Applicability of the software Security code metrics for Ethereum smart contract using Solidity

N'Da, Aboua Ange Kevin; Matalonga, Santiago; Dahal, Keshav

Published in:
2021 International Conference on Deep Learning, Big Data and Blockcain (DEEP-BDB)

Accepted/In press: 24/05/2021

Document Version
Peer reviewed version

Applicability of the software Security code metrics for Ethereum smart contract using Solidity

Abstract. The Ethereum blockchain allows, through software called smart contract, to automate the contract execution between multiple parties without requiring a trusted middle party. However, smart contracts are vulnerable to attacks. Tools and programming practices are available to support the development of secure smart contracts. These approaches are effective to mitigate the smart contract vulnerabilities, but the unsophisticated ecosystem of the smart contract prevents these approaches from being foolproof. Besides, the Blockchain immutability does not allow smart contracts deployed in the Blockchain to be updated. Thus, businesses and developers would develop new contracts if vulnerabilities were detected in their smart contracts deployed in Ethereum, which would imply new costs for the business. To support developers and businesses in the smart contract security decision makings, we investigate the applicability of the security code metric from non-blockchain into the smart contract domain. We use the Goal Question Metric (GQM) approach to analyze the applicability of these metrics into the smart contract domain based on metric construct and measurement. As a result, we found 15 security code metrics that can be applied to smart contract development.

Keywords: Security, Metric, QGM, smart contract, Ethereum, Blockchain

1 Introduction

Blockchain technology has started gaining more attention from businesses and governments after the introduction of Bitcoin [1]. Defined as a shared public ledger that stores transactions in a decentralized peer-to-peer network of computers, Blockchain guarantees the transaction execution without third-party intervention. Initially limited to peer-to-peer payment-based transactions [1], Blockchain now through platforms like Ethereum [2] and Hyperledger [3], can also be applied in many other domains[4][5].

For this study, we focus on the Ethereum Blockchain. The Ethereum Blockchain enables through computing protocols, known as smart contracts, to verify and enforce contract negotiation on the top of the Blockchain. The smart contract can be built and deployed by anyone using Solidity – one of the Turing complete language provided by Ethereum. However, it is publicly accessible [2]. Therefore, the smart contract code is exposed to the attackers, which can lead to attacks on the contracts such as the attack on DAO contract [6]. This attack led to a loss of 3.6m Ether. The growing number of attacks on Ethereum has forced businesses and researchers to incorporate the aspect of security during the development process of the smart contract. Verification tools [7]–[10] and programming practices and patterns [11]–[13] are currently available in the literature to support this process. However, these approaches are not effective enough to being foolproof. They do not fully cover the security of the smart contract. Thus, given Blockchain immutability, identifying critical vulnerabilities in a deployed contract would lead

the developer or business to create and deploy a new contract. Therefore, an indication of smart contract security is necessary to help developers and businesses in decision-making before deploying smart contracts.

In non-blockchain environement, software security metrics are described as good indicators for the security of the software. To the best of our knowledge, there are no security metrics for the Ethereum smart contract domain.

We investigate the applicability of the existing security code metric in non-blockchain environement into Ethereum smart contract domain for contract build with Solidity. By using the Goal Question Metric (GQM) approach we analyzed the applicability of the security code metric from the non-blockchain into the Solidity smart contract domain based on metric construct and measurement. We found 15 among the list of 20 security code metrics are applicable for the Ethereum smart contract with respect to both construct and measurement. These metrics can be used by developers or organizations during smart contract security development.

2 Background

2.1 Ethereum smart contract and Solidity

Blockchain is an encrypted transaction-based-ledger running on the top of a decentralized peer-to-peer network of computers known as nodes [1]. Guaranteeing the transparency, immutability, anonymity of the transaction through Bitcoin (first implementation of the Blockchain), Blockchain has been implemented under new platforms such as Ethereum. Defined as a second-generation of Blockchain, Ethereum implements computing protocols known as smart contract on the top of the Blockchain for the transaction flexibility[2]. A smart contract is software running on top of the Blockchain. It is executed in a virtual machine (EVM) provided by Ethereum located in each node [14]. It aims to encode rules to reflect any kind of multi-party interactions. Smart contract is represented in the Ethereum environment by an account which consists of a unique address and account state with the amount of Ether (balance), storage, and code hashes respectively pointing toward storage memory in EVM and the contract code in Blockchain ledger as shown Fig 1. Locating in the Blockchain ledger, the smart contract code stays immutable, thus preventing a malicious entity from tampering with the code to get the control of the smart contract. However, the immutability of the contract code does not allow the smart contract to rely on the standard life cycle of software. The maintenance of the smart contract is impossible. The developer could not update a contract program containing errors unless a new version of the contract is released.

For the contract development, the developer can refer to Solidity – a Turing-complete programming language provided by Ethereum. Solidity's syntax is quite similar to JavaScript and Java language. Solidity as the Java and Javascript language supports some Objected oriented constructs and procedural constructs [15].It also supports coupling, cohesion and inheritance of the contracts. Solidity based-contracts are executed through the EVM bytecode generated by a Solidity compiler, which bytecode is interpreted by the EVM.

2.2 Smart contract security

Smart contracts are increasingly being adopted to implement applications in many sectors such as financia [16], health [4] and IoT [5]. Smart contracts enable the development of potential new business models but also introduces computing infrastructures capable of defect, error, and failure.

According to Delmolino et al. [17], after analysing various smart contract application source codes developed by students, they identified smart contract source codes are victim to common pitfalls such contracts prone to logical errors, fail to use cryptography, they do not incentive user to the expected behaviour, and not aware Ethereum specific bug such as Call-stack bug.

Similarly, Luu et al. [7] highlighted gaps in the understanding of the distributed semantics of the Ethereum platform, which favours smart contract attacks. They introduce new security vulnerabilities in smart contracts and show possible attacks based on these vulnerabilities. Atzei et al. [18] provide a systematic classification of smart contract vulnerabilities according to the vulnerability source. They show Ethereum Blockchain, EVM and Solidity are sources of smart contract vulnerability. Similarly, Chen et al. [19]show that Ethereum blockchain, Solidity and EVM are the sources of vulnerability by providing a taxonomy of smart contract vulnerability after analysing 44 vulnerabilities which 19 of those vulnerabilities are caused by Solidity language and the misunderstanding of the programming with Solidity. They suggest the development of more secure programming languages and more secure supporting tools for smart contract security.

The security which is becoming a critical aspect of the smart contract development remains the prerogative of stakeholders to verify the correctness and fairness of the smart contract. Thus, research works provide tools to analyse the smart contract vulnerabilities. Luu et al. [7] provide an OYENTE - static analysis tool using symbolic execution for detecting bugs at the bytecode level of the smart contract. However, this tool does not offer sufficient enough to detect most vulnerabilities identifies in a smart contract such as integer overflow/underflow [10]. Tsankov et al. [8] developed a static analysis tool, called SECURIFY, using both symbolic execution and filter of predefined compliance and violation patterns to explore all the contract behaviours (avoiding false negatives) and also to avoid false positives. However, this tool has limitations. For instance, SECURIFY is based on properties that do not capture all the violations that might be exploited by attackers, leading to certain vulnerabilities remaining in smart contracts.

Another approach for analysing the program is to apply formal verification which incorporates mathematical models to make sure that the code is free of errors. Bhargavan et al. [20] conducted a study on smart contracts using this approach. They outline a framework able to parse the Solidity source and EVM byte code into a functional programming language F*, to proceed to the smart contract verification. Similarly, Amani et al. [9] presented a verification tool of the smart contract based on Isabelle/HOL proof assistant. However, these formal verification tools do not take into account all the EVM semantics. For instance, the properties based on inter-message calls of contracts are not supported by the tool provided by Amani et al. [9] leading the related vulnerabilities to remain in smart contracts.

Some research works have been conducted to help the developers to adopt security practices and patterns for securing the smart contracts. Security practices

and patterns refer to realistic and enforceable actions that address the most common software security issues. Whorer and Zdun [11] identified six security design patterns based on the grounded theory techniques for the smart contract with Solidity. Mavridou et al. [12] proposed two security patterns that can be implemented as plugins to facilitate their application. Regarding the security programming practices, N'Da et al [13] conducted a study for characterizing the cost of the security programming practices in the smart contract. As a result, they identified a list of 30 security programming practices from JAVA and C++ that can be used for smart contract security.

2.3 Metrics for measuring software security

In the non-Blockchain environement, the use of metrics has received a lot of attention. Metrics provide information on the software which can be analysed and utilised in business decisions. A software metric is defined as a quantitative measure of a given software property, which differs from a measurement used in some literature to represent a metric[21],[22]. Research studies assessed the efficiency of the software metric in security vulnerability prediction. Nguyen and Tran[23] evaluated the semantic complexity of the software in vulnerability detection through complexity metrics. They extracted semantic complexity metrics from dependency graph on Mozilla JSE software. By using classification AI techniques(Naïve Bayesian, Random Forest, Neural Network and Bayesuian Network) for these complexity metrics evaluation, their result showed the false negative(FN) rate can be reduced of 89% for nesting metrics to 40%.[24]et al investigated the usefulness of metrics such us complexity, code churn and developer activity for vulnerability prediction. By using three prediction model of each group and a model combining these metrics based on discriminant analysis and Bayesain network, they showed these models were able to predict about 70% of vulnerable files from two open sources projects (Mozilla Firefox, and Linux Kernel) with accuracy lower than 5%. Moshtari et al[25] replicated the shin et al study by considering more complete vulnerabilities and cross-project vulnerability that were not considered by Shin et al. They evaluated complexity metrics in the prediction of vulnerability. Their result showed about 92% of the vulnerable files with false positive (FP) of 0.12% are detected in the Mozilla Firefox project. And complexity metrics could detect about 70% of vulnerable files with tolerable false positive (FP) rate of 26% for cross-project vulnerability prediction on five open source projects. Chowdhury and Zulkernie[38]used Coupling, cohesion, and complexity metric to predict vulnerability. They performed the experiment on 52 releases on the Mozilla Firefox and through the use of C4.5 Decision tree, Random Forest, Logistic regression and Naives Bayes, the Coupling and Cohesion metric were able to predict correctly 75% of the vulnerable files with false-positive rate less than 30%.

Besides, security metrics were also proposed and used to assess security at the early stage of software development. Sultan et al. [26] provided a catalog of metrics for assessing security risks of the software through the Software Development Life Cycle (SDLC). They defined metrics for requirement, design, implementation, and maintenance phases by using the GQM approach. Metrics are also defined to characterise the security vulnerabilities of software systems. Based

on Common Vulnerability Scoring System (CVSS) research [27],[28] provided security metrics to assess software system vulnerability.

2.4 Metric in smart contract domain

In this section, we provided an overview of the reported researches based on the metrics for the smart contract context.

There are few researches works using metrics in the smart contract domain. Hegedus [29] proposes a set of metrics derived from non-blockchain metrics to help developers during the Solidity smart contract development. The author also provides a tool called Solmet to collect them. The study was able to provide an overview of the structure of smart contracts in terms of size, complexity, coupling, and inheritance properties. The result of his study shows these metrics in the context of smart contracts have lower values compared with the context non-blockchain programs.

Similarly, Perio et al. [30] propose a fully web-based tool called PASO, which is able to compute smart contract metrics. They discussed a few numbers of software code metrics derived from OO metrics in non-blockchain environement and also metrics specifics to Solidity language. Some of their OO metrics have been already discussed by Hegedus [29].

Vandenbogaerde [31] proposes a graph-based framework for computing OO design metrics from non-blockchain environement into the smart contract context. Their framework allows analysing the design of Solidity smart contract through the simple queries which can extract the contract function and design metric from a generated graph-based semantic meta-model. After implementing this framework to a list of contracts, the authors mentioned that most of the contracts in solidity have some similarity to java practice when creating class, and also that contract coupling and inheritance are less utilized in smart contracts.

Unlike Hedegus and Vandenbogaerde whose study was just oriented smart contract structure through metric, [32]investigated OO metrics for smart contract structure as well as their correlation with gas consumption in smart contracts. The author uses Solmet, Truffle suite [33] to extract respective OO metrics and deployment cost of the list of contacts from a Github project. By using Spearman's rank correlation method, the author shows there is a statistically significant correlation between some of the OO metrics and the resources consumed on the Ethereum Blockchain network when deploying the contract.

In contrast to these papers, our study focuses on the security aspect of the smart contract by investigating the security code metrics from non-blockchain environment for the solidity smart contract.

3 Applicability of software security metrics for Ethereum smart contracts

In this section, we present an investigation of the applicability of security metrics for solidity smart contracts. To design our research we exploit the GQM approach [34]. This section describes the research method and results.

3.1 Method and process

Our goal is to characterise the applicability of security metrics for Ethereum smart contracts written in Solidity. To achieve this, we will evaluate the potential applicability of the non-blockchain security metrics into the Ethereum smart contracts written in Solidity. **Error! Reference source not found.** **Table 1.** GQM model for applicability of the security source code metrics in smart contract contextpresents the GQM approach used to design this research [34].

Table 1. GQM model for applicability of the security source code metrics in smart contract context

Purpose:	
Analyze	Source code metrics for measuring security in non-blockchain software development
With the purpose of:	Characterizing their applicability in smart contract development
With respect to:	The applicability of Construct: Identification of built concepts around the metric and their potential interpretation in the Blockchain domain The capability of measurement: getting the metric measures based on the measurement process in the targeted domain
Point of view:	Researcher
In the context of:	The software coding process of smart contracts in Solidity for the Ethereum Blockchain platform. Subjects are smart contracts obtained from Ethereum and published peer-review papers.

We define the following two research questions and their derived metrics based on these research needs, developed through the GQM.

Research Question 1 (Applicability) Is it possible to identify and interpret the constructs embedded in the security metrics and interpret them in the Solidity smart contract domain? – Metric: Qualitative Judgement and "Yes/No" dictum.

Research Question 2 (Capability) Is it possible to perform the measurement (defined in the metric) in the Solidity smart contract domain? – Metric: Qualitative Judgement, "Yes/No" dictum, and corresponding corroborating evidence

Instrumentation

To answer **Research question** 1 (related to the construct), a literature review was conducted to gather the candidate security metrics. We used snowballing method guidelines proposed by [35] for getting the relevant quality papers related to the security code metrics. Hence for the start set we refer to the known datasource such as academic literature from online database, conference proceeding, academic journal. From those sources, we looked for literature respectively according the input based on the following string expression: software metric-security metric- security measurement-software security metric-security code metric. As result, we could identify a list of 16 literature related to the security code metric.

Table 2 The start set for snowballing of the research study

No	Paper
1	K. Sultan, A. En-Nouaary, and A. Hamou-Lhadj, "Catalog of metrics for assessing security risks of software throughout the software development life cycle," Proc. 2nd Int. Conf. Inf. Secur. Assur. ISA 2008, pp. 461–465, 2008
2	J. A. Wang, H. Wang, M. Guo, and M. Xia, "Security metrics for software systems," Proc. 47th Annu. Southeast Reg. Conf. ACM-SE 47, 2009.
3	Chowdhury and M. Zulkernine, "Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?," Proc. ACM Symp. Appl. Comput., pp. 1963–1969, 2010.

After a deep analysis of the latter, we considered the paper as candidates for inclusion, those that focus on source code security, publicly refereed, and which metrics presented are well defined and provide evidence that they measure the security. For the well-defined metric, we follow the metric pattern below:

- **Metric name**: given name of the metric
- **Subject**: The type and the description of the entity being measure
- **Attribute**: property of the feature from the subject being measure
- **Measurement**: process describing how the subject attribute is measured
- **Validation approach**: validation process providing to evaluate the reliability of the metric base on the attribute being measure

Thus, we decided to exclude 13 papers because some are out of our scope and the others are from the same authors. Therefore we use as candidates for inclusion 3 remain paper for the start set as shown in **Table *2***.Since the use of metrics to measure security are lately get attention[36][37], we decided for the time frame of the start set to focus on research between 2007 and 2010 as shown the **Table *2***.

From these 3 papers both backward and forward snowballing were conducted. For the backward snowballing, for each paper, we consider their reference list which title is related to "the security code metric", "security code measurement". Then we examined the full text of the paper seeming relevant in alignment with our inclusion criteria mentioned above. For the forward snowballing, for each paper, we look at the paper citing it and applied the same approach based on the inclusion criteria mentioned above to select the new relevant paper for our study. We continue that process until we were not able to find more relevant papers for our study scope.

In total, we were able to identify a list of 9 relevant paper focusing on the security code metric and which security measurement have been proved with evidence. From these 9 relevant papers, we identified a list of 20 security code metrics as the candidates of security code metrics for our study. For each security code metric, we applied a judgment and our experience to critically appraise their construct. Our judgment is based on the interpretation of the metric constructs in the smart contract environment. Hence, we deemed applicable a metric based on the construct when its construct can be interpreted in Ethereum smart contract environment.

To Answer **Research question 2** (related to measurement), we analyse the measurement process against the smart contract environment for each security metric used to answer RQ1 (related to the construct). If the measurement of a security code metric can be performed in the context of the smart contract, to ensure the consistency of our judgment, we try to collect its measure from two testimonial contracts. The applicability of the metric is decided based on its measure. The two testimonial contracts used in this study are E-voting and Supply chain contracts. The E-Voting contract is provided by the Ethereum platform[38]. The second one is a module contract for the supply chain that consists of two smart contracts developed by another research study [39]. We claim that these two smart contracts provide a representative sample of smart contracts in Ethereum. Firstly, the E-voting development is independent of this research. This is a single smart contract sample having a simple structure design as required by Ethereum (compare to traditional software), representing then most of the smart contracts that are in Ethereum. Secondly, the supply chain smart contract has been validated by the research community and (comparing to the E-voting) provides attack surfaces that stem the interaction between contracts.

3.2 Result

In this section, we present the results of the analysis deriving from the applicability of the security code metric from non-blockchain into the smart contract domain with Solidity.

RQ1: Analysis of applicability of the security metrics base on the metric construct. To determine the applicability of the security code metric based on the construct in the smart contract context we proceed to the review of literature through which we identified 20 security code metrics as shown in **Table 3**. The identified metrics can be grouped by OO metrics (WMC, DIT, CBO, NOC, RFC, LCOM,), complexity metric (McCabe complexity, Halsted volume, CER), and exception metrics (NCBC, Rserr, EHF). For each metric, we identified and compared its construct against the smart contract environment by considering Solidity language constructs, Ethereum Blockchain mechanism, and theoretical basis. Finally, the construct of each metric is deemed to be applicable when the construct is represented (implemented) into the smart contract environment.

As a result, we found 15 security code metrics, shown in **Table 3**, for which the construct is applicable in the smart contract context.

Since space constraints limit our capacity to provide detail on each metric, this paragraph presents the type of discussions that lead our judgment. For instance, the constructs of metrics such as DIT and CBO are based on the Object-Oriented concepts: Coupling, inheritance, cohesion program [40], [41], which can be readily interpreted in the development of the smart contract with Solidity given those concepts are used in Solidity[15]. Therefore, we accept these metrics to be applicable. In contrast, the metric construct of the VBW [22] metric (that stands for Vulnerability Based Weakness) cannot be applied in Solidity. Indeed, the VBW construct implies standard lists. The first one is CWE [42], which is a list of Common Weaknesses related to software. The second one is [43], which is Common Vulnerabilities related to software. These two lists are linked by the fact

that weakness can imply vulnerability (ies). Therefore, for a specific weakness in CWE, the list of vulnerabilities from the CVE can be identified. However, in the smart contract context, currently, there is no standard common weaknesses list from which smart contract vulnerabilities are associated. There is a standard common weakness list implementation for smart contracts named SWC [44] (stand for Standard Weakness classification) similar to CWE, but unlike CWE, it does not associate vulnerabilities to weaknesses. Therefore, in absence of CWE and CVE list in the smart contract context, we claim that this metric construct is not applicable in the smart contract.

Table 3. List Security code metric in the traditional (non-blockchain) environement

Metric name	Applicability of construct? (M1)	The capability of measurement (M2)
VBW[22] **(vulnerability based weakness)**	NO	NO
WMC[21], [41] **(Weigthed Method Per Class**	YES	YES
DIT[21], [41]**(Depth of Inheritance)**	YES	YES
NOC[21], [41]**Number of Children)**	YES	YES
CBO[21], [41]**(Coupling between objects class)**	YES	YES
RFC[21], [41]**(Response for the Class)**	YES	YES
LCOM[21], [41]**(Lack of Cohesion in Method)**	YES	YES
Stall Ratio[45]	YES	YES
CER[45]	YES	YES
Nerr[26]	NO	NO
Nserr[26]	NO	NO
Rserr[26]	NO	NO
Nex[26]	YES	YES
Noex[26]	YES	YES
Roex[26]	YES	YES
McCabe Complexity(CCM)[24][46]	YES	YES
Halstead's volume metric[47], [48]	YES	YES
CCP[45]	YES	YES
NCBC[49]	YES	YES
EHF[49]	NO	NO

RQ2: Analysis of the capability of measurement of the security metrics. -To answer this research question, we followed the metric definition to obtain a measurement of the smart contracts under study. We were able to obtain the measurement for 15 metrics from the 20 metrics from **Table 3**. Therefore, we conclude that smart contracts in Solidity are capable of providing the elements to measure these metrics.

As shown the **Table 4**, we were able to determine the value of the metrics for E-voting and Supply chain contracts by following the measurement process of

these 15 metrics. Like in the previous section, we cannot provide a detailed account of how each measurement is taken due to the space concerns. As a representative example, the NCBC metric measurement consists to count the number of exception handling statements (try-catch) in the program, which can be identified in the smart contract context by determining the number of exception handling statements used in Solidity. These exception statement are: (revert(), required(), assert(), etc…).

The reader will notice from **Table 4** that some metric measures resulting in a value of 0. We highlight that this does not challenge the answer of our research question. The high-level explanation for these results is that the smart contracts used here do not provide the complexities (or vulnerabilities) that those metrics intend to count. We will discuss this further in the next section.

Table 4. Measures of applicable security code metrics based on measurement from E-voting and supply chain contracts

Metric	E-voting	Supply chain(Bidding)	Supply chain value(Tracking)
WMC	12	11	11
DIT	0	0	0
NOC	0	0	0
CBO	0	1	1
RFC	5	18	23
LCOM)	0	0	0
Stall Ratio	0	0	0
CER	1	0.78	0.9
Nex	0	13	14
Noex	7	5	8
Roex	1	0.28	22
McCabe Complexity	3	1.58	1.38
Halstead's volume metric	1850.15	3134.25	2589.4
CCP	0	0	6
NCBC	0	0.72	0.64

4 Discussion

In this section, we present some points that stem from our research results.

Applicability of the security code metric construct into smart contract-We analysed a list of 20 security code metrics from a review of the literature and found that 15 of these metric constructs are applicable to the smart contract domain. These applicable security metrics based on construct are generally related to the

complexity, coupling and cohesion, inheritance, and exception handling of the program code. These elements of the source code have been empirically shown to be suitable proxies for measuring source code security [21][49]–[51]. Regarding the smart contract context, though there is little (or none) empirical evidence, it is reasonable to expect that the relationship between security and these aspects is kept. For instance, a smart contract with high complexity in code structure might lead to security vulnerabilities. So, with this information, the smart contract development team will take proactive steps to deal with the complexity of the smart contract code to keep the contract secure before it is deployed.

Measuring capacity and automation in smart contracts-The result from the analysis of the measurement of the non-blockchain security code metric reveals that the measurements of 15 metrics are applicable to smart contracts. Regarding automation, the only WMC, DIT, CBO and McCabe complexity can be collected automatically using Solmet tool [29]. However, for the rest of the metrics in **Table 3**, to the best of our knowledge, there is not yet any automated tool to collect them in the smart contract context. They can currently be collected through a manual process as shown in 1 and 2.This is a laborious task for developers. For instance, the Halstead volume measurement which is applied manually to an Ethreum smart contract requires identifying all operators and operands of the program code. Thus, the higher the program size will be, the harder the measure collection of the Halstead volume is going to be. The automation of these security metrics is needed to help the developer and businesses to manage the security of the smart contract more effectively before deploying them.

Consistency of the metric measurements-**Table 4** shows the measures of the DIT, NOC, stall ratio and LCOM equal 0 for all the testimonial contracts. In fact, the reason for the measures of the DIT and NOC being equal to 0 is that these metrics use inheritance as a proxy for security. None of the testimonial contracts used inheritance. Therefore, the measures resulted in 0. It is expected that, for a smart contract involving inheritance, applying this measurement will provide a non-zero measure. Similarly, Stall ratio and LCOM have 0 as value for both testimonial contracts. The Stall ratio looks at the statements in the loop which can delay the program (i.e log functions), but the codes for the both testimonial contracts do not contain loop statements which can enforce tardiness. Hence the measure of this metric for the testimonial contracts is equal to 0. Similarly, it can be expected that for a smart contract code involving a statement leading to the tardiness of contract execution, the measurement of this metric will be non-zero.

[1]https://github.com/ndaangekevin/security-code-metric-collection-for-smart-contract.-E-voting-casae-study

[2] https://github.com/ndaangekevin/security-code-metric-collection-for-smart-contract.-Supply-chain-casae-study-

5 Conclusion

We investigated the applicability of the non-blockchain security code metric into Solidity smart contract to help Ethereum developers and organizations to deal with the security decisions of the smart contracts. We used the QGM approach to design our research case study to achieve this goal. Our study reveals that 15 security code metrics from both construct and measurement are applicable in the smart contract context we provided evidence of their measurement. Therefore, we claim that these 15 metrics can be used by developers and organizations to manage smart contract security. Moreover, the study suggests that complexity, cohesion, coupling, and exception handling might impact the security of the smart contract.

The use of the proposed metrics from our study also presents a real challenge for developers. Most of the measurement processes of these metrics have to be performed manually. We, therefore, incentivize the research community to provide automated tools to collect these security code metrics to help developers and organizations to manage security issues during the development of smart contracts.

6 References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System."

[2] V. Buterin, "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM."

[3] E. Androulaki *et al.*, "Hyperledger fabric," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15.

[4] T. T. Kuo, H. E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *Journal of the American Medical Informatics Association*, vol. 24, no. 6. Oxford University Press, pp. 1211–1220, 01-Nov-2017.

[5] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*. 2016.

[6] "The DAO Attacked: Code Issue Leads to $60 Million Ether Theft - CoinDesk." [Online]. Available: https://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft. [Accessed: 21-Oct-2019].

[7] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2016, vol. 24–28–Octo, pp. 254–269.

[8] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2018, pp. 67–82.

[9] S. Amani, M. Bortin, M. Bégel, and M. Staples, "Towards verifying ethereum smart contract bytecode in Isabelle/HOL," in *CPP 2018 - Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, Co-located with POPL 2018*, 2018, vol. 2018–Janua, pp. 66–77.

[10] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing Safety of Smart Contracts," no. February, 2018.

[11] M. Wohrer and U. Zdun, "Smart contracts: Security patterns in the ethereum ecosystem and solidity," *2018 IEEE 1st Int. Work. Blockchain Oriented Softw. Eng. IWBOSE 2018 - Proc.*, vol. 2018–Janua, pp. 2–8, 2018.

[12] A. Mavridou and A. Laszka, "Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach," Nov. 2017.

[13] A. A. K. N'Da, S. Matalonga, and K. Dahal, *Characterizing the cost of introducing secure programming patterns and practices in ethereum*, vol. 1160

AISC. 2020.
[14] N. Szabo, “Smart Contracts: Building Blocks for Digital Free Markets,” *Extropy J. Transhuman Thought*, 1996.
[15] “Solidity Documentation Release 0.5.0 Ethereum,” 2018.
[16] P. Treleaven, R. G. Brown, and D. Yang, “Blockchain Technology in Finance,” *Computer (Long. Beach. Calif).*, 2017.
[17] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.
[18] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10204 LNCS, no. July, pp. 164–186, 2017.
[19] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses,” *ACM Comput. Surv.*, vol. 53, no. 3, 2020.
[20] K. Bhargavan *et al.*, “Formal verification of smart contracts: Short paper,” in *PLAS 2016 - Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, co-located with CCS 2016*, 2016, pp. 91–96.
[21] I. Chowdhury and M. Zulkernine, “Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?,” *Proc. ACM Symp. Appl. Comput.*, pp. 1963–1969, 2010.
[22] J. A. Wang, H. Wang, M. Guo, and M. Xia, “Security metrics for software systems,” *Proc. 47th Annu. Southeast Reg. Conf. ACM-SE 47*, 2009.
[23] V. H. Nguyen and L. M. S. Tran, “Predicting vulnerable software components with dependency graphs,” *6th Int. Work. Secur. Meas. Metrics, MetriSec 2010*, pp. 1–8, 2010.
[24] Y. Shin and L. Williams, “An empirical model to predict security vulnerabilities using code complexity metrics,” *ESEM’08 Proc. 2008 ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas.*, pp. 315–317, 2008.
[25] S. Moshtari, A. Sami, and M. Azimi, “Using complexity metrics to improve software security,” *Comput. Fraud Secur.*, vol. 2013, no. 5, pp. 8–17, 2013.
[26] K. Sultan, A. En-Nouaary, and A. Hamou-Lhadj, “Catalog of metrics for assessing security risks of software throughout the software development life cycle,” *Proc. 2nd Int. Conf. Inf. Secur. Assur. ISA 2008*, pp. 461–465, 2008.
[27] J. A. Wang, F. Zhang, and M. Xia, “Temporal metrics for software vulnerabilities,” *CSIIRW’08 - 4th Annu. Cyber Secur. Inf. Intell. Res. Work. Dev. Strateg. to Meet Cyber Secur. Inf. Intell. Challenges Ahead*, 2008.
[28] A. J. A. Wang, M. Xia, and F. Zhang, “Metrics for Information Security Vulnerabilities,” *J. Appl. Glob. Res.*, vol. 1, no. 1, pp. 48–58, 2008.
[29] P. Hegedűs, “Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts,” *Technologies*, vol. 7, no. 1, p. 6, 2019.
[30] G. Antonio Pierro and R. Tonelli, “PASO: A Web-Based Parser for Solidity Language Analysis,” *IWBOSE 2020 - Proc. 2020 IEEE 3rd Int. Work. Blockchain Oriented Softw. Eng.*, pp. 16–21, 2020.
[31] B. Vandenbogaerde, “A graph-based framework for analysing the design of smart contracts,” *ESEC/FSE 2019 - Proc. 2019 27th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 1220–1222, 2019.
[32] N. Ajienka, P. Vangorp, and A. Capiluppi, *An empirical analysis of source code metrics and smart contract resource consumption*. 2020.
[33] “Documentation | Truffle Suite.” [Online]. Available: https://www.trufflesuite.com/docs. [Accessed: 16-Nov-2020].
[34] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach,” *Encycl. Softw. Eng.*, vol. 2, pp. 528–532, 1994.
[35] C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” *ACM Int. Conf. Proceeding Ser.*, 2014.
[36] V. Patriciu and S. Nicolaescu, “Interdisciplinarity – New Approaches and

Perspectives in the Use of Quantitative Methods SECURITY METRICS FOR ENTERPRISE Interdisciplinarity – New Approaches and Perspectives in the Use of Quantitative Methods," *Comput. Eng.*, pp. 151–159.

[37] S. E. Schimkowitsch, "Key Components of an Information Security Metrics Program Plan," *Univ. Oregon*, no. July, p. 89, 2009.

[38] "Contracts — Solidity 0.5.3 documentation." [Online]. Available: https://solidity.readthedocs.io/en/v0.5.3/contracts.html. [Accessed: 11-Nov-2019].

[39] R. C. Koirala, K. Dahal, and S. Matalonga, "Supply chain using smart contract: A blockchain enabled model with traceability and ownership management," *Proc. 9th Int. Conf. Cloud Comput. Data Sci. Eng. Conflu. 2019*, pp. 538–544, 2019.

[40] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *J. Syst. Archit.*, vol. 57, no. 3, pp. 294–313, 2011.

[41] S. Chidamber and C. Kemerer, "MetricForOOD_ChidamberKemerer94.pdf," *IEEE Transactions on Software Engineering*, vol. 20, no. 6. pp. 476–493, 1994.

[42] "CWE - Common Weakness Enumeration." [Online]. Available: https://cwe.mitre.org/. [Accessed: 17-Nov-2020].

[43] "CVE - Common Vulnerabilities and Exposures (CVE)." [Online]. Available: https://cve.mitre.org/. [Accessed: 17-Nov-2020].

[44] "SmartContractSecurity/SWC-registry: Smart Contract Weakness Classification and Test Cases." [Online]. Available: https://github.com/SmartContractSecurity/SWC-registry. [Accessed: 14-Nov-2020].

[45] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," *Proc. - Int. Conf. Softw. Eng.*, no. June, pp. 57–64, 2008.

[46] T. J. Mccabe, "A Complexi-ty," no. 4, pp. 308–320, 1976.

[47] T. Hariprasad, G. Vidhyagaran, K. Seenu, and C. Thirumalai, "Software complexity analysis using halstead metrics," *Proc. - Int. Conf. Trends Electron. Informatics, ICEI 2017*, vol. 2018–Janua, pp. 1109–1113, 2018.

[48] M. Davari and M. Zulkernine, "Analysing vulnerability reproducibility for Firefox browser," *2016 14th Annu. Conf. Privacy, Secur. Trust. PST 2016*, pp. 674–681, 2016.

[49] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Software design metrics for object-oriented software," *J. Object Technol.*, vol. 6, no. 1, pp. 121–138, 2007.

[50] Y. Shin and L. Williams, "Is complexity really the enemy of software security?," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 47–50, 2008.

[51] R. A. Maxion, "Eliminating exception handling errors with dependability cases: a comparative, empirical study," *IEEE Trans. Softw. Eng.*, vol. 26, no. 9, pp. 888–906, 2000.