, Goatx Casino, Truffle framework, Mitx, Oyente framework.

Security is always an important issue to consider. In this section several aspects have been seen as possible fraud through the Ponzi scheme but also how to detect some of the most famous bugs such as Parity bug and DAO bug. In addition, recurring vulnerabilities and patterns have been identified in smart contracts, as well as methods to create and prevent exploits. However, there are still vulnerabilities that can cause monetary losses.

Open challenges in Smart Contract Security: *patterns to detect specific attacks; consolidate a taxonomy of smart contract vulnerabilities, searching for new ones; effective processes of security testing.*

4.5. Dapp performance measurement

The definition of benchmarks for Dapp (Dinh et al., 2018, 2017) and performances analysis (Rouhani and Deters, 2017; Pongnumkul et al., 2017; Thakkar et al., 2018) have been among the main topics of Dapp performance measurement.

It is assumed that recent private blockchain systems have been designed with greater attention to safety and performance. However, a framework that can analyze and compare the different systems to identify advantages and possible bottlenecks to solve is still missing. Dinh et al. (2018, 2017) developed a benchmarking framework, called BLOCKBENCH, for the comparison of blockchain systems such as Ethereum, Parity and Hyperledger in terms of data processing performance and workloads. BLOCKBENCH can be integrated via API into private blockchains and measures presentations in terms of throughput, latency, scalability and tolerance to errors.

Rouhani and Deters (2017) compared the performances in Ethereum when using two clients, Parity[11] and Geth[12] Geth is one of the first Ethereum client implemented in Go language, instead Parity is newer, written in the Rust language and is focused on efficiency. The results showed that, by using the same configuration, the Parity client is 89.8% faster than Geth. The reason is that Geth uses a diverse implementation approaches based on stack.

As opposed to public blockchains such as Ethereum, Hyperledger Fabric[13] is established as permissioned blockchain. Pongnumkul et al. (2017) analyzed the performance and limits of Hyperledger Fabric and Ethereum: Hyperledger Fabric achieves higher throughput and lower latency than Ethereum when workloads increase in terms of transactions. These differences get stronger when the number of transactions grows.

Instead, Thakkar et al. (2018) performed an empirical study to characterize the performance of Hyperledger Fabric and find bottlenecks. They proposed guidelines for the configuration of the parameters analyzed and identified the main performance bottlenecks.

While in previous studies in this section reference is made to specific blockchain technologies, in the following two reference is made to generic blockchain platforms.

Adding a blockchain in existing systems raises several issues. Wessling et al. (2018) proposed an approach to determine which architectural or system elements can benefit from the use of blockchain technology. After identifying participants, relationships of trust and interactions, they propose a hybrid architecture, which uses blockchain in existing software systems.

Using blockchain instead of traditional databases or protocols is an architectural choice that creates trade-offs in non-functional requirements. Yasaweerasinghelage et al. (2017) analyzed the latency of blockchain-based systems using modeling and simulation of architectural performance. The approaches used help to understand the differences between the cost model per blockchain compared to conventional systems.

Open challenges in Dapp Performance Measurement: *comprehensive frameworks to identify advantages and possible bottlenecks in different blockchain technologies and architectures; benchmarks to evaluate scalability, error tolerance, and impact of different consensus algorithms; approaches to estimate the impact of adopting blockchain-based technologies on functional and non-functional requirements.*

4.6. Blockchain application

With regard to blockchain-based applications, a number of issues have been addressed in different fields such as economics (Meng and Qian, 2018a,b; Suankaewmanee et al., 2018; Ranganthan et al., 2018; Clack et al., 2016; Jiang et al., 2019), payment systems (Manzoor et al., 2018; Zinca and Negrean, 2018), health (Delgado-Mohatar et al., 2019), blockchain system (Hamida et al., 2017; Amoordon and Rocha, 2019; Bez et al., 2019), query and database (Xu et al., 2019), and IoT applications (Hinckeldeyn and Jochen, 2018; Wang et al., 2019c).

Applications have been proposed to bring innovation in business operations, corporate logistics, but also financial services and m-commerce applications. In this section, most of the work refers to applications that can use blockchain technology in general (Meng and Qian, 2018a,b; Suankaewmanee et al., 2018; Jiang et al., 2019; Chang et al., 2019; Pradeepkumar et al., 2018; Clack et al., 2016; Wang et al., 2019c; Liu, 2018; Zhang et al., 2019; Xu et al., 2019; Martens and Maalej, 2018), only a part is related to Ethereum (Nizamuddin et al., 2019; Ranganthan et al., 2018; Aung and Tantidham, 2017; Manzoor et al., 2018; Zinca and Negrean, 2018; Delgado-Mohatar et al., 2019; Hukkinen et al., 2019; Hinckeldeyn and Jochen, 2018; Kfoury and Khoury, 2018; Bez et al., 2019; Bragagnolo et al., 2018b; McConaghy et al., 2016) and enterprise blockchain (Hamida et al., 2017).

For example, Meng and Qian (2018a,b) analyzed projects in the business sector and propose an evaluation framework called DelivChain. It is useful to increase real-time predictive delivery performance in supply chain management and accuracy. Nizamuddin et al. (2019) proposed a blockchain-based solution and a framework for document sharing and version control to facilitate multi-user collaboration and track changes.

11 https://www.parity.io/
12 https://geth.ethereum.org/
13 https://www.hyperledger.org/use/fabric

Suankaewmanee et al. (2018) proposed MobiChain, a mobile application that uses blockchain technology to protect e-commerce transactions. Mining processes are performed on mobile devices through an Android module. Jiang et al. (2019) have designed a corporate protocol to preserve private information, such as identity, address and phone number, using smart contracts in the e-commerce trading phase. In a similar way, Ranganthan et al. (2018) proposed an application that uses the blockchain to provide support platforms for online purchases, such as eBay. This application could be used to block merchants, pay commissions, sell a product, and preserve the privacy of user data. Instead, Chang et al. (2019) showed the advantages of using blockchain for monitoring the logistics status in real-time and reducing the costs associated with cash backlogs in practical business system design.

In order to promote the adoption of blockchain in digitization processes, it is necessary to understand which rules, mechanisms and specifications of the regulations are best suited. To fill this gap, Pradeepkumar et al. (2018) proposed a modeling approach that supports automated analysis of human-readable regulatory representations. In order to support the financial services in adoption of legally-enforceable smart contracts, Clack et al. (2016) analyzed potential formats for the archiving and transmission of intelligent legal agreements.

Aung and Tantidham (2017) applied the blockchain technology for checking each transaction history and set the policies for managing the transactions of a smart home system.

Within the context of payment systems, Manzoor et al. (2018) proposed a cashless payment scheme for remote villages based on keeping a log of transactions verifiable in a distributed way. Similarly, Zinca and Negrean (2018) developed a road tax payment application using the Ethereum platform.

Instead, Delgado-Mohatar et al. (2019) studied the feasibility of blockchain-based biometric systems with the storage of biometric models.

Another important area to take into account is the Internet of Things (IoT). Wang et al. (2019c) proposed a comprehensive survey of IoT-oriented Blockchain technologies. Instead, Hinckeldeyn and Jochen (2018) presented an intelligent storage container that uses Ethereum-based smart contract.

Similarly, Hukkinen et al. (2019) proposed a blockchain-based application designed to conduct electricity microtransactions in a nanogrid environment.

Blockchain technologies can also be leveraged for enabling more secure services over the network. For instance, Voice over Long Term Evolution (Volte) technology provides standards for real-time services such as voice and video on LTE mobile network. The security implementation in VoLTE is End-to-Access (e2a), which means that the sessions are only encrypted between the mobile terminals and the IP Media Subsystem (IMS) network. Kfoury and Khoury (2018), to achieve End-to-End (e2e) protection of Volte media, proposed to create public and private key pairs for Volte user equipment and store public keys in the Ethereum blockchain. The authors show that this solution has minimal impact on the IMS network and the setup time is negligible compared to the original VoLTE setup time.

In addition to such applications, another study has tried to help beginners understanding key blockchain concepts. Liu (2018) proposes ChainTutor, a Java application that uses a graphical interface instead of long text descriptions in which users can generate keys, hashes, transactions, blocks, wallets, and add blocks. This application can be used by instructors in classroom environments when they introduce blockchain notions.

Bez et al. (2019) evaluated the current implementation of the blockchain application and the related improvement proposals. Instead, Hamida et al. (2017) analyzed enterprise blockchains focusing on main components, technologies and applications. They provided a taxonomy of applications and use cases. They suggested to improve real time analysis, data privacy, security, and scalability. Another type of blockchain is Tendermint, a Byzantine Fault Tolerant (BFT) application-based blockchain (Amoordon and Rocha, 2019). Unlike Ethereum, that allows to run several heterogeneous smart contracts on the same blockchain, Tendermint uses one application per blockchain.

A different context from the previous ones is that which regards the use of the databases and the associated queries. It is important to have search methods to directly access the blocks, that continue to grow over time. To this aim, Bragagnolo et al. (2018b) presented the Ethereum Query Language (EQL), a query language for retrieving information from the blockchain with queries similar to SQL. Considering the databases, BigchainDB (McConaghy et al., 2016) is a large-scale decentralized blockchain database. The characteristics of BigchainDB are: 1 million writes per second throughput, storing petabytes of data, sub-second latency, user-friendly and efficient queries, authorization system for public and private blockchains. Other researchers have studied the problem of authenticating range queries for databases stored in the hybrid storage blockchain (Zhang et al., 2019).

Another application is originChain, a blockchain-based traceability system that restructures the current system by replacing the central database with blockchain (Xu et al., 2019).

To solve the problem of central authorities, which can influence review processes, Martens and Maalej (2018) presented ReviewChain, a decentralized review approach that (i) uses blockchain technologies, decentralized apps and storage and (ii) allows users to send and retrieve reviews that have not been tampered with.

Open challenges in Blockchain Application: *integration between blockchain and existing systems; approaches for assessing costs and benefits of blockchain integration.*

5. Discussion

By analyzing the current literature on blockchain-oriented software engineering, it emerges that most of the research focuses on the Ethereum platform with a few exceptions as Bitcoin and Hyperledger. Consequently, while there is a large convergence of results concerning this specific platform, replications and new studies on the other platforms are necessary for generalizing the results. The size of datasets used in the experiments varies in a wide interval, spanning from a few units up to hundreds of thousands of smart contracts, with a strong prevalence of studies using very huge datasets. This gives greater external validity to the results of the papers and put large datasets at the community's disposal.

More specifically, the research community paid a lot of attention to the testing of smart contracts, but the main purpose of the existing literature is to find bugs and vulnerabilities. Surprisingly, approaches for conducting specific kinds of tests, like integration and regression, are almost completely unexplored. In particular, it would be interesting to understand how to adapt the existing testing techniques to a smart contract or a Dapp. Moreover, while fuzz testing has been largely studied, other techniques related to test-case generation and mutation testing are still poorly investigated. For building a complete body of knowledge of Software Engineering in this area, these steps are inescapable.

Existing studies on smart contract code analysis head to (i) locate security and quality problems, (ii) detect clones, for dealing with reuse and the protection of intellectual property, and (iii) identify cost-effective code. Since on specific blockchain architectures (e.g., Ethereum) Dapp execution is related to gas

consumption, it would be helpful to have techniques for identifying patterns of code that may affect the gas consumption, jointly with automated approaches for making the code cost-effective. Furthermore, code measurement is a fundamental component for gathering quantitative characterization of the code, which could lead to more precise planning, prediction, and projections of the overall software lifecycle. This area needs further developments concerning smart contracts and Dapp. In particular, the examined methods of code analysis are mainly designed for Solidity and Go, with the need of posing efforts towards a stronger generalization.

Much work has been devoted to the security of smart contracts, resulting in defining specific vulnerabilities for smart contracts and tools for finding vulnerabilities in smart contracts. However, smart contract security is still at the beginning and many other issues deserve attention. Even if many researchers identified vulnerabilities, a reference taxonomy that organizes and collects vulnerabilities according to criteria like impact, scope, effect, cost of exploitation, and fix is still missing. Similarly, we still lack a comprehensive catalog of attack patterns to smart contracts. Consequently, paramount is to systematize the process of smart contract security testing, including the practices, the tools, and the procedures to be followed.

As previously remarked, the performances of a blockchain are a widely acknowledged problem, especially by industry: till now research produced benchmarks and analysis frameworks for measuring performances. We should make headway in producing comprehensive frameworks to identify possible bottlenecks in different blockchain technologies and architectures. Scalability and availability are two key factors for the success of a Dapp: more complete benchmarks are necessary for helping developers to (i) evaluate (and improve) error tolerance, (ii) avoid performance degradation when the number of transactions increases, and (iii) measure how the different consensus algorithms affect the responsiveness of a Dapp.

The blockchain is becoming an enabling technology and software architects could choose whether to adopt it or not. For this reason, methods for assessing how a blockchain could modify the system requirements and properties are needed to drive the architectural choices. Literature reports many experiences of using blockchain in software systems. It should be interesting to investigate which issues are raised when integrating blockchain in existing systems, and consequently provide approaches that could support such an integration and measure costs and risks. An effort is necessary for making such methods independent from the specific technology the blockchain is implemented in.

6. Threats to validity

There are several threats to validity that may arise while conducting a systematic literature review (Zhou et al., 2016). A common threat could be represented by an ineffective strategy for searching relevant studies or selecting sources of information. To mitigate this concern and retrieve as many relevant documents as possible, we used simple search strings and queried four different digital libraries. In particular, we avoided restricting the sources of information to certain publishers, journals, or conferences with the specific aim of adequately covering the breadth of the investigated research field. We also cross-checked the completeness of searches and validated the suitability of each study for inclusion.

Although we tried to minimize the possibility of missing important studies, it could have happened that the title, abstract, or keywords of some relevant studies did not match with the search strings used for querying the digital libraries. To alleviate this issue, we manually inspected the references of the extracted studies to identify papers that were missed during the initial search but had to be included in the analysis. Considering that the topic is quite new, we believe that the number of relevant articles which have possibly been missed could be sufficiently small and, thus, would only marginally influence the findings of our study.

Another threat to validity can be related to the bias of authors when selecting the articles. To cope with this threat and reduce the bias while filtering the papers, we defined a selection protocol with clear inclusion and exclusion criteria. For avoiding the misinterpretations in the titles or abstracts of the retrieved documents that could cause the exclusion of relevant papers, before deciding whether a specific document was relevant or not, introduction and conclusion sections were carefully examined. Besides, retrieved documents were discussed by authors and included in the review only after achieving the consensus.

7. Conclusions

Given the growing spread of blockchain platforms and the use of smart contracts in different sectors, it is essential to have a picture of the current state of the technology and analyze in which areas that technology is bringing innovation. From the analysis carried out, we observed that the review of existing literature is focused only on specific themes, such as security or blockchain applications. However, this technology can be improved by introducing software engineering practices. Based on these motivations, we carried out a software engineering oriented literature review, to better understand the techniques, practices, and tools specifically conceived to address the issues posed by the development of blockchain-based software and identify the challenges that are still open.

To carry out this analysis, after inspecting the different papers identified, on the basis of the defined inclusion and exclusion criteria, 96 papers were selected. Six topics of interest have been defined (i.e., smart contract testing, smart contract code analysis, smart contract metrics, smart contract security, Dapp performance, and blockchain application). For each of these topics, we also identified open challenges needing further research.

The analysis highlighted that further investigation is needed to understand how to apply traditional testing techniques to blockchain-oriented software, but also methods to measure specific code metrics enabling code optimization. In addition, guidelines are needed for developers or even the development of a new programming language to simplify the creation and understanding of smart contracts. As for security, patterns could be identified to detect specific attacks but also create a taxonomy with the most common attacks to help developers prevent them. Considering the different blockchain platforms, a framework would also be needed to compare them. This comparison, based on the advantages offered by each platform, would be useful to help select the one that is most suitable when dealing with specific scenarios.

CRediT authorship contribution statement

Anna Vacca: Methodology, Investigation, Validation, Visualization, Data curation, Writing - original draft, Writing - review & editing. **Andrea Di Sorbo:** Conceptualization, Methodology, Investigation, Writing - original draft, Writing - review & editing. **Corrado A. Visaggio:** Conceptualization, Methodology, Investigation, Visualization, Writing - original draft, Writing - review & editing. **Gerardo Canfora:** Writing - review & editing, Supervision.

Declaration of competing 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.

References

Aldweesh, A., Alharby, M., van Moorsel, A., 2018a. Performance benchmarking for Ethereum opcodes. In: 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications. AICCSA, IEEE, pp. 1–2.

Aldweesh, A., Alharby, M., Solaiman, E., van Moorsel, A., 2018b. Performance benchmarking of smart contracts to assess miner incentives in Ethereum. In: 2018 14th European Dependable Computing Conference. EDCC, IEEE, pp. 144–149.

Amoordon, A., Rocha, H., 2019. Presenting tendermint: Idiosyncrasies, weaknesses, and good practices. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 44–49.

Anjana, P.S., Kumari, S., Peri, S., Rathor, S., Somani, A., 2019. An efficient framework for optimistic concurrent execution of smart contracts. In: 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. PDP, IEEE, pp. 83–92.

Atzei, N., Bartoletti, M., Cimoli, T., 2016. A survey of attacks on Ethereum smart contracts.. IACR Cryptol. ePrint Archive 2016, 1007.

Atzei, N., Bartoletti, M., Lande, S., Yoshida, N., Zunino, R., 2019. Developing secure bitcoin contracts with BitML. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1124–1128.

Aung, Y.N., Tantidham, T., 2017. Review of Ethereum: Smart home case study. In: 2017 2nd International Conference on Information Technology. INCIT, IEEE, pp. 1–4.

Bach, L., Mihaljevic, B., Zagar, M., 2018. Comparative analysis of blockchain consensus algorithms. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics. MIPRO, IEEE, pp. 1545–1550.

Bartoletti, M., Carta, S., Cimoli, T., Saia, R., 2020. Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact. Future Gener. Comput. Syst. 102, 259–277.

Bartoletti, M., Pompianu, L., 2017. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In: International Conference on Financial Cryptography and Data Security. Springer, pp. 494–509.

Bartoletti, M., Zunino, R., 2018. BitML: A calculus for bitcoin smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 83–100.

Bez, M., Fornari, G., Vardanega, T., 2019. The scalability challenge of Ethereum: An initial quantitative analysis. In: 2019 IEEE International Conference on Service-Oriented System Engineering. SOSE, IEEE, pp. 167–176.

Bragagnolo, S., Rocha, H., Denker, M., Ducasse, S., 2018a. Smartinspect: solidity smart contract inspector. In: 2018 International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 9–18.

Bragagnolo, S., Rocha, H., Denker, M., Ducasse, S., 2018b. Ethereum query language. In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 1–8.

Casino, F., Dasaklis, T.K., Patsakis, C., 2019. A systematic literature review of blockchain-based applications: Current status, classification and open issues. Telemat. Inform. 36, 55–81.

Chakraborty, P., Shahriyar, R., Iqbal, A., Bosu, A., 2018. Understanding the software development practices of blockchain projects: A survey. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM 2018, Oulu, Finland, October 11–12, 2018, pp. 28:1–28:10. http://dx.doi.org/10.1145/3239235.3240298.

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

Chang, S.E., Chen, Y.-C., Lu, M.-F., 2019. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. Technol. Forecast. Soc. Change 144, 1–11.

Chen, T., Feng, Y., Li, Z., Zhou, H., Luo, X., Li, X., Xiao, X., Chen, J., Zhang, X., 2020a. GasChecker: Scalable analysis for discovering gas-inefficient smart contracts. IEEE Trans. Emerg. Top. Comput..

Chen, T., Li, X., Luo, X., Zhang, X., 2017. Under-optimized smart contracts devour your money. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 442–446.

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

Chen, W., Zheng, Z., Cui, J., Ngai, E., Zheng, P., Zhou, Y., 2018a. Detecting Ponzi schemes on Ethereum: Towards healthier blockchain technology. In: Proceedings of the 2018 World Wide Web Conference, pp. 1409–1418.

Chen, W., Zheng, Z., Ngai, E.C.-H., Zheng, P., Zhou, Y., 2019. Exploiting blockchain data to detect smart ponzi schemes on Ethereum. IEEE Access 7, 37575–37586.

Chen, T., Zhu, Y., Li, Z., Chen, J., Li, X., Luo, X., Lin, X., Zhange, X., 2018b. Understanding Ethereum via graph analysis. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, pp. 1484–1492.

Chepurnoy, A., Rathee, M., 2018. Checking laws of the blockchain with property-based testing. In: 2018 International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 40–47.

Chinen, Y., Yanai, N., Cruz, J.P., Okamura, S., 2020. Hunting for re-entrancy attacks in Ethereum smart contracts via static analysis. arXiv preprint arXiv:2007.01029.

Clack, C.D., Bakshi, V.A., Braine, L., 2016. Smart contract templates: essential requirements and design options. Comput. Res. Repos.

Coblenz, M., Sunshine, J., Aldrich, J., Myers, B., 2019. Smarter smart contract development tools. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB, IEEE, pp. 48–51.

Conoscenti, M., Vetro, A., De Martin, J.C., 2016. Blockchain for the Internet of Things: A systematic literature review. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications. AICCSA, IEEE, pp. 1–6.

Delgado-Mohatar, O., Fierrez, J., Tolosana, R., Vera-Rodriguez, R., 2019. Biometric template storage with blockchain: a first look into cost and performance tradeoffs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.

Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., Hierons, R., 2018. Smart contracts vulnerabilities: a call for blockchain software engineering? In: 2018 International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 19–25.

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

Dinh, T.T.A., Liu, R., Zhang, M., Chen, G., Ooi, B.C., Wang, J., 2018. Untangling blockchain: A data processing view of blockchain systems. IEEE Trans. Knowl. Data Eng. 30 (7), 1366–1385.

Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.-L., 2017. BLOCKBENCH: A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 1085–1100.

Drosatos, G., Kaldoudi, E., 2019. Blockchain applications in the biomedical domain: A scoping review. Comput. Struct. Biotechnol. J.

Ellul, J., Pace, G.J., 2018. Runtime verification of Ethereum smart contracts. In: 2018 14th European Dependable Computing Conference. EDCC, IEEE, pp. 158–163.

Feist, J., Grieco, G., Groce, A., 2019. Slither: A static analysis framework for smart contracts. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB, IEEE, pp. 8–15.

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

Gao, Z., Jiang, L., Xia, X., Lo, D., Grundy, J., 2020. Checking smart contracts with structural code embedding. IEEE Trans. Softw. Eng.

Gencer, A.E., Basu, S., Eyal, I., Van Renesse, R., Sirer, E.G., 2018. Decentralization in bitcoin and Ethereum networks. In: International Conference on Financial Cryptography and Data Security. Springer, pp. 439–457.

Golosova, J., Romanovs, A., 2018. The advantages and disadvantages of the blockchain technology. In: 2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering. AIEEE, IEEE, pp. 1–6.

Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., Smaragdakis, Y., 2018. MadMax: Surviving out-of-gas conditions in Ethereum smart contracts. Proc. ACM Prog. Lang. 2 (OOPSLA), 1–27.

Hamida, E.B., Brousmiche, K.L., Levard, H., Thea, E., 2017. Blockchain for enterprise: Overview, opportunities and challenges.

Hartmann, F., Grottolo, G., Wang, X., Lunesu, M.I., 2019. Alternative fundraising: Success factors for blockchain-based vs. conventional crowdfunding. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 38–43.

Hegedűs, P., 2019. Towards analyzing the complexity landscape of solidity based Ethereum smart contracts. Technologies 7 (1), 6.

Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth, D., Moore, B., Park, D., Zhang, Y., Stefanescu, A., et al., 2018. KEVM: A complete formal semantics of the Ethereum virtual machine. In: 2018 IEEE 31st Computer Security Foundations Symposium. CSF, IEEE, pp. 204–217.

Hinckeldeyn, J., Jochen, K., 2018. (Short paper) Developing a smart storage container for a blockchain-based supply chain application. In: 2018 Crypto Valley Conference on Blockchain Technology. CVCBT, IEEE, pp. 97–100.

Huang, T.H.-D., 2018. Hunting the Ethereum smart contract: Color-inspired inspection of potential attacks. arXiv preprint arXiv:1807.01868.

Hukkinen, T., Mattila, J., Smolander, K., Seppala, T., Goodden, T., 2019. Skimping on gas–reducing Ethereum transaction costs in a blockchain electricity market application. In: Proceedings of the 52nd Hawaii International Conference on System Sciences.

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

Jiang, Y., Wang, C., Wang, Y., Gao, L., 2019. A privacy-preserving e-commerce system based on the blockchain technology. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 50–55.

Kalra, S., Goel, S., Dhawan, M., Sharma, S., 2018. ZEUS: Analyzing Safety of Smart Contracts. In: NDSS, pp. 1–12.

Kfoury, E.F., Khoury, D.J., 2018. Secure end-to-end volte based on Ethereum blockchain. In: 2018 41st International Conference on Telecommunications and Signal Processing. TSP, IEEE, pp. 1–5.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Citeseer.

Kondo, M., Oliva, G.A., Jiang, Z.M.J., Hassan, A.E., Mizuno, O., 2020. Code cloning in smart contracts: A case study on verified contracts from the Ethereum blockchain platform. Empir. Softw. Eng. 1–59.

Krupp, J., Rossow, C., 2018. TEETHER: Gnawing at Ethereum to automatically exploit smart contracts. In: 27th {USENIX} Security Symposium. {USENIX} Security 18, pp. 1317–1333.

Liu, X., 2018. A small java application for learning blockchain. In: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference. IEMCON, IEEE, pp. 1271–1275.

Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B., Roscoe, B., 2018. ReGuard: finding reentrancy bugs in smart contracts. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion. ICSE-Companion, IEEE, pp. 65–68.

Luu, L., Chu, D.-H., Olickel, H., Saxena, P., Hobor, A., 2016. Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 254–269.

Manzoor, A., Hu, Y., Liyanage, M., Ekparinya, P., Thilakarathna, K., Jourjon, G., Seneviratne, A., Kanhere, S., Ylianttila, M.E., 2018. A delay-tolerant payment scheme on the Ethereum blockchain. In: 2018 IEEE 19th International Symposium on "a World of Wireless, Mobile and Multimedia Networks". WoWMoM, IEEE, pp. 14–16.

Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G., Tigano, D., 2020. Design patterns for gas optimization in Ethereum. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 9–15.

Marchesi, M., Marchesi, L., Tonelli, R., 2018. An agile software engineering method to design blockchain applications. In: Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia, pp. 1–8.

Martens, D., Maalej, W., 2018. ReviewChain: Untampered product reviews on the blockchain. In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 40–43.

Mavridou, A., Laszka, A., 2018. Designing secure Ethereum smart contracts: A finite state machine based approach. In: International Conference on Financial Cryptography and Data Security. Springer, pp. 523–540.

McConaghy, T., Marques, R., Müller, A., De Jonghe, D., McConaghy, T., McMullen, G., Henderson, R., Bellemare, S., Granzotto, A., 2016. BigchainDB: A scalable blockchain database. white paper, BigChainDB.

Meng, M.H., Qian, Y., 2018a. A blockchain aided metric for predictive delivery performance in supply chain management. In: 2018 IEEE International Conference on Service Operations and Logistics, and Informatics. SOLI, IEEE, pp. 285–290.

Meng, M.H., Qian, Y., 2018b. The Blockchain Application in Supply Chain Management: Opportunities, Challenges and Outlook. Tech. rep., EasyChair.

Nakamoto, S., 2008. Bitcoin whitepaper. Online; https://bitcoin.org/bitcoin.pdf. (Accessed 18 October 2020).

Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A., 2018. Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp. 653–663.

Nizamuddin, N., Salah, K., Azad, M.A., Arshad, J., Rehman, M., 2019. Decentralized document version control using Ethereum blockchain and IPFS. Comput. Electr. Eng. 76, 183–197.

Norvill, R., Pontiveros, B.B.F., State, R., Awan, I., Cullen, A., 2017. Automated labeling of unknown contracts in Ethereum. In: 2017 26th International Conference on Computer Communication and Networks. ICCCN, IEEE, pp. 1–6.

Oliva, G.A., Hassan, A.E., Jiang, Z.M.J., 2020. An exploratory study of smart contracts in the Ethereum blockchain platform. Empir. Softw. Eng. 25 (3), 1864–1904.

Ortu, M., Orrú, M., Destefanis, G., 2019. On comparing software quality metrics of traditional vs blockchain-oriented software: An empirical study. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 32–37.

Payette, J., Schwager, S., Murphy, J., 2017. Characterizing the Ethereum address space.

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

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

Pierro, G.A., Tonelli, R., 2020. PASO: A web-based parser for solidity language analysis. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 16–21.

Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S., 2017. Performance analysis of private blockchain platforms in varying workloads. In: 2017 26th International Conference on Computer Communication and Networks. ICCCN, IEEE, pp. 1–6.

Porru, S., Pinna, A., Marchesi, M., Tonelli, R., 2017. Blockchain-oriented software engineering: Challenges and new directions. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion. ICSE-C, IEEE, pp. 169–171.

Pradeepkumar, D., Singi, K., Kaulgud, V., Podder, S., 2018. Evaluating complexity and digitizability of regulations and contracts for a blockchain application design. In: 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB, IEEE, pp. 25–29.

Ranganthan, V.P., Dantu, R., Paul, A., Mears, P., Morozov, K., 2018. A decentralized marketplace application on the Ethereum blockchain. In: 2018 IEEE 4th International Conference on Collaboration and Internet Computing. CIC, IEEE, pp. 90–97.

Rocha, H., Ducasse, S., 2018. Preliminary steps towards modeling blockchain oriented software. In: 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB, IEEE, pp. 52–57.

Rouhani, S., Deters, R., 2017. Performance analysis of Ethereum transactions in private blockchain. In: 2017 8th IEEE International Conference on Software Engineering and Service Science. ICSESS, IEEE, pp. 70–74.

Samreen, N.F., Alalfi, M.H., 2020. Reentrancy vulnerability identification in Ethereum smart contracts. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 22–29.

Sayeed, S., Marco-Gisbert, H., Caira, T., 2020. Smart contract: Attacks and protections. IEEE Access 8, 24416–24427.

Shrivas, M.K., Yeboah, T., Brunda, S., 2020. Hybrid security framework for blockchain platforms. In: 2020 First International Conference on Power, Control and Computing Technologies. ICPC2T, IEEE, pp. 339–347.

Suankaewmanee, K., Hoang, D.T., Niyato, D., Sawadsitang, S., Wang, P., Han, Z., 2018. Performance analysis and application of mobile blockchain. In: 2018 International Conference on Computing, Networking and Communications. ICNC, IEEE, pp. 642–646.

Taylor, P.J., Dargahi, T., Dehghantanha, A., Parizi, R.M., Choo, K.-K.R., 2019. A systematic literature review of blockchain cyber security. Digit. Commun. Netw.

Thakkar, P., Nathan, S., Viswanathan, B., 2018. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS, IEEE, pp. 264–276.

Tian, G., Wang, Q., Zhao, Y., Guo, L., Sun, Z., Lv, L., 2020. Smart contract classification with a bi-LSTM based approach. IEEE Access 8, 43806–43816.

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

Tonelli, R., Destefanis, G., Marchesi, M., Ortu, M., 2018. Smart contracts software metrics: a first study. arXiv preprint arXiv:1802.01517.

Tonelli, R., Lunesu, M.I., Pinna, A., Taibi, D., Marchesi, M., 2019. Implementing a microservices system with blockchain smart contracts. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 22–31.

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

Wang, J., Wang, S., Guo, J., Du, Y., Cheng, S., Li, X., 2019a. A summary of research on blockchain in the field of intellectual property. Procedia Comput. Sci. 147, 191–197.

Wang, X., Wu, H., Sun, W., Zhao, Y., 2019b. 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.

Wang, X., Zha, X., Ni, W., Liu, R.P., Guo, Y.J., Niu, X., Zheng, K., 2019c. Survey on blockchain for Internet of Things. Comput. Commun. 136, 10–29.

Wessling, F., Ehmke, C., Hesenius, M., Gruhn, V., 2018. How much blockchain do you need? Towards a concept for building hybrid dapp architectures. In: 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB, IEEE, pp. 44–47.

Wessling, F., Ehmke, C., Meyer, O., Gruhn, V., 2019. Towards blockchain tactics: Building hybrid decentralized software architectures. In: 2019 IEEE International Conference on Software Architecture Companion. ICSA-C, IEEE, pp. 234–237.

Wohrer, M., Zdun, U., 2018. Smart contracts: Security patterns in the Ethereum ecosystem and solidity. In: 2018 International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 2–8.

Wöhrer, M., Zdun, U., 2020. Domain specific language for smart contract development.

Wu, Z., Zhang, J., Gao, J., Li, Y., Li, Q., Guan, Z., Chen, Z., 2020. Kaya: A testing framework for blockchain-based decentralized applications. arXiv preprint arXiv:2006.01476.
Xu, X., Lu, Q., Liu, Y., Zhu, L., Yao, H., Vasilakos, A.V., 2019. Designing blockchain-based applications a case study for imported product traceability. Future Gener. Comput. Syst. 92, 399–406.
Yamashita, K., Nomura, Y., Zhou, E., Pi, B., Jun, S., 2019. Potential risks of hyperledger fabric smart contracts. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering. IWBOSE, IEEE, pp. 1–10.
Yasaweerasinghelage, R., Staples, M., Weber, I., 2017. Predicting latency of blockchain-based systems using architectural modelling and simulation. In: 2017 IEEE International Conference on Software Architecture. ICSA, IEEE, pp. 253–256.
Ye, J., Ma, M., Peng, T., Peng, Y., Xue, Y., 2019. Towards automated generation of bug benchmark for smart contracts. In: 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops. ICSTW, IEEE, pp. 184–187.
Zhang, P., Walker, M.A., White, J., Schmidt, D.C., Lenz, G., 2017. Metrics for assessing blockchain-based healthcare decentralized apps. In: 2017 IEEE 19th International Conference on E-Health Networking, Applications and Services. Healthcom, IEEE, pp. 1–4.
Zhang, C., Xu, C., Xu, J., Tang, Y., Choi, B., 2019. GEMˆ 2-Tree: A gas-efficient structure for authenticated range queries in blockchain. In: 2019 IEEE 35th International Conference on Data Engineering. ICDE, IEEE, pp. 842–853.
Zhang, P., Yu, J., Ji, S., 2020. ADF-GA: Data flow criterion based test case generation for Ethereum smart contracts.
Zhou, X., Jin, Y., Zhang, H., Li, S., Huang, X., 2016. A map of threats to validity of systematic literature reviews in software engineering. In: 23rd Asia-Pacific Software Engineering Conference. APSEC 2016, Hamilton, New Zealand, December 6–9, 2016, pp. 153–160.
Zinca, D., Negrean, V.-A., 2018. Development of a road tax payment application using the Ethereum platform. In: 2018 International Symposium on Electronics and Telecommunications. ISETC, IEEE, pp. 1–4.
Zou, W., Lo, D., Kochhar, P.S., Le, X.-B.D., Xia, X., Feng, Y., Chen, Z., Xu, B., 2019. Smart contract development: Challenges and opportunities. IEEE Trans. Softw. Eng..

Anna Vacca is a Ph.D. student in Software Engineering at the Department of Engineering of the University of Sannio, Italy. She obtained B.Sc. in Computer Engineering from the University of Sannio, Italy, in 2014, while, in 2017, she received M.Sc. in Computer Engineering from the same university. Her research interests include blockchain applications, software maintenance, and software security.

Andrea Di Sorbo is a research fellow at the University of Sannio, Italy. He received the Ph.D. degree in information technology from the University of Sannio, in 2018. His research interests include software maintenance and evolution, mining software repositories, empirical software engineering, text analysis and software security and privacy. He coauthored several papers appeared in flagship international conferences (ICSE, FSE, ASE) and journals (TSE, JSS, IST, JSEP). He serves and has served as review editor and guest associate editor for Frontiers in Big Data, guest editor for the Information and Software Technology journal, and reviewer for several journals in the field of software engineering, as Transactions on Software Engineering, edited by IEEE, Journal of Software: Evolution and Processes edited by Wiley, and the Empirical Software Engineering journal edited by Springer. He is also a program committee member of some international conferences (ARES, MOBILESoft, SEAA).

Corrado Aaron Visaggio is an associate professor of CyberSecurity at the Department of Engineering of University of Sannio. He is chair of the node of University of Sannio for the CINI National Cyber Security Lab. He is the scientific coordinator of several projects funded by firms operating in CyberSecurity, concerning malware analysis, vulnerability assessment, and data protection. He is also among the founders of the academic spin-off SER&Practice. He serves in the Editorial Board of the International Journal of Computer Virology and Hacking techniques (Springer), as associate editor in Frontiers in Big Data, and in several Program Committees (MALWARE, ARES, SECRYPT, SEKE, ITASEC, FoRSE, DATA, Hufo, MobiSys, WETSOM, ISSRE); he was also the workshop chair of WETSOM and WMA. His main research interests are: malware analysis, data privacy and protection, software security, empirical software engineering.

Gerardo Canfora is a professor of computer science at the School of Engineering of the University of Sannio, Italy. He serves on the program and organizing committees of a number of international conferences. He was general chair of WCRE06 and CSMR03, and program cochair of ICSE15, WETSoM12 and 10, ICSM01 and 07, IWPSE05, CSMR04 and IWPC97. He is co-editor of the Journal of Software: Evolution and Processes. Canfora authored 200 research papers; his research interests include software maintenance and evolution, security and privacy, empirical software engineering, and service-oriented computing.