Smart contracts and issue reporting: a preliminary analysis

Ermanno Francesco Sannini[1,*], Andrea Di Sorbo[1] and Corrado Aaron Visaggio[2]

[1]University of Sannio, Benevento, Italy
[2]University of Foggia, Foggia, Italy

Abstract
The widespread adoption of blockchain and smart contracts (SCs) in contexts and tasks beyond currency management has introduced novel challenges in their development and maintenance. This paper investigates the characteristics of issue reports and their handling dynamics in SC-related projects, comparing them against traditional software. To this end, we curated a dataset of Ethereum-based smart contracts, selecting the top 2,000 active and used ones. Out of them, 42 projects with publicly maintained source code on GitHub were discovered. Using data extracted from the issue trackers of these projects, we conducted a statistical comparison with known patterns in traditional software projects. The results reveal that SC developers and users actively engage with issue-tracking systems, though most SC projects receive fewer than 50 issue reports. Besides, a significant portion of issue reports in SC-related projects pertain to feature enhancements, and resolution tends to be faster, often involving extensive use of comments. These trends reflect the community's emphasis on modularity, upgradeability, and collaborative development practices, suggesting that, in SC-related projects, maintenance tools and platforms should prioritize support for enhancement-oriented workflows and collaborative communication. Additionally, the heavy use of commentary features underscores the importance of integrated communication tools and traceability features to support transparent and audit-friendly development processes.

Keywords
Solidity, Ethereum, Issue Report, Github

1. Introduction

Smart Contracts (SCs) have been established as one of the most innovative and promising tools, mainly valued for their ability to automate transactions and processes in a secure and decentralized way. Initially tied to the transfer of digital assets, their use has rapidly expanded to areas such as decentralized finance (DeFi), digital identity, supply chain tracking, gaming, and virtual property management [1]. A chief factor that has contributed significantly to the massive use of SCs was the completion and capability of the Ethereum blockchain, which led the blockchain as a programmable platform [2]. While early blockchain systems focused only on digital asset transfer, Ethereum introduced an execution environment through a distinctive Virtual Machine (EVM), which makes it possible to build and deploy decentralized applications (dApps) [3]. They are usually developed in a high-level language, such as Solidity, Vyper, or other languages. *Solidity* is the programming language's most popular way to develop SCs on Ethereum. Based on the concept of *immutability*, SCs cannot be modified once added to the blockchain. Once started, all contract execution is based on its code. No one can change it, even the creator [4]. Another distinctive facet of Ethereum is the introduction of the *gas* mechanism, a measure of the computational work required to perform operations on the platform, i.e. the cost of operations performed on the blockchain [5]. Every transaction, from a simple value transfer to a complex function call, incurs a gas cost that must be paid in Ether. Efficient gas usage is therefore a key design concern in SCs development [4].

The inherent characteristics of SCs lead to unusual challenges influencing all phases of the software life cycle [6]. For instance, due to the characteristics of immutability and transparency of SCs, in the design steps, it is necessary to make systems modular and upgradable and avoid storing sensitive

DLT2025: 7th Distributed Ledger Technology Workshop, June, 12-14 2025
*Corresponding author.
e.sannini@studenti.unisannio.it (E. F. Sannini); disorbo@unisannio.it (A. D. Sorbo); corrado.visaggio@unifg.it (C. A. Visaggio)
0000-0002-3192-739X (A. D. Sorbo); 0000-0002-0558-4450 (C. A. Visaggio)

information, as, once published, it will be accessible to everyone. During the development phase, instead, care must be taken in implementing the smart contract by optimizing both the computational cost of gas that would be required for its execution and security because smart contracts often control and manage sensitive digital assets and, after the deployment, an attacker could identify any flaws and exploit them. Due to immutability, it will not be possible to accomplish updates or corrective patches [7]. Furthermore, the lack of consolidated best practices, universal standards in terms of security and efficiency, and adequate tools to verify the correctness of the code complicates these already tangled steps. Finally, concerning the maintenance phase, challenges inevitably arise in implementing corrective and adaptive actions when a problem is discovered or a change disrupts the SC operation. Once deployed, modifications to the specific smart contract to restore proper functionality are no longer possible.

However, little is known about the issues users and developers of active SCs typically encounter after SC deployment and how these issues are specifically managed and resolved. To start filling this gap, this work explores issues reported in SC-related projects on collaborative platforms, investigating how they are communicated and managed. By comparing these practices with those of more conventional software repositories, we aim to identify patterns occurring in similar underlying reports and assess the effectiveness of the current reporting and management dynamics, with the long-term goal of improving the accuracy and timeliness of issue management practices in SC-related projects. The analysis specifically focuses on (i) identifying active smart contract repositories with issue reporting enabled, (ii) analyzing the frequency, nature, and classification of reported issues, and (iii) drawing comparisons with traditional software projects in terms of issue types and their lifecycle. In particular, understanding how often issues arise and what types are most common can help teams prioritize recurring pain points and allocate resources more effectively. In addition, parallels with conventional software allow for identifying gaps or best practices, leading to improved maturity in issue management.

Paper structure. The paper proceeds as follows. Section 2 discusses the related literature, highlighting the novelty of our findings. Section 3 presents the research questions and the methodology to answer them. Section 4 reports and discusses the preliminary results. Finally, Section 5 concludes the paper outlining future research directions.

2. Related work

The scientific community is currently paying particular attention to the critical phases of the smart contract life cycle, focusing on development and maintenance. To date, these phases represent the ones that most differ from traditional systems, giving rise to innovative challenges to improve security, reliability, and efficiency.

Regarding the development phase, the study conducted by the Karlsruhe Institute of Technology (KIT) [8] pinpointed three main categories of challenges: (i) the *platform* makes it difficult to trace the user's identity through the address used, and the updateability of the code; (ii) the *programming language and execution environment* are lean to computational inefficiency and inadequate memory management, leading to performance drops and gas leaks; (iii) the adoption of complex *code practices* concerning transactions' security and the currency management that can disrupt code readability and understandability.

Regarding the maintenance phase, instead, Chen et al. [4] focused on the challenges affecting three main tasks: (i) the *corrective maintenance* is hampered by the lack of refined tools for bugs and error detections, alongside skimpy community support in reporting due to the lack of developers and researchers with expertise in the field; (ii) the *adaptive maintenance* is made tricky by the ongoing evolution of the Ethereum network, that can make any SC obsolete or even no longer operational, heavily affecting its dependability; (iii) the *perfective maintenance* is deeply restricted by the plainness of Solidity grammar, which limits projects' scalability, and imposes trade-offs between gas cost optimization and code readability. Besides, *preventive maintenance* is fairly absent due to the constant SCs' ecosystem growth and the lack of consolidated standards and experts in the field. Finally, the study emphasizes the shortage of fruitful tools and methodologies to support the maintenance phase of smart contracts.

A further obstacle impacting the development and maintenance of smart contracts is represented by *code duplication*, a widespread practice, especially among less skilled developers. For example, Pierro and Tonelli [9] discovered that on 7,500 smart contracts carried from the Ethereum platform, over 80% of them include duplicate portions of code. This behavior makes the implementation phase easier but raises severe issues. Without appropriate verification and customization, code clones improve complexity. This often translates into greater gas consumption, increasing the maintenance efforts required for optimization. Moreover, code clones often contain outdated functions or libraries or functions developed for other purposes, introducing errors or vulnerabilities. Indeed, He et al. [10] observed that the high presence of SCs containing duplicate code fragments has led to a proliferation of contracts that inherit vulnerabilities or design errors in the Ethereum ecosystem.

The high propagation of vulnerabilities represents an effective criticality for SC security and reliability. In 2022, observing 30 SC-related security flaws, the estimated financial losses amounted to 1.2 billion, with an increase of 837% compared to the previous year, indicative that many developers had not updated the code of their SCs to fix known vulnerabilities [11]. In light of these findings, the scientific community is reserving greater awareness of vulnerability analysis and developing increasingly refined tools for the static and dynamic analysis of SCs.

In contrast to prior studies focusing on technical and structural challenges, the work by Vaccargiu et al. [12] investigates the socio-technical factors of issue management within blockchain-based projects. Analyzing $15,954$ Go-Ethereum issues and $50,023$ comments from GitHub, the authors apply topic modeling to reveal that sustainability emerges as a prominent theme, accounting for about 38% of the identified topics. Additionally, network analysis highlights gas prices and cost efficiency as main topics in sustainability communities.

Unlike the studies above, this work investigates issue reporting and management in SC-related projects. Bug reports or requests for new features are essential to software maintenance and evolution tasks. In traditional software systems, issue management happens through consolidated tools, such as issue tracking systems, which allow recording, categorizing, and managing reports in a structured way. As regards smart contracts, however, it is unclear how and what types of issues are reported and effectively addressed. Therefore, this study evaluates how much and in what way consolidated software maintenance tools are adopted in the distinct context of smart contracts.

3. Objectives and Study Design

The *goal* of our study is to investigate the issue reporting and management practices in SC-related projects to spot operational peculiarities. Starting from open source projects (as a preliminary analysis, only projects publicly available on GitHub were considered), the goal is to understand the process of issue report discussion, management, and resolution in SC-related projects and compare them with those observed in traditional software systems. The findings provide useful knowledge to improve the tools and practices of problem management in blockchain-based projects. With these objectives, we pose the following research questions:

- ***RQ$_1$: Among the most active and used SCs on the Ethereum blockchain, how many have their source code publicly developed and maintained on the GitHub collaborative development platform?***
- ***RQ$_2$: What percentage of smart contract projects found on GitHub have received at least one issue report and how many issues are tracked?***
- ***RQ$_3$: What are the most recurring labels associated with issue reports in smart contract projects found on GitHub?***
- ***RQ$_4$: What are the characteristics of closed issue reports across smart contract projects hosted on GitHub?***

RQ$_1$ assesses the transparency and accessibility of the source code of the most used smart contracts on the Ethereum blockchain. With RQ$_2$, we pinpoint the adoption of issue trackers among SC developers. The frequency and distribution of issue reports are also evaluated to understand the extent and complexity of issue management in the identified smart contract-related projects. RQ_3 determines the recurrent cases and their categorization for gaining a better understanding of the most common criticalities. Finally, with RQ_4, we analyze the dynamics of closed issues reported on the different projects.

3.1. Experimental dataset

Starting from the public Ethereum dataset *bigquery-public-data.crypto_ethereum*[1], Google BigQuery was used to find all contract addresses, joined with the total number of transactions, performed since the contract deployment on the blockchain, and the date of the last recorded transaction. This stage has led to the identification of $8,922,530$ contract addresses. Since we were interested in mining active and used SCs, the relevance of the collected data was assured by removing addresses with less than 10 transactions associated or whose last transaction was earlier than 2024, reducing the addresses to $272,014$. Later, the contracts' source code was retrieved using *Etherscan*, a well-known platform to explore and examine the Ethereum blockchain [13], obtaining 203,033 files. All the contracts with a language different from Solidity and any duplicates were eventually filtered out. After this filtering step, the experimental dataset evolved to $82,337$ files. We then ordered the remaining files by the number of transactions received and selected the top $2,000$, ensuring extensive interaction and representativeness.

3.2. Analysis methodology

To answer RQ_1, a Python script was built to automate the invocation of the GitHub REST API. This service allows access to a wide range of GitHub features [14], i.e. *full_name, description, created_at, updated_at, stargazers_count, forks_count, open_issues, open_issues_count, pushed_at, license, default_branch, topics*. For each of the 2,000 smart contracts featured in the dataset detailed in Section 3.1, the script exploits the GitHub search feature by (i) building a query containing a noteworthy piece of the Solidity source code and (ii) analyzing all the retrieved repositories in terms of the number of Solidity files contained: if this number is greater than 100, the script recognizes the repository as a collection of SCs and, therefore, discard it; otherwise, the repository is identified as worth of further analysis. For the remaining repositories (i.e., the ones worthy of further analysis), a deeper comparison between the whole source code of the smart contract extracted from the initial dataset and the one retrieved from GitHub is performed. The main limitations encountered are: (i) the limited number of searches that can be executed via the GitHub API in a given time frame, causing long execution time; (ii) a high number of search results for each query due to smart contracts with duplicate code fragments; (iii) the GitHub API limit for query size to 256 characters, leading to high false positives in query results. The first limitation was mitigated through a sleep function. To address the others, we created queries reporting the first 200 characters of the source code contained in the initial dataset and filtering the results to the Solidity files only. Despite being simple, this method has proven valid as, usually, details at the beginning of the source code, such as specific metadata, introductory comments, the Solidity version, or any logos, allow for obtaining more pertaining search results. GitHub's search algorithm sorts results based on the position of the search phrase within the code; that is, the first outcomes produced match the search string in the initial lines, increasing the precision of the search. The second approach populates the query with the functions' signatures of the source code, obtained with regular expressions. The final results include all distinct projects successfully retrieved using these two approaches, without considering out-of-scope (e.g., audits, replication packages, API documentation, testing tools, etc.) and projects with less than five stargazers, to exclude toy projects or projects with too limited visibility.

The other RQs were answered using the *Perceval* tool [15]. Leveraging the GitHub REST API, Perceval allows users to pull out all issue reports' data from a repository and collect them in JSON format. A Python script was used to run Perceval on all repositories found in RQ_1, analyze the corresponding outputs, and compute the required statistics. The procedure first involves the extraction of issues only, as the GitHub API includes also pull requests in issue retrieval. For each issue retrieve the current

[1]https://cloud.google.com/blog/products/data-analytics/data-for-11-more-blockchains-in-bigquery-public-datasets

state, and, if it is *closed*, collect the comments' number and compute the Time to Resolution (TTR), by subtracting *created_at* from *closed_at* time. Eventually, gather the tags related to each issue.

4. Results

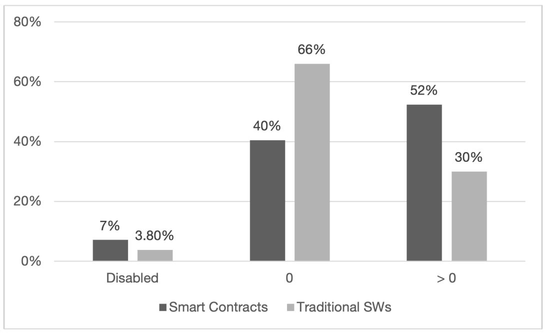
This section reports the obtained results and compares them with those concerning traditional software observed in previous work: for RQ_2 and RQ_3, we considered the study of Bissyandé et al. [16], who analyzed $100,000$ GitHub repositories, focusing on the adoption and impact of issue tracking systems in traditional open source projects on GitHub; for RQ_4, the comparison is made considering the work of Sülün et al. [17], who examined the adoption of issue templates across GitHub projects and the managing dynamics of 1.9 million issue reports with and without conformance to templates.

4.1. Smart contracts developed on GitHub and issue report adoption

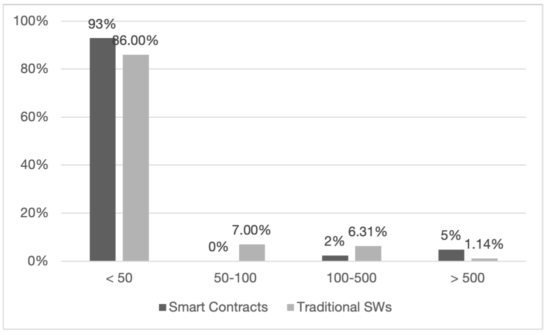
To answer RQ_1, the execution of the queries described in 3.2 led respectively to 339 files within 251 repositories and 217 files within 175 repositories, merged into 436 unique files within 316 distinct repositories. After filtering out low visibility and out-of-scope projects, the repositories' number significantly decreased to 42 items. Therefore, among the $2,000$ most active and used SCs on the Ethereum blockchain, only 2.1% of them are publicly developed and maintained on the GitHub collaborative development platform. The limited number of identified repositories highlights a key limitation of relying solely on open-source sources for smart contract investigation, suggesting that most active contracts are likely developed and maintained privately or via alternative platforms. To investigate this further, since in some platforms such as GitLab the global search is restricted to premium users, and manually querying every single project was prohibitively time-consuming, we performed a raw Google query search using the first 150 characters for each of the $2,000$ contracts and filtered the results matching the domains *https://gitlab.com*, *https://bitbucket.com*, and *https://openzeppelin.com*. However, this strategy yielded no successful matches. Therefore, we proceeded by collecting the top ten domains from Google's response to each of the queries above. As reported in Table 1, GitHub is the most frequently occurring domain, confirming its essential role for future analyses. Notably, for 223 repositories, no trace could be found on the public web, suggesting they may be private or hosted on non-indexed or restricted-access platforms. Among the domains identified, several are not hosting platforms but discussion forums (e.g., Ethereum StackExchange, OpenZeppelin Forum), documentation sites (e.g., docs.openzeppelin.com, docs.soliditylang.org), or educational resources (e.g., O'Reilly, GeeksforGeeks, Metaschool). This mix indicates that while GitHub remains prominent for code hosting, much of the smart contract discussions, learning, and technical reference happen on generic domains, reflecting the scraped nature of the smart contract development ecosystem. On the contrary, further platforms like GitLab and Bitbucket yielded no successful outcomes via direct or indirect query methods, emphasizing the limitations in visibility and accessibility on those platforms.

Table 1
Top 20 domains from most used smart contract Google queries

Domain		Domain	
github.com	1552	swcregistry.io	262
ethereum.stackexchange.com	1345	coinsbench.com	241
forum.openzeppelin.com	1235	metana.io	219
stackoverflow.com	1167	docs.blockscout.com	200
docs.openzeppelin.com	657	etherscan.io	196
medium.com	538	www.geeksforgeeks.org	189
www.oreilly.com	517	www.cyfrin.io	181
docs.soliditylang.org	498	gnosisscan.io	157
gist.github.com	311	www.quicknode.com	140
metaschool.so	291	huggingface.co	134



(a) Issue Tracker Adoption



(b) Issues distribution

Figure 1: Issue management comparison between SCs and traditional SWs.

Concerning RQ$_2$, out of 42 SC-related projects found on GitHub, three have the issue tracker disabled (7%), 18 have not received any issue reported (40%), and 21 feature at least one issue report (52%). Figure 1a compares these results with those from traditional software (SWs) [16], where 3.8% projects have the issue tracker disabled, 66% have no issues received, and 30% feature at least one issue report. Therefore, SC projects tend to take advantage of the issue-tracking systems more. Instead, Figure 1b provides a picture of issue distributions. For SC-related projects, (i) 39 repositories have less than 50 issues reported, 93% against the 86% of traditional software, (ii) no repository has between 50 and 100 issue reported, against the 7% of traditional software, (iii) one repository has between 100 and 500 issues, 2% against the 6.31% of traditional software, and (iv) two repositories have more than 500 issues, 5% against the 1.14% of traditional software. The distribution of SC-related projects is not homogeneous in the intermediate ranges, showing a high proportion of projects with very few issues, while a few projects show higher issue rates. This highlights developers' and users' trends to consider more popular projects, leading to a more prominent reporting of bugs, requests, and suggestions, with lesser-known projects resulting in few reports.

4.2. Label analysis and issue characteristics in smart contract-related projects

Table 2
Top 10 popular Tags on GitHub Issue Reports for SCs projects

Tag	# of Tagged Issues	% of Tagged Issues
enhancement	1101	29%
bug	760	20%
apps:closed	342	9%
tools:old	314	8%
libraries	222	6%
build	142	4%
apps:cmd	99	3%
apps:export	88	2%
indexing	72	2%
epic	69	2%

To answer RQ_3, we analyzed the $3,793$ issues extracted through Perceval from the 42 SC-related repositories, examining the labels (i.e., customizable tags used to classify an issue) assigned to each issue report and reporting a total of $4,126$ labels. Table 2 reports the top-10 frequent tags: the most used are *enhancement* and *bug*, which occur in 29% and 20% of cases, respectively. The comparison with traditional software (see Figure 2) corroborates the trend to use an issue tracker to mainly request enhancements, while bugs have similar distributions in both contexts. Although the relevant presence of bug reports is expected as SCs generally handle financial transactions or sensitive data, the high frequency of enhancement requests likely reflects the dynamic nature of the blockchain ecosystem. As this ecosystem evolves rapidly, developers and users propose improvements to adapt to emerging standards or optimize existing functionalities to keep pace with technological advancements and changing user needs.

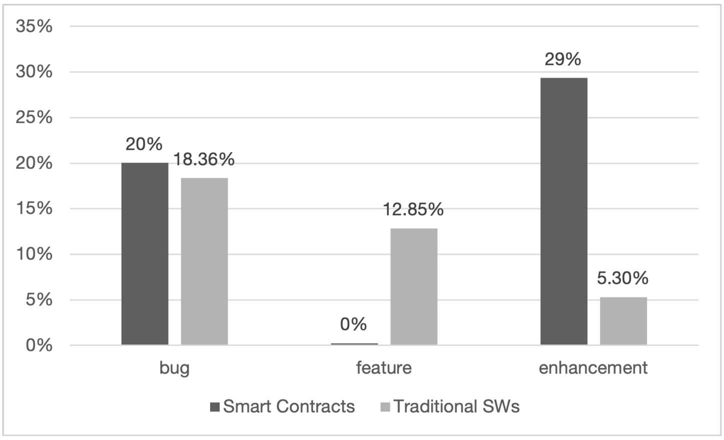


Figure 2: Labels frequency comparison with traditional SWs

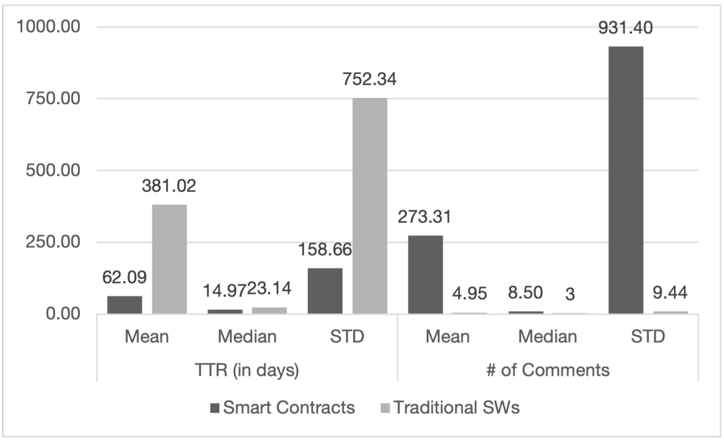


Figure 3: Issue characteristics compared with traditional SWs

To answer RQ_4, we considered the Time To Resolve (TTR) in days and the number of comments of closed issues. The collected statistics are compared with the work of Sülün et al. [17], whose main scope is to inspect how the adoption of templates affects issue management. Out of the 42 SC-related projects, only one employs issue templates. Therefore, the comparisons of Figure 3 consider issues in projects with no templates available. The histograms of Figure 3 reveal that issue resolution in SC-related projects generally requires less time than traditional software despite the high number of comments received. However, the high standard deviation value suggests that large projects notably impact results.

A manual inspection of the issue reports in our dataset reveals recurring themes. In particular, several reports [18, 19], even unlabeled [20, 21, 22, 23, 24], concern bugs, highlighting that the percentages in Figure 2 are underestimated. Security-related concerns [25, 26] also emerge, including potential vulnerabilities, typical blockchain attacks, and security flaws. Other issues [27] regard development workflows, emphasizing the increasing complexity of managing smart contract ecosystems. Instead, enhancement requests are mainly oriented toward testing improvements [28], gas optimization [29, 30] and general code quality refinement [31, 32]. Finally, our manual inspection confirms that SC developers follow a collaborative approach to issue resolution, making intensive use of issue comments [33, 24].

5. Conclusion

Nowadays, the scientific community has started investigating the general development dynamics and problem management practices in the smart contract ecosystem, as the widespread diffusion of this technology has inevitably led to an increase in specific challenges developers and smart contract users have to address. In this context, this paper explores the characteristics of issue reports and their management in SC-related projects, comparing them with those of traditional software projects. To conduct this analysis, we built a dataset of public data containing smart contracts from the Ethereum blockchain, considering the programming language, source code accessibility, and their degree of activity. As a preliminary investigation, the top 2000 SCs in terms of transactions carried out were selected. However, only 42 of them have their source code publicly developed and maintained on the GitHub platform. Leveraging the information stored in the issue trackers of these projects, we gathered the data required to pursue the statistical analysis. The collected data were compared with those from previous literature concerning traditional software, emphasizing differences between the two areas within the limits of the small size of the experimental dataset. Our results remark that SC developers and users employ the issue-tracking system more than traditional software. However, the vast majority of SC-related projects receive less than 50 issues; only a few projects receive more than 500 issues, lacking an intermediate range. Furthermore, while the rate of bug reports is similar for the two contexts, the study highlighted that, in SC-related projects, there is a greater demand for enhancements compared to traditional software. However, among SC projects, we also found a scarce adoption of standardized issue templates. Standardized issue templates and contribution guidelines would ensure higher consistency and clarity in reporting, likely enhancing interactions and collaborative development. Finally, there is a general trend to resolve issues earlier than traditional software, with an extensive use of comments in the resolution process. Therefore, the integration of communication and traceability tools like Gitter or Discord would allow for dynamic and threaded conversations, ensuring more transparent, efficient, and audit-friendly development workflows.

5.1. Limitations and Future Work

The reported results are subject to several limitations that should be addressed in future work. First, the process used to match smart contracts with their corresponding GitHub repositories (see Section 3.2) may lack completeness and precision. Refining the search methodology, both by improving matching heuristics and incorporating additional collaborative platforms such as GitLab (with premium functionalities), could increase the effectiveness and accuracy of the dataset. Besides, the analysis is based on Ethereum smart contracts only, with just 42 repositories out of the top 2,000 most used contracts publicly available and maintained on GitHub. This small subset (2.1%) may introduce a significant selection bias and limit the findings' generalizability. Further, popular contracts are inherently more likely to receive issue reports, potentially growing metrics related to issue frequency and management activity. To mitigate these limitations, we plan to extend the dataset both in scale and variety. This expansion will involve incorporating a substantially larger number of smart contracts, not only from Ethereum but also from other leading blockchain platforms such as Binance Smart Chain and Solana. This would ensure a more representative and reliable understanding of the smart contract development landscape, enabling the analysis of a broader range of development and maintenance practices.

Beyond the dataset issue, there is also a noteworthy possibility to pull practical insights from the issue reports themselves. Future directions should investigate the nature and timing of reported issues to identify common error patterns, recurring feature requests, and the stages of the typical problem-solving cycle.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] A. Di Sorbo, S. Laudanna, A. Vacca, C. A. Visaggio, G. Canfora, Profiling gas consumption in solidity smart contracts, Journal of Systems and Software 186 (2022) 111193.

[2] M. Pacheco, G. Oliva, G. K. Rajbahadur, A. Hassan, Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform, ACM Transactions on Software Engineering and Methodology 32 (2023) 1–46.

[3] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.

[4] J. Chen, X. Xia, D. Lo, J. Grundy, X. Yang, Maintenance-related concerns for post-deployed ethereum smart contract development: issues, techniques, and future challenges, Empirical Software Engineering 26 (2021) 117.

[5] P. Maidamwar, N. Chavhan, Blockchain technology: a review on architecture, security issues and challenges, in: International Conference on IoT, 2023.

[6] M. H. Miraz, M. Ali, Blockchain enabled smart contract based applications: Deficiencies with the software development life cycle models, arXiv preprint arXiv:2001.10589 (2020).

[7] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, B. Xu, Smart contract development: Challenges and opportunities, IEEE transactions on software engineering 47 (2019) 2084–2106.

[8] N. Kannengiesser, S. Lins, C. Sander, K. Winter, H. Frey, A. Sunyaev, Challenges and common solutions in smart contract development, IEEE Transactions on Software Engineering 48 (2021) 4291–4318.

[9] G. A. Pierro, R. Tonelli, Analysis of source code duplication in ethreum smart contracts, in: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2021, pp. 701–707.

[10] N. He, L. Wu, H. Wang, Y. Guo, X. Jiang, Characterizing code clones in the ethereum smart contract ecosystem, in: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24, Springer, 2020, pp. 654–675.

[11] G. Wu, H. Wang, X. Lai, M. Wang, D. He, S. Chan, A comprehensive survey of smart contract security: State of the art and research directions, Journal of Network and Computer Applications (2024) 103882.

[12] M. Vaccargiu, S. Aufiero, S. Bartolucci, R. Neykova, R. Tonelli, G. Destefanis, Sustainability assessment of ethereum developers issues and comments: Topic and network analysis, Preprint (2025).

[13] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, Y. Smaragdakis, Madmax: Surviving out-of-gas conditions in ethereum smart contracts, Proceedings of the ACM on Programming Languages 2 (2018) 1–27.

[14] A. Lima, L. Rossi, M. Musolesi, Coding together at scale: Github as a collaborative social network, in: Proceedings of the international AAAI conference on web and social media, volume 8, 2014, pp. 295–304.

[15] S. Dueñas, V. Cosentino, G. Robles, J. M. González-Barahona, Perceval: software project data at your will, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018, 2018, pp. 1–4.

[16] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, Y. Le Traon, Got issues? who cares about it? a large scale investigation of issue trackers from github, in: 2013 IEEE 24th international symposium on software reliability engineering (ISSRE), IEEE, 2013, pp. 188–197.

[17] E. Sülün, M. Saçakçı, E. Tüzün, An empirical analysis of issue templates usage in large-scale projects on github, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–28.

[18] Github.com, Trueblocks/trueblocks-core issue #3587, https://github.com/TrueBlocks/trueblocks-core/issues/3587, 2024.

[19] Github.com, Trueblocks/trueblocks-core issue #3914, https://github.com/TrueBlocks/trueblocks-core/issues/3914, 2024.

[20] Github.com, datachainlab/ethereum-ibc-relay-chain issue #41, https://github.com/datachainlab/ethereum-ibc-relay-chain/issues/41, 2024.
[21] Github.com, kryptoklob-deprecated/zethr-issue-tracker issue #2, https://github.com/kryptoklob-deprecated/Zethr-Issue-Tracker/issues/2, 2018.
[22] Github.com, larvalabs/cryptopunks issue #3, https://github.com/larvalabs/cryptopunks/issues/3, 2024.
[23] Github.com, Nibiruchain/nibiru issue #2252, https://github.com/NibiruChain/nibiru/issues/2252, 2025.
[24] Github.com, Nibiruchain/nibiru issue #2158, https://github.com/NibiruChain/nibiru/issues/2158, 2025.
[25] Github.com, aragon/aragon-network-token issue #23, https://github.com/aragon/aragon-network-token/issues/23, 2020.
[26] Github.com, larvalabs/cryptopunks issue #31, https://github.com/larvalabs/cryptopunks/issues/31, 2024.
[27] Github.com, radicle-dev/radicle-contracts issue #130, https://github.com/radicle-dev/radicle-contracts/issues/130, 2021.
[28] Github.com, bcnmy/mexa issue #14, https://github.com/bcnmy/mexa/issues/14, 2020.
[29] Github.com, balancer/exchange-proxy issue #3, https://github.com/balancer/exchange-proxy/issues/3, 2020.
[30] Github.com, bokkypoobah/bokkypoobahstokenteleportationservicesmartcontract issue #10, https://github.com/bokkypoobah/BokkyPooBahsTokenTeleportationServiceSmartContract/issues/10, 2018.
[31] Github.com, hexoul/ether-stealer issue #2, https://github.com/hexoul/ether-stealer/issues/2, 2018.
[32] Github.com, bokkypoobah/bokkypoobahstokenteleportationservicesmartcontract issue #8, https://github.com/bokkypoobah/BokkyPooBahsTokenTeleportationServiceSmartContract/issues/8, 2018.
[33] Github.com, larvalabs/cryptopunks issue #19, https://github.com/larvalabs/cryptopunks/issues/19, 2024.