Challenges and Opportunities in Smart Contract Development on the Ethereum Virtual Machine: A Systematic Literature Review

Gislainy Crisostomo Velasco[1], **Noeli Antonia Pimentel Vaz**[1,2], **Sergio T. Carvalho**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia – GO – Brasil

[2]Universidade Estadual de Goiás (UEG)
Caixa Postal 459 – 75220-400 – Anápolis – GO – Brasil

gislainycrisostomo@discente.ufg.br, noeli@ueg.br,

sergio@inf.ufg.br

Abstract. *The development of smart contracts presents significant challenges compared to traditional software development, such as the immutability of the blockchain. This paper presents a systematic literature review (SLR) that aims to understand the limitations and challenges faced by smart contract developers on the Ethereum Virtual Machine (EVM). Among the main challenges identified are language restrictions, infrastructure limitations, and the lack of sufficient information on interface patterns and implementation specifications. Existing proposals are difficult to understand, with complex formal verifications that require advanced technical knowledge. Additionally, the scarcity of accessible educational resources for training new smart contract developers represents a major challenge to be overcome. In this regard, the SLR seeks to identify opportunities for improvement and innovation in the field, as well as validation and evaluation strategies to make the smart contract development process more efficient and secure.*

1. Introduction

Smart contracts are one of the key innovations of blockchain technology, enabling the creation of new forms of decentralized applications with characteristics such as immutability, transparency, and privacy [Angelis and Ribeiro da Silva 2019, Khan et al. 2019]. The Ethereum Virtual Machine (EVM) is responsible for enabling the writing of high-level code instructions and their autonomous execution on the blockchain.

Despite the advantages offered by smart contracts, their development still faces considerable technical challenges, including the complexity of the programming language and the lack of supporting tools. Throughout the process of conception and implementation, smart contract developers face several challenges, with a critical issue being the creation of immutable smart contracts [Zeng et al. 2022, Feist et al. 2019, Li et al. 2021, Dharanikota et al. 2021, Jiao et al. 2020a]. This requires a differentiated approach to writing, validating, and implementing an application since correcting errors in deployed code is a complex. Additionally, the understanding of smart contracts by non-technical users is a significant challenge [Qasse et al. 2021], as the relationships between elements and available resources are not presented in a clear and visually understandable way.

The development of smart contracts on the EVM platform requires knowledge of both the mechanism's operation and the high-level language used to write smart contracts. However, the programming language commonly used to write smart contracts, such as Solidity, can present challenges for developers, especially beginners, leading to errors and vulnerabilities in smart contracts [Garamvölgyi et al. 2018]. Additionally, the language's semantics can be obscure and difficult to understand, even for experienced programmers [Dharanikota et al. 2021].

Contract modeling is a recurring concern in the field of contract development. Researchers have highlighted the importance of using software engineering techniques to minimize human errors in contract creation [Velasco and Carvalho 2022, Ait Hsain et al. 2021, Chirtoaca et al. 2020, Hamdaqa et al. 2022, Santiago et al. 2021]. Among the software engineering techniques, Model-Driven Architecture (MDA) and Model-Driven Engineering (MDE) stand out, as they allow high-level modeling, enabling the creation of intelligent, secure, and reliable contracts. MDA is an architectural approach, while MDE is a more comprehensive methodology that encompasses the entire software life cycle, from conception to code generation.

This paper presents a systematic literature review (SLR) that aims to understand the limitations and challenges faced by smart contract developers on the EVM. The SRL aims to identify, based on software engineering techniques, proposals for facilitating and innovating the development of these smart contracts, as well as validation and evaluation strategies for opportunities of improvement and innovation in the field. The focus of the SLR is to identify barriers and obstacles to smart contract development, gather information on existing proposal validation and evaluation methods in the literature, and identify promising research horizons for making the smart contract development process more efficient and secure. The aim is to contribute to the advancement of smart contract development, enabling the creation of new decentralized applications that explore the properties of the blockchain in a safe and efficient manner.

The following sections provide a detailed overview of the research conducted. Section 2 presents the concepts that guide this SRL. Section 3 presents the research methodology adopted for the SLR. Section 4 discusses the results obtained for each research question. Finally, in section 5, the main findings of the SLR are discussed, highlighting their implications for smart contract development on the EVM.

2. Background

Blockchain technology enables decentralized and immutable storage of information [Angelis and Ribeiro da Silva 2019, Khan et al. 2019]. Additionally, this technology enables the creation of smart contracts, which are digital contracts capable of automatically executing agreed-upon clauses without the need for intermediaries. These contracts are recorded on a blockchain, allowing all involved parties to verify the authenticity and integrity of the information. Smart contracts have facilitated the development of decentralized applications (DApps) that operate autonomously on a peer-to-peer (P2P) network, enabling transactions to be automatically executed based on predetermined conditions in the smart contracts.

The Ethereum Virtual Machine (EVM) is an essential platform for the creation and execution of smart contracts and DApps. Contracts are executed as autonomous and independent objects within the EVM, but with limited resources and restricted access to external file systems and information [Antonopoulos and Wood 2018, Buterin et al. 2014]. Each operation performed on the EVM incurs a cost measured in gas, and to avoid the waste of resources in malicious or repetitive transactions, need to set a limit. The creation of contracts on the EVM is carried out through programming languages such as Solidity or Vyper. Furthermore, other blockchain platforms, such as Hyperledger Besu, EOS, Tron, Cardano, and Polygon, are EVM compatible, allowing contracts to be written on one platform and reused on others, which facilitates DApp development.

3. Research Methodology

The methodology used in this research for the systematic literature review followed the approach described by [Nakagawa et al. 2017], which is composed of three phases:

1. Planning: In this phase, the research objective is established, and an evaluation protocol is developed. This includes the definition of research questions, search strategy, research sources, keywords, and synonyms, selection criteria (inclusion and exclusion), and data extraction form.
2. Conduction: In this phase, the protocol defined in the previous phase is executed. This involves carrying out a broad search in the chosen databases, applying the inclusion and exclusion criteria, thoroughly reading the identified articles, and extracting the data.
3. Conclusion: This phase refers to the dissemination of the results and discussions obtained in the previous phases. It involves summarizing the findings, analyzing and interpreting the data, and drawing conclusions that address the research questions. These conclusions are then disseminated through publications, presentations, or other means of communication.

3.1. Research Questions

To meet the objective of this SLR, four research questions were developed and are presented below:

RQ1: "What are the common limitations and challenges faced by smart contract developers on the EVM?" Identifying the barriers and obstacles that hinder smart contract development allows for the creation of more efficient and secure solutions, as well as contributing to the evolution and growth of the technology.

RQ2: "How are software engineering techniques being used to facilitate or innovate the development of smart contracts for the EVM?" Identifying the approaches used in smart contract development helps to detect methods, trends, and, most importantly, research opportunities.

RQ3: "What are the methods used by authors to validate or evaluate proposals in the smart contract development field for the EVM?" There are several approaches in the smart contract development field that aim to make the process more efficient. These approaches employ the use of methods for validation or metrics for evaluation. In this sense, gathering and grouping the methods that authors have used aims to assist new research in this process.

RQ4: "What are the opportunities for improvement and innovation in the smart contract development field for the EVM?" With the constant evolution of blockchain technology and the growing use of smart contracts, there are several opportunities for innovation and research in this field. Thus, it is important to identify possible research horizons to make smart contracts more efficient and secure.

3.2. Keywords and Search String

The search string for this SLR was generated from a pilot search whose primary objective is to assist in identifying the most suitable search string for the SLR goal [Brereton et al. 2007]. The pilot search can be conducted based on the main terms that guide the research question and/or with the input of experts [Keele et al. 2007]. As results are obtained in each search database and the refinement process takes place, a main string is defined.

During the pilot phase of the research, appropriate keywords were selected to align with the research objectives. Since the focus of this investigation is on smart contract development, the pilot search was scoped to encompass keywords related to software engineering techniques, with a specific emphasis on Model-Driven Engineering and Model-Driven Architecture. As noted by [Ait Hsain et al. 2021], the Model-Driven techniques are relevant for constructing solutions in smart contracts, as they improve the efficiency and productivity of the development process. Furthermore, the search included the terms *Framework* and *Code Generation* to reflect recent developments that have presented tools to assist in the development process of smart contracts. The search string also included the term *Smart Contracts* and keywords related to EVM. To refine the search and narrow it down to the most pertinent works, the decision was made to limit the search string to the presence of these terms in the final search. This approach aimed to ensure that the results obtained in this SLR are of high quality and effectiveness.

The main search string was composed of the following keywords and their synonyms, which were divided into three groups: (*i*) *Model-Driven Engineering*, MDE, *Model-Driven Architecture*, MDA, *Model-driven*, *Code Generation*, *Framework*; (*ii*) *Smart Contracts*, *Smart Contract*; and (*iii*) *Ethereum*, *Ethereum Virtual Machine*, EVM. The search string was developed from these groups, grouped with the connector "AND," and each word in the group with the connector "OR." The generic search string is presented below:

("Model-Driven Engineering" OR "MDE" OR "Model-Driven Architecture" OR "MDA" OR "Model-driven" OR "Code Generation" OR "Framework") AND ("Smart Contracts" OR "Smart Contract") AND ("Ethereum" OR "Ethereum Virtual Machine" OR "EVM")

However, due to the number of results in the selected databases, search strings were produced according to the search engine of each platform. The first and second groups should have the keywords appearing in the title. Finally, the third group should appear throughout the document (title, metadata, and text). Table 1 presents the search strings according to the databases.

3.3. Inclusion and Exclusion Criteria

Inclusion and exclusion criteria are used to select relevant articles that meet the objectives and research questions of the SLR. These criteria are defined in advance and help to ensure that the articles included in the review are of high quality and relevance.

Database	Search string
ACM	(Title: “smart contracts” OR Title: “smart contract”) AND ()Title: “model-driven engineering” OR Title: “mde” OR Title: “model-driven architecture” OR Title: “mda” OR Title: “model-driven” OR Title: “code generation” OR Title: “framework”) AND (“ethereum” OR “ethereum virtual machine” OR “evm”)
IEEE	((“Document Title”:“Smart Contracts” OR “Document Title”: “Smart Contract”) AND (“Document Title”: “Model-Driven Engineering” OR “Document Title”:“MDE” OR “Document Title”: “Model-Driven Architecture” OR “Document Title”:“MDA” OR “Document Title”:“Model-driven” OR “Document Title”:“Code Generation” OR “Document Title”:“Framework”) AND (“Full Text & Metadata”:“Ethereum” OR “Full Text & Metadata”: “Ethereum Virtual Machine” OR “Full Text & Metadata”:“EVM”))
Scopus	(TITLE ((“Model-Driven Engineering” OR “MDE” OR “Model-Driven Architecture” OR “MDA” OR “Model-driven” OR “Code Generation”)) AND TITLE ((“Smart Contracts” OR “Smart Contract”)) AND ALL ((“Ethereum” OR “Ethereum Virtual Machine” OR ”EVM“)))

Table 1. Search string according to the database

Inclusion criteria:

- the article proposes an improvement or innovation in the field of smart contract development;
- the article describes an approach to smart contract development using EVM;
- the article discusses the limitations and challenges faced by smart contract developers; and
- the work presents results from experiments or empirical evaluations of the effectiveness of the proposed approach.

Exclusion criteria:

- the article does not cover smart contract development for EVM;
- the article does not propose an improvement or innovation in the field of smart contract development;
- the full document is not available for reading and;
- the work does not address or have applicability for the EVM platform.

3.4. Conduction

After the planning phase, the execution stage was initiated by submitting the search string to the selected platforms. The Parsif.al[1] tool was used as it assists in all phases of SLR. This process was carried out in early February 2023.

The articles were extracted from the selected databases and submitted to Parsif.al. Duplicated articles were automatically excluded by the tool. Following this, inclusion and exclusion criteria were applied to the remaining articles, such as reading the title and metadata (abstracts and keywords). Table 2 presents the list of articles found and selected for full-text reading.

To facilitate the full-text reading of the articles, a data extraction form was created to assist in answering the research questions. The topics of the form are presented below:

- What are the ways in which the authors facilitate or innovate in the area of smart contract development?
- What are the limitations or challenges faced by smart contract developers according to the authors?
- What are the research or innovation opportunities in the field according to the authors?
- What are the validation or evaluation methods used by the authors?

[1]Parsif.al is a web-based tool that assists in the planning and conducting of systematic literature reviews and mappings. The tool can be accessed through the website `https://parsif.al`.

Databases	F	A	R	D	References of the accepted articles
ACM	28	10	14	4	[Santiago et al. 2021], [Jiao et al. 2020a], [Vandenbogaerde 2019], [Jiao et al. 2020b], [Liu 2023], [Kaleem et al. 2021], [Annenkov et al. 2020], [Wang et al. 2020], [Li et al. 2021], and [Medeiros et al. 2019].
IEEE	50	20	29	1	[Dharanikota et al. 2021], [Dingman et al. 2019], [Zeng et al. 2022], [Tsiounis and Kontogiannis 2022], [Qasse et al. 2021], [Barišić et al. 2021], [Feist et al. 2019], [Ferreira et al. 2020], [Dong et al. 2022b], [Jurgelaitis et al. 2022], [Garamvölgyi et al. 2018], [Chirtoaca et al. 2020], [Dong et al. 2022a], [Ren et al. 2021], [Yu et al. 2021], [Akca et al. 2019], [Tsai et al. 2019], [Bao and Liu 2021], [Jin et al. 2022], and [Keilty et al. 2022]
Science Direct	5	5	0	0	[Ait Hsain et al. 2021], [Hamdaqa et al. 2022], [Boubeta-Puig et al. 2021], [Köpke et al. 2023], and [Corradini et al. 2021].
Scopus	10	1	3	6	[Jurgelaitis et al. 2023].
Total	**93**	**36**	**46**	**11**	

Table 2. (F)ound, (A)ccepted, (R)ejected, and (D)uplicated articles per database.

4. Results and Discussion

The following research questions were answered based on the full-text reading of 36 articles and with the support of the authors experience in this study area. Each of the research questions is addressed below.

4.1. (*RQ1*) What are the common limitations and challenges faced by smart contract developers on the EVM?

One of the most significant challenges in smart contract development is the immutability of the blockchain, as highlighted by several authors [Zeng et al. 2022, Feist et al. 2019, Dharanikota et al. 2021, Jiao et al. 2020a, Li et al. 2021, Jurgelaitis et al. 2023]. This feature makes the process even more complex than traditional software development, as the smart contract cannot be altered after implementation.

Despite iterative software development allowing for iterations throughout the phases of requirements specification, design, implementation, testing, and deployment, smart contract development can only support iterations before the implementation phase due to the immutability of the smart contract [Jurgelaitis et al. 2023]. This necessitates extra caution throughout the coding process, which includes rigorous testing such as unit and integration testing.

Developing decentralized solutions is a challenging, vague, and error-prone process [Jurgelaitis et al. 2022]. In particular, non-technical users face significant difficulties in this process due to factors such as language knowledge, infrastructure restrictions and limitations, and the relationships between deployed artifacts and resources [Qasse et al. 2021]. Furthermore, [Garamvölgyi et al. 2018] point out that many smart contract developers are new to the field, which can result in modeling errors related to these factors.

Smart contract programming imposes size restrictions that differentiate it from traditional software development architecture, making the process more complex. In addition, the interaction between different smart contracts can be complicated, and there are no guarantees that they meet the requirements and policies required by stakeholders [Tsiounis and Kontogiannis 2022]. It is important to note that smart contract artifacts, such as bytecode, and smart contract information, such as the state of public variables or transaction history, are publicly available, which increases the attention of malicious actors to exploit security vulnerabilities [Dong et al. 2022b].

Smart contract programmers face challenges due to the characteristics of the Solidity programming language. According to [Dharanikota et al. 2021], the language semantics are obscure and only partially understood by experienced programmers. Additionally, the authors of the [Garamvölgyi et al. 2018] have pointed out deficiencies in the language that can contribute to attacks, such as the DAO attack [Dhillon et al. 2017]. These deficiencies, coupled with coding errors, can result in the deployment of vulnerable smart contracts [Ferreira et al. 2020].

The lack of sufficient information in interface standards for smart contracts also hinders the development process [Chirtoaca et al. 2020]. Implementation specifications are often left to the discretion of the programmer, which can negatively impact the quality of the smart contract. Furthermore, even smart contracts that conform to a standard may not meet the required specifications, leaving interactions between them vulnerable [Dingman et al. 2019]. In the EVM model, detecting errors can be difficult since incorrect or nonexistent function calls trigger the fallback function, which requires verifying the supporting interface of the external smart contract [Jiao et al. 2020b].

Developers encounter several difficulties when using vulnerability analysis tools. These challenges include tool installation, limited availability, the need to contact the authors for use [Ferreira et al. 2020]. Comparing these tools is also challenging for developers and researchers due to a lack of information, such as databases, which makes reproducing previous research difficult [Ferreira et al. 2020]. Furthermore, many tools are available to detect vulnerabilities, but the learning curve for each is relatively high [Jiao et al. 2020b]. The work [Dong et al. 2022a] has reported that some of these tools have a low degree of automation and, in some cases, require expert evaluation, making them challenging to use for novice programmers [Jiao et al. 2020b, Yu et al. 2021]. Moreover, as pointed out by [Keilty et al. 2022], in some cases, developers lack knowledge about pre-existing vulnerabilities that could have been prevented, as well as the existence of relevant tools.

Due to the immutable nature of smart contracts, developers seek solutions to update them. One such solution is the use of the Proxy pattern, in which there is a data smart contract and another one with the implementation. However, as [Dong et al. 2022b] points out, this approach presents problems regarding the internal state of the smart contract, particularly in relation to native currency values. In some cases, these values become locked and cannot be transferred to another implementation. Additionally, implementing this solution is not easy for all smart contract developers, as it requires knowledge beyond programming, such as understanding the EVM. In some cases, this renders the deployment smart contract update process unfeasible [Dharanikota et al. 2021].

4.2. (*RQ2*) How are software engineering techniques being used to facilitate or innovate the development of smart contracts for the EVM?

In light of the challenges faced by smart contract developers, code reuse is essential. Initiatives, such as Ethereum Improvement Proposals (EIPs), help standardize solution development [Jurgelaitis et al. 2022]. These EIPs are crucial to ensuring standardized coding within the application context. Additionally, there are community projects, such as OpenZeppelin, that offer code snippets or building blocks to be used in smart contracts. These codes are widely discussed and tested by the community, which contributes to the security of the process. Thus, it is already possible to find smart contracts implementing standards such as ERC-20, ERC-721, among others.

Writing secure smart contracts is a challenge for developers [Ferreira et al. 2020]. To address this issue, several initiatives aim to identify vulnerabilities in smart contracts from their conception to execution. The scientific community has been working to provide tools that assist developers in writing more secure smart contracts, either through static or dynamic analysis, or to fix already available smart contracts with vulnerabilities. Some research focuses on detecting vulnerabilities [Zeng et al. 2022, Dong et al. 2022b, Dong et al. 2022a, Feist et al. 2019, Wang et al. 2020, Yu et al. 2021, Akca et al. 2019, Keilty et al. 2022], while others aim to gather available tools [Ferreira et al. 2020, Jiao et al. 2020b].

Tools such as the proposal by [Dingman et al. 2019] facilitate the process of testing smart contracts by allowing them to be tested according to software testing practices, which speeds up the runtime of tests. Additionally, there are other tools that allow developers to write smart contracts in Solidity and perform unit and integration tests in the same codebase, such as Hardhat[2] and Truffle[3], using in-memory EVM instances.

Privacy is one of the concerns of some authors [Ren et al. 2021, Bao and Liu 2021]. In [Bao and Liu 2021], authors use model checking and formal verification to protect privacy data in contracts. The proposed tool allows developers to graphically design privacy requirements. Meanwhile, in [Ren et al. 2021], a Multi-party Transaction (MPT) application is used in which developers specify privacy invariants and then generate code to be deployed on the blockchain.

The MDE and MDA approaches have been employed in several works [Qasse et al. 2021, Barišić et al. 2021, Tsai et al. 2019, Jurgelaitis et al. 2022]. In [Qasse et al. 2021], a chatbot that uses natural language to model smart contracts through the IContractML metamodel [Hamdaqa et al. 2022] was proposed, allowing for the creation of smart contracts. In [Barišić et al. 2021], MDE was used to propose a multi-chain contract structure that addresses reliability issues through models. Furthermore, in [Tsai et al. 2019], a framework based on MDE was presented for generating legal contract templates. In [Jurgelaitis et al. 2022], the MDA approach was employed to create a specific model for the Ethereum platform, demonstrating the process of transforming the model into Solidity code. Finally, in [Garamvölgyi et al. 2018], a solution using Model-Driven Development (MDD) was presented for generating code from UML diagrams.

Other tools have been proposed to facilitate or innovate in the area of smart contracts, such as in [Santiago et al. 2021], where a tool was proposed that allows for the writing of contracts at runtime using models in JSON and Typescript. On the other hand, in [Jin et al. 2022], a framework is presented that uses a patching mechanism to repair already deployed smart contracts. The framework works by intercepting calls to the original contract and blocking malicious transactions through another contract.

Additionally, other approaches include the use of UML state diagrams in smart contract generation, as seen in [Garamvölgyi et al. 2018] and [Jurgelaitis et al. 2022]. There are also tools that allow for the writing of formally verified smart contracts,

[2]https://hardhat.org/
[3]https://trufflesuite.com/

as presented by [Dharanikota et al. 2021, Annenkov et al. 2020, Kaleem et al. 2021, Keilty et al. 2022]. In [Chirtoaca et al. 2020], the authors propose a specification language for writing ERC-721 smart contracts, while [Vandenbogaerde 2019] constructs a graph-based framework that allows for representing, querying, and analyzing smart contracts. The work [Tsiounis and Kontogiannis 2022] explores the use of models that allow the use of extended metamodels to represent the necessary actions, tasks, and policies. Finally, some authors explore the automation of business processes using BPMN and blockchain in [Köpke et al. 2023] and [Corradini et al. 2021].

The authors of the work [Kaleem et al. 2021] present a solution for notifying smart contracts about external events and their relationship with the real world. However, this is a challenging task due to the lack of integration with the external world in the EVM blockchain ecosystem. Currently, this proposal is still in the conceptual stage.

In addition to the works previously identified, the review presented by [Ait Hsain et al. 2021] also identified other works that make use of MDE techniques for the development of smart contracts. In the review, the authors identified that the main methods employed in 10 selected works were BPMN, Finite State Machine, and UML, demonstrating the diversity of approaches used in the field. However, in contrast to the review, this SRL identified works that use metamodels, MDA, and MDD, highlighting the variety of techniques and tools available for the development of smart contracts through software engineering techniques.

4.3. (*RQ3*) What are the methods used by authors to validate or evaluate proposals in the smart contract development field for the EVM?

The development of smart contracts is an area of research in constant evolution. The validation or evaluation of these works is crucial to ensure the security and efficiency of the proposals in this area. Depending on the stage of the proposal and the area of application, the forms of validation or evaluation may vary.

Some of the most common ways to validate or evaluate smart contract proposals include using use cases in smart contract implementation, extracting open or vulnerable smart contracts, conducting comparative experiments with other tools, evaluating with quality metrics, and performance evaluation. Use cases allow for checking the applicability of the proposal in real situations, while the extraction of vulnerable smart contracts helps to identify areas for improvement. Comparative experiments enable the evaluation of the proposal's efficiency and effectiveness in comparison with other existing solutions, while evaluations based on quality metrics provide a more objective view of the proposal. Finally, performance evaluation allows for assessing the scalability and ability of the proposal to handle large amounts of data.

In Table 3, the forms used by the authors to validate or evaluate their works are presented.

4.4. (*RQ4*) What are the opportunities for improvement and innovation in the smart contract development field for the EVM?

Smart contract correctness is an important area that needs improvement [Dong et al. 2022b]. Some attempts have been made to solve this problem, such as the proposal presented by [Vandenbogaerde 2019], which proposes a mechanism that

Forms	Works
Implementation of use cases in smart contracts.	[Garamvölgyi et al. 2018], [Qasse et al. 2021], [Jurgelaitis et al. 2022], [Tsiounis and Kontogiannis 2022], [Ferreira et al. 2020], [Dharanikota et al. 2021], [Annenkov et al. 2020], [Kaleem et al. 2021], [Li et al. 2021], [Wang et al. 2020], [Medeiros et al. 2019], [Jurgelaitis et al. 2023], [Hamdaqa et al. 2022], [Boubeta-Puig et al. 2021], [Köpke et al. 2023], [Ren et al. 2021], [Tsai et al. 2019], [Bao and Liu 2021], [Keilty et al. 2022].
Extraction of open or vulnerable smart contracts.	[Zeng et al. 2022], [Dong et al. 2022b], [Liu 2023], [Vandenbogaerde 2019], [Yu et al. 2021], [Akca et al. 2019], [Jin et al. 2022].
Comparative experiments with other tools.	[Dong et al. 2022b], [Feist et al. 2019], [Akca et al. 2019], [Jin et al. 2022], [Keilty et al. 2022].
Evaluation with quality metrics.	[Chirtoaca et al. 2020] and [Jiao et al. 2020a].
Performance evaluation	[Dong et al. 2022a], [Li et al. 2021], [Medeiros et al. 2019], [Ren et al. 2021], [Yu et al. 2021], [Bao and Liu 2021].

Table 3. The forms of validation and evaluation used.

allows for destroying a vulnerable smart contract by voting, transferring the assets to another ownership, and re-implementing a corrected version. This mechanism differs from the Proxy solution, where in the case of update failure, the assets remain trapped in the smart contract.

The security of smart contracts is a crucial subject for the consolidation of blockchain technology. Although initiatives have been taken to find and mitigate vulnerabilities, there is still plenty of room for improvements and innovations in the field. One way to enhance security is through the provision of more accessible tools to find and fix existing and new vulnerabilities [Ferreira et al. 2020]. Additionally, the programming language itself could incorporate elements that make smart contracts less vulnerable, making the formation of new developers easier and enhancing the overall security of smart contracts [Garamvölgyi et al. 2018].

Complexity in the area of smart contract development represents a barrier to entry for new programmers, and lack of understanding by non-technical people is also a challenge [Qasse et al. 2021]. To address these issues, it is important to incorporate software engineering concepts into smart contract development solutions [Hamdaqa et al. 2022]. This would help to minimize errors and make the field more accessible and understandable for all involved. Additionally, education and awareness about best smart contract security practices should be expanded to encourage and empower more people to get involved with smart contract development.

The scalability of blockchain is a critical issue that needs to be addressed to ensure the long-term success of smart contract and DApp development. The increasing demand for new application domains and the constant use of smart contracts makes smart contract size a scalability obstacle. Additionally, interactions between smart contracts and DApps are also difficult, limiting network functionality and efficiency [Kaleem et al. 2021]. To overcome these obstacles, innovation in the area of scalability is needed, seeking solutions that allow for efficient scalability of smart contracts and interactions between DApps.

Interoperability between blockchains is a critical issue in the development of decentralized applications. Currently, each blockchain has its own rules, standards, and protocols, making it difficult for smart contracts and DApps to interact with each other. The solution to this problem would be the creation of software engineering solutions that allow for the development of blockchain-independent smart contracts, enabling easier interoperability and support for decentralized applications across multiple blockchains [Chirtoaca et al. 2020]. This would increase the scalability and flexibility of the technology, as well as broaden the user base and make the technology more accessible for the industry and end-users.

5. Conclusion

Developers of smart contracts face various challenges and limitations in the process of developing decentralized solutions, such as immutability, size limitations, language constraints, infrastructure, relationships between artifacts, and potential vulnerabilities. Code reuse is an important solution to ensure standardization and security of smart contracts. Writing secure smart contracts is a constant concern, with various initiatives being undertaken to identify vulnerabilities and provide assistance tools to developers. Approaches other than MDE and MDA include the use of UML state diagrams, BPMN, formal verification tools, specification languages, and graph-based structures. The process of testing smart contracts is also facilitated by tools, including unit and integration testing.

However, there is still a lack of solutions that allow for the training of new smart contract developers capable of creating, implementing, and ensuring the security of these smart contracts. Existing proposals are often difficult to understand, involving complex formal verifications or requiring participants to have advanced knowledge in modeling. The use of graphical tools makes it more accessible for new programmers to enter the field, helping with the modeling of smart contracts and models in a more intuitive way. Therefore, it is essential to develop new methods for creating smart contracts that are more secure and less susceptible to human error. This article highlights the importance of software techniques to ensure the security of smart contracts, playing a crucial role in this aspect.

According to [Dingman et al. 2019], the lack of documentation and classification is the primary cause of the proliferation of vulnerabilities in blockchain-based smart contracts. Although there are numerous efforts in academic literature to address these issues, many smart contract developers in the industry do not have access to this information due to a lack of freely available resources. The availability of easily accessible and understandable materials on secure smart contract development, including vulnerabilities and best practices, would be extremely useful for the training of new smart contract developers.

References

Ait Hsain, Y., Laaz, N., and Mbarki, S. (2021). Ethereum's smart contracts construction and development using model driven engineering technologies: a review. *Procedia Computer Science*, 184:785–790. The 12th ANT / The 4th EDI40 / Affiliated Workshops.

Akca, S., Rajan, A., and Peng, C. (2019). Solanalyser: A framework for analysing and testing smart contracts. In *2019 26th APSEC*, pages 482–489.

Angelis, J. and Ribeiro da Silva, E. (2019). Blockchain adoption: A value driver perspective. *Business Horizons*, 62(3):307–314.

Annenkov, D., Nielsen, J. B., and Spitters, B. (2020). Concert: A smart contract certification framework in coq. In *Proceedings of the 9th ACM SIGPLAN*, page 215–228, New York, NY, USA. CPP 2020.

Antonopoulos, A. M. and Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps*. O'reilly Media.

Bao, T. and Liu, Y. (2021). A privacy-preserving framework for smart contracts based on stochastic model checking. In *2021 IEEE 20th TrustCom*, pages 460–467.

Barišić, A., Zhu, E., and Mallet, F. (2021). Model-driven approach for the design of multi-chain smart contracts. In *2021 3rd BRAINS)*, pages 37–38.

Boubeta-Puig, J., Rosa-Bilbao, J., and Mendling, J. (2021). CEPchain: a graphical model-driven solution for integrating complex event processing and blockchain. *Expert Systems with Applications*, 184:115578.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, 80(4):571–583.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1.

Chirtoaca, D., Ellul, J., and Azzopardi, G. (2020). A framework for creating deployable smart contracts for non-fungible tokens on the ethereum blockchain. In *2020 IEEE DAPPS*, pages 100–105.

Corradini, F., Marcelletti, A., Morichetta, A., Polini, A., Re, B., Scala, E., and Tiezzi, F. (2021). Model-driven engineering for multi-party business processes on multiple blockchains. *Blockchain: Research and Applications*, 2(3):100018.

Dharanikota, S., Mukherjee, S., Bhardwaj, C., Rastogi, A., and Lal, A. (2021). Celestial: A smart contracts verification framework. In *2021 FMCAD*, pages 133–142.

Dhillon, V., Metcalf, D., Hooper, M., Dhillon, V., Metcalf, D., and Hooper, M. (2017). The dao hacked. *blockchain enabled applications: Understand the blockchain Ecosystem and How to Make it work for you*, pages 67–78.

Dingman, W., Cohen, A., Ferrara, N., Lynch, A., Jasinski, P., Black, P. E., and Deng, L. (2019). Classification of smart contract bugs using the nist bugs framework. In *2019 IEEE 17th SERA*, pages 116–123.

Dong, H., He, Y., Tao, H., and Duan, Q. (2022a). A framework for formal transformation and analysis of smart contract code. In *2022 9th DSA*, pages 993–994.

Dong, W., Zhou, T., and Yan, D. (2022b). Solchecker: A practical static analysis framework for ethereum smart contract. In *2022 CNCIT*, pages 179–186.

Feist, J., Grieco, G., and Groce, A. (2019). Slither: A static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd WETSEB*, pages 8–15.

Ferreira, J. F., Cruz, P., Durieux, T., and Abreu, R. (2020). Smartbugs: A framework to analyze solidity smart contracts. In *2020 35th IEEE/ACM ASE*, pages 1349–1352.

Garamvölgyi, P., Kocsis, I., Gehl, B., and Klenik, A. (2018). Towards model-driven engineering of smart contracts for cyber-physical systems. In *2018 48th Annual IEEE/IFIP DSN-W*, pages 134–139.

Hamdaqa, M., Met, L. A. P., and Qasse, I. (2022). iContractML 2.0: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms. *Information and Software Technology*, 144:106762.

Jiao, J., Lin, S.-W., and Sun, J. (2020a). A generalized formal semantic framework for smart contracts. In *23rd FASE 2020 and ETAPS 2020 Proceedings*, page 75–96, Berlin, Heidelberg. Springer-Verlag.

Jiao, J., Lin, S.-W., and Sun, J. (2020b). A generalized formal semantic framework for smart contracts. In *23rd FASE 2020 and ETAPS 2020*, page 75–96, Berlin, Heidelberg. Springer-Verlag.

Jin, H., Wang, Z., Wen, M., Dai, W., Zhu, Y., and Zou, D. (2022). Aroc: An automatic repair framework for on-chain smart contracts. *IEEE Transactions on Software Engineering*, 48(11):4611–4629.

Jurgelaitis, M., Čeponienė, L., Butkus, K., Butkienė, R., and Drungilas, V. (2023). Mda-based approach for blockchain smart contract development. *Applied Sciences (Switzerland)*, 13(1).

Jurgelaitis, M., čeponienė, L., and Butkienė, R. (2022). Solidity code generation from uml state machines in model-driven smart contract development. *IEEE Access*, 10:33465–33481.

Kaleem, M., Kasichainula, K., Karanjai, R., Xu, L., Gao, Z., Chen, L., and Shi, W. (2021). An event driven framework for smart contract execution. page 78–89, New York, NY, USA. DEBS '21.

Keele, S. et al. (2007). Guidelines for performing systematic literature reviews in software engineering.

Keilty, E., Nelaturu, K., Wu, B., and Veneris, A. (2022). A model-checking framework for the verification of move smart contracts. In *2022 IEEE 13th ICSESS*, pages 1–7.

Khan, A. G., Zahid, A. H., Hussain, M., Farooq, M., Riaz, U., and Alam, T. M. (2019). A journey of web and blockchain towards the industry 4.0: An overview. In *2019 International Conference on Innovative Computing (ICIC)*, pages 1–7.

Köpke, J., Meroni, G., and Salnitri, M. (2023). Designing secure business processes for blockchains with secbpmn2bc. *Future Generation Computer Systems*, 141:382–398.

Li, Z., Zhou, Y., Guo, S., and Xiao, B. (2021). Solsaviour: A defending framework for deployed defective smart contracts. In *ACSAC*, page 748–760, New York, NY, USA. ACSAC '21.

Liu, Y. (2023). A unified specification mining framework for smart contracts. New York, NY, USA. ASE '22.

Medeiros, H., Vilain, P., Mylopoulos, J., and Jacobsen, H.-A. (2019). Solunit: A framework for reducing execution time of smart contract unit tests. In *Proceedings of the 29th CASCON*, page 264–273, USA. IBM Corp.

Nakagawa, E. Y., Scannavino, K. R. F., Fabbri, S. C. P. F., and Ferrari, F. C. (2017). Revisão sistemática da literatura em engenharia de software: teoria e prática.

Qasse, I., Mishra, S., and Hamdaqa, M. (2021). icontractbot: A chatbot for smart contracts' specification and code generation. In *2021 IEEE/ACM 3rd BotSE*, pages 35–38.

Ren, Q., Liu, H., Li, Y., and Lei, H. (2021). Demo: Cloak: A framework for development of confidential blockchain smart contracts. In *2021 IEEE 41st ICDCS*, pages 1102–1105.

Santiago, L., Abijaude, J., and Greve, F. (2021). A framework to generate smart contracts on the fly. In *Proceedings of the XXXV SBES*, page 410–415, New York, NY, USA. Association for Computing Machinery.

Tsai, W.-T., Ge, N., Jiang, J., Feng, K., and He, J. (2019). Invited paper: Beagle: A new framework for smart contracts taking account of law. In *2019 IEEE SOSE*, pages 134–13411.

Tsiounis, K. and Kontogiannis, K. (2022). Goal and policy based code generation and deployment of smart contracts. In *2022 IEEE SANER*, pages 1227–1230.

Vandenbogaerde, B. (2019). A graph-based framework for analysing the design of smart contracts. page 1220–1222, New York, NY, USA. ESEC/FSE.

Velasco, G. and Carvalho, S. (2022). Uma abordagem dirigida por modelo para desenvolvimento de contratos inteligentes na ethereum virtual machine. In *Anais da X ERI-GO*, pages 106–117, Porto Alegre, RS, Brasil. SBC.

Wang, Z., Dai, W., Choo, K.-K. R., Jin, H., and Zou, D. (2020). Fsfc: An input filter-based secure framework for smart contract. *J. Netw. Comput. Appl.*, 154(C).

Yu, X., Zhao, H., Hou, B., Ying, Z., and Wu, B. (2021). Deescvhunter: A deep learning-based framework for smart contract vulnerability detection. In *2021 IJCNN*, pages 1–8.

Zeng, Q., He, J., Zhao, G., Li, S., Yang, J., Tang, H., and Luo, H. (2022). Ethergis: A vulnerability detection framework for ethereum smart contracts based on graph learning features. In *2022 IEEE 46th COMPSAC*, pages 1742–1749.