An Empirical Investigation on the Trade-off between Smart Contract Readability and Gas Consumption

Anna Vacca
University of Sannio
Benevento, Italy
avacca@unisannio.it

Michele Fredella
University of Sannio
Benevento, Italy
fredella@unisannio.it

Andrea Di Sorbo
University of Sannio
Benevento, Italy
disorbo@unisannio.it

Corrado A. Visaggio
University of Sannio
Benevento, Italy
visaggio@unisannio.it

Gerardo Canfora
University of Sannio
Benevento, Italy
canfora@unisannio.it

ABSTRACT

Blockchain technology is becoming increasingly popular, and smart contracts (i.e., programs that run on top of the blockchain) represent a crucial element of this technology. In particular, smart contracts running on Ethereum (i.e., one of the most popular blockchain platforms) are often developed with Solidity, and their deployment and execution consume gas (i.e., a fee compensating the computing resources required). Smart contract development frequently involves code reuse, but poor readable smart contracts could hinder their reuse. However, writing readable smart contracts is challenging, since practices for improving the readability could also be in contrast with optimization strategies for reducing gas consumption. This paper aims at better understanding (i) the readability aspects for which traditional software and smart contracts differ, and (ii) the specific smart contract readability features exhibiting significant relationships with gas consumption. We leverage a set of metrics that previous research has proven correlated with code readability. In particular, we first compare the values of these metrics obtained for both Solidity smart contracts and traditional software systems (written in Java). Then, we investigate the correlations occurring between these metrics and gas consumption and between each pair of metrics. The results of our study highlight that smart contracts usually exhibit lower readability than traditional software for what concerns the number of parentheses, inline comments, and blank lines used. In addition, we found some readability metrics (such as the average length of identifiers and the average number of keywords) that significantly correlate with gas consumption.

KEYWORDS

Software Engineering for Blockchain Technologies, Readability Metrics, Code Quality, Software Metrics, Empirical Study

ICPC '22, May 16–17, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9298-3/22/05…$15.00
https://doi.org/10.1145/3524610.3529157

ACM Reference Format:
Anna Vacca, Michele Fredella, Andrea Di Sorbo, Corrado A. Visaggio, and Gerardo Canfora. 2022. An Empirical Investigation on the Trade-off between Smart Contract Readability and Gas Consumption . In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524610.3529157

1 INTRODUCTION

The success of Bitcoin [25] has pushed the attention of companies and developers towards the powerful features of blockchain technology, such as immutability of transactions, decentralization, and enhanced security and privacy. Companies, organizations, and government agencies are more and more adopting this technology due to its potential to improve existing processes and enable new business models. However, while the main application of Bitcoin consists of digital currency transactions [38], Ethereum, the second most popular blockchain platform, has overcome this limitation since its appearance by providing smart contracts (i.e., Turing-complete programs that run on the blockchain). Indeed, smart contracts enable the possibility to develop decentralized applications (dApps) [22]. In this context, Ethereum became a popular platform powering the development of decentralized applications, with tens of new dApps released monthly[1].

Although the immutability of blockchain ensures the trustworthiness and fairness of transactions, it also complicates the smart contracts' development, testing, maintenance, and evolution activities [37]. Besides, the deployment and execution of Ethereum smart contracts, usually developed through the Solidity programming language, consume gas. Since smart contracts run on a decentralized infrastructure, on the Ethereum platform, the gas (in Ether) represents a fee compensating the computing resources required to make available and execute the smart contracts on the machines of miners [13]. Features like the gas system and blockchain immutability complicate the development and maintenance of smart contracts. In particular, to reduce gas consumption, smart contract developers often apply optimizations, consequently reducing code readability and understandability [42]. For any piece of code, readability is a crucial aspect, as high readability fosters desirable properties such as portability, maintainability, and reusability [21]. This is especially true in smart contracts, as recent research demonstrated

[1] https://www.stateofthedapps.com/stats

that code reuse is quite frequent in this context [10]. Thus, on the one hand, low readability significantly reduces the understandability and maintainability of the smart contracts, also hampering code reuse [7]. On the other hand, a recent survey involving 165 Ethereum developers highlighted the difficulty of writing readable smart contracts due to the developers' need to balance the readability with gas consumption [8]. As also indicated by the survey conducted by Zou et al. [43], the trade-off between code readability and code optimizations is particularly relevant in the blockchain domain, highlighting the need for *source-code-level gas-estimation and optimization tools that consider code readability.*

Stemming from these previous findings, in this paper, we aim at better understanding (i) the specific readability aspects for which traditional software and smart contracts differ, as well as (ii) the specific smart contract readability features that are related to gas consumption. To the best of the authors' knowledge, this is the first study quantitatively and qualitatively investigating (i) the commonalities and differences between the code readability of traditional software and Solidity smart contracts, and (ii) the trade-off between readability and gas consumption by directly measuring the different readability features on the smart contract source code. To carry out our investigation, we make use of software readability models proposed in the literature. In particular, several studies have proposed methods and approaches to evaluate the code readability of traditional software, like the ones evaluating Java [4, 5, 29], or Python [15, 32] programs. Recently, we proposed iSCREAM [6], a tool that automatically inspects Solidity smart contract source code and computes the set of code features that Buse and Weimer [4] demonstrated to be related to code readability. Thus, we first compare the values of such code features obtained for both a set of smart contracts and a set of traditional programs written in Java, and, then, we inspect the correlations occurring (i) between smart contract readability features and the gas required for smart contracts deployment, and (ii) between the different smart contract readability features.

The results of our study highlight that smart contracts usually exhibit lower readability than traditional software for what concerns the usage of parentheses, inline comments, and blank lines used. In addition, we found some readability metrics (such as the average length of identifiers and the average number of keywords) that significantly correlate with gas consumption. We believe that a similar investigation could help better understanding how to improve smart contract readability without impacting gas consumption too much.

The original contribution of this paper includes:

- *a comparison of readability metrics computed on both traditional software and smart contracts;*
- *an empirical study for identifying the readability metrics that most correlate with deployment costs.*

Paper structure. The paper proceeds as follows. Section 2 highlights the novelty of our findings with respect to the existing literature. Section 3 discusses the readability metrics used in this study, while Section 4 illustrates the research questions and the methodology followed to answer them. Section 5 reports and discusses the achieved results. Threats to our study's validity are commented in Section 6 and, finally, Section 7 concludes the paper outlining future research directions.

2 RELATED WORK

In this section, the related literature about code readability models and smart contract quality is discussed.

2.1 Code Readability

Due to its impact on other software quality aspects, such as portability, maintainability, and reusability [21], several studies have proposed methods and approaches to estimate the code readability of traditional software. In particular, software engineering researchers demonstrated that specific code metrics could be used as proxies to assess the readability level of a piece of code. A first model was proposed by Buse and Weimer [4, 5]. To define such a model, 100 code snippets extracted from 5 open source projects with different maturity levels were considered. The model consists of 25 code metrics related to structural aspects of code. To validate the model, 120 computer science students evaluated the readability degree of such snippets by assigning a score of 1 to 5. The study shows that the model can be effective, and better than a human on average, at predicting readability judgments. The study also demonstrates that the output of the model strongly correlates with two traditional measures of software quality: code changes and defect reports.

While Posnett et al. [29] introduced a code readability model based on only three structural features (i.e., lines of code, entropy, and Halstead's Volume metric) and demonstrated that their model outperforms the one by Buse and Weimer [4, 5], Dorn [15] proposed a code readability model relying on a larger set of features organized into four categories (i.e., visual, spatial, alignment, and linguistic). Tashtoush et al. [35] defined the Impact of Programming Features on Code Readability (IPFCR) approach. The study examines the impact of 22 code attributes on code readability and understanding. To estimate such an impact, the authors surveyed 141 experts with questions asking opinions and ratings (from 1 to 5) on the proposed attributes. The results of the study found that features like meaningful names, comments, and consistency might positively impact readability, whereas recursive functions, nested loops, and arithmetic formulas might reduce readability.

Other studies [32, 33] proposed to extend the existing source code readability models by also considering textual (or lexical) aspects in addition to structural ones. Textual aspects consider consistency in the name of identifiers (taking into account the context in which they are inserted) and well-structured comments to increase the understanding of the code. The study involved about 5k people who rated the readability of 600 snippets of code. The results show that the textual features proposed are complementary to the structural characteristics defined in previous studies. Therefore, a code readability model based on both structural and textual features achieves better accuracy than readability models relying on only structural aspects.

Fakhoury et al. [16] highlighted inconsistencies between the results of metrics related to cohesion, coupling, complexity, and readability and the interpretation of these results in practice. To improve code readability, they suggested using tools to measure structural metrics, refactoring, and style problems.

2.2 Smart Contract Quality

Smart contracts define the rules of an agreement between two or more parts. They are used to exchange money, transfer property, and anything else of value transparently and without resorting to the services of an intermediary. Since smart contracts are mainly used for money exchanges, it is important to guarantee the security of smart contracts against monetary losses. To this aim, previous research proposed security analysis approaches leveraging different techniques [31]. For example, to detect smart contract vulnerabilities, Oyente analyzes the source code of smart contracts through symbolic execution [24], while SmartInspect analyzes deployed smart contracts by leveraging decompilation techniques [3]. Other tools can identify smart contract issues not only related to security. For instance, SmartCheck [36] is able to detect Solidity code issues related to the Security, Functional, Operational, and Development categories, while MAIAN detects greedy, prodigal, and suicidal smart contract behaviors by analyzing the different invocations of a contract during its lifetime [26].

In Ethereum, the gas is a measure of the computational effort required to perform operations (such as transactions or executions of instructions) for a smart contract [1]. The Ethereum project yellow paper [40] details the cost of each individual instruction. Clearly, some instructions are more expensive than others. For gas consumption optimization, researchers have proposed several tools. These tools usually analyze the smart contract code and highlight the most expensive portions, recommending, in some cases, possible optimizations. Among these tools, Gasol [1] provides information about the upper bound of the cost of specific functions. Gasol is able to detect under-optimized storage patterns and to apply automated optimizations for the selected functions. Similarly, GasTap [2] automatically infers gas upper-bounds for a smart contract's public functions. Instead, GasChecker [9] analyzes the source code and automatically identifies inefficient portions that can increase gas consumption. SCRepair [41] is an automated tool to repair smart contracts. Beyond detecting the vulnerabilities present in smart contracts, it also tries to optimize gas consumption.

Another aspect related to smart contract quality is represented by the assessment of transactions and smart contracts execution performance. In this context, for private blockchains, it is possible to use the BLOCKBENCH framework, which allows comparing different blockchain platforms and design choices in terms of throughput, latency, scalability, and fault-tolerance [14].

Researchers also conducted interviews and surveys with smart contract developers to understand the most important challenges related to the development and maintenance of smart contracts. Among the different challenges, Zou et al. [43] report the lack of mature tools to measure smart contract quality and guarantee the security of smart contracts. Besides, the results of the investigation highlighted an important trade-off between code readability and code optimizations, as code optimizations are often applied in the blockchain domain to reduce resource (or gas) consumption. Blockchain immutability also complicates the maintenance of smart contracts [37]. For this reason, it is important to ensure that a smart contract is bug-free and well-designed before its deployment [7]. Chen et al. [10] surveyed smart contract developers to identify the most relevant maintenance problems that may exist after the deployment of smart contracts and the approaches used to address these problems. Also in this survey, it emerges the need to improve code readability of smart contracts.

While previous studies have theoretically highlighted the need for guidelines to improve code readability of smart contracts, to the best of the authors' knowledge, this is the first study quantitatively and qualitatively investigating (i) the commonalities and differences between the code readability of traditional software and Solidity smart contracts, and (ii) the trade-off between readability and gas consumption by directly measuring the different readability features on the smart contract source code. To carry out our investigation, we leveraged the iSCREAM (suite for Smart Contract REAdability assessMent) tool [6] which is able to parse smart contracts and automatically compute a set of readability metrics for each function in it contained.

3 METRICS & TOOLS

Although in the software engineering literature different models for software readability have been proposed [29, 32], we rely on the model proposed by Buse and Weimer [5]. We chose this model, as it contains metrics that can be measured by statically analyzing the code. Moreover, such metrics are intuitive and simple, and they consider factors related to structure, density, logical complexity, and documentation of source code. Specifically, the model takes into account typical aspects of programming, such as commas, periods, keywords, identifiers, parentheses, and numbers, but also more specific aspects, such as loops, conditionals, and comparisons. In addition, they are independent on the size of the source code analyzed. Table 1 reports the set of metrics considered by the model. Each metric represents either an average value per line or a maximum value for all lines. In particular, average values are obtained by counting the total occurrences of a specific language element (e.g., period, keyword, or blank line) in the considered code portion divided by its total number of lines. Instead, the count of the maximum values rely on the occurrences of the specific language element in each line. The symbols *N* (i.e., negative) or *P* (i.e., positive) appearing in the last column of Table 1 (i.e., the *Corr* column) indicate the direction of the correlation between the metric reported on the row and code readability, as reported in previous work [5]. To compute the code metrics detailed in Table 1 on the different software artifacts, we used the tool proposed by Buse and Weimer [5] in the case of traditional software, while the iSCREAM tool [6] was used in the case of smart contracts.

4 STUDY DESIGN

The *goal* of our study is to better understand the specific aspects that could reduce smart contract readability, with the *purpose* of helping developers write more readable and understandable smart contracts without affecting gas consumption too much. To this aim, we explore (i) the differences in terms of readability between smart contracts and traditional software and (ii) the smart contract readability aspects that correlate more with gas consumption.

In this section, we discuss the design of our study, including the research questions, the data collection and selection processes, and the analysis methodology we followed to answer our research questions.

Table 1: The list of metrics used. The symbols appearing in the last column indicate whether the correlation between the metric on the row and code readability is negative (*N*) or positive (*P*), as reported in previous work [5].

ID	Metric	Avg	Max	Corr
1	assignments	✓		N
2	blank.lines	✓		P
3	commas	✓		N
4	comments	✓		P
5	comparisons	✓		N
6	conditionals	✓		N
7	identifier.length	✓	✓	N
8	indentation.length	✓	✓	N
9	keywords	✓	✓	N
10	line.length	✓	✓	N
11	loops	✓		N
12	number.of.identifiers	✓	✓	N
13	numbers	✓	✓	N
14	operators	✓		P
15	parenthesis	✓		N
16	periods	✓		N
17	spaces	✓		N
18	occurrences.single.character		✓	N
19	occurrences.single.identifier		✓	N

4.1 Research Questions

Based on the aforementioned goal, in this study, we aim to answer the following research questions:

- ***RQ$_1$: What are the main differences between traditional software and smart contracts in terms of code readability?*** The first research question aims at investigating the specific readability aspects for which smart contracts and traditional software differ. Making smart contract readable is a challenge, as best practices to obtain high readability could be often in contrast with strategies to reduce gas consumption [8]. Since the development of traditional software (e.g., Java applications) is more marginally affected by the resource consumption problem, we compare the values obtained on both smart contracts and traditional software of specific code metrics that previous research [4, 5] identified as reliable proxies for estimating code readability.
- ***RQ$_2$: How do readability metrics correlate with the gas consumed for smart contract deployment?*** The second research question aims at investigating the smart contract readability aspects that most correlate with gas consumption. Previous literature [43] highlighted the need for guidelines and tools to improve smart contract readability while minimizing the impact on gas consumption. In this research question, we attempt to start filling this gap by investigating the relationships existing (i) between the specific smart contract readability metrics and gas consumption and (ii) between each pair of readability metrics, with the purpose of identifying the readability features that deserve attention during smart contract development.

Replication package. The results and dataset of smart contracts have been arranged as replication package available into a Github repository[2].

4.2 Context selection and data extraction

To answer RQ$_1$, we needed to collect real-world code snippets extracted from both traditional software and smart contracts. For traditional software, we decided to leverage the code snippets data previously used by Buse and Weimer [5]. In particular, we considered all the 100 Java code snippets encompassed in such a dataset. These snippets are short, logically consistent, and do not span multiple method bodies. More specifically, each snippet ranges between 4 and 13 rows (with an average of 7.69 rows) and includes simple statements (e.g., field declarations, assignments, function calls, breaks, continues, throws, and returns) and lines that are not simple statements, such as comments, function headers, blank lines, or headers of compound statements (e.g. `if-else`, `try-catch`, `while`, `switch`, and `for`). These snippets were selected from 5 open source projects (i.e., Hibernate, jFreeChart, FreeCol, jEdit, jUnit) with different application domains and maturity levels.

For what concerns the selection of smart contract code snippets, we leveraged a dataset containing 2,186 real-world Solidity files. Such a dataset has been used in previous work [6]. While the source code of the vast majority of smart contracts actually deployed on Ethereum is not publicly available [27], the Solidity files in the dataset were randomly selected from the set of smart contracts accessible through Etherscan[3], a popular service for exploring the Ethereum blockchain that offers the possibility to publish the source code of deployed smart contracts. All the considered smart contracts are validated and distributed on the Ethereum blockchain platform and their source code is available on Etherscan. The 2,186 Solidity files encompassed in such a dataset, contain a total of 24,663 functions. To collect smart contract snippets having similar characteristics to the ones extracted from traditional software, from the collection of 24,663 smart contract functions, we selected 100 smart contract functions having a number of rows between 4 and 13 (7.69 rows, on average) and matching the selection criteria used in previous work [5] to extract code snippets from traditional software.

To answer RQ$_2$, we considered all the 2,783 smart contracts encompassed in the 2,186 real-word Solidity files in our dataset (note that each Solidity file may contain more than one contract).

4.3 Analysis method

To answer RQ$_1$, the 100 snippets of traditional software and the 100 smart contract functions described in Section 4.1 were considered. In particular, we first computed the readability metrics (reported in Table 1) for each of the 100 traditional software snippets and the 100 Solidity functions selected. For each readability metric, we thus obtained two distributions: a first distribution of the specific readability metric values computed on traditional software snippets, and the other distribution encompassing the values of the same readability metric obtained on smart contract functions (e.g.,

[2]https://github.com/papersubmition/ReadabilitySC
[3]https://etherscan.io/

Avg.keywords in traditional software and *Avg.keywords* in smart contracts). We then compared the two distributions obtained for each metric through statistical tools. Specifically, to establish whether the differences observed in the two distributions are statistically significant, we used the non-parametric Mann-Whitney test (with p-value fixed to 0.05). We used a non-parametric statistical test as the variables of interest are not well-modeled by normal distributions (as verified through the Shapiro-Wilk test [34]). In addition, to estimate the magnitude of the statistically significant differences, we used the Cliff's *d* effect size, a measure of how often the values in one distribution are larger than the values in a second distribution. Following the guidelines in [30], we interpret the effect size as: *negligible* for $|d| < 0.147$, *small* for $0.147 \leq |d| < 0.33$, *medium* for $0.33 \leq |d| < 0.474$, and *large* for $|d| \geq 0.474$.

To answer RQ_2, for each of the 2,783 smart contracts in our dataset, we considered the readability metric values obtained for the specific smart contract and the corresponding gas consumed for deploying the smart contract. More specifically, while in the previous analysis, the readability metrics were computed at the function level, in this case we slightly modified the iSCREAM tool [6] to make it traverse the whole smart contract and compute the readability metrics (reported in Table 1) at the smart contract level. For example, to obtain the average number of keywords, the modified version of the tool counts the total occurrences of Solidity keywords appearing in the whole smart contract (and not only in a particular function) and divides it by the total number of lines of code related to the smart contract. Similarly, for obtaining the max number of keywords appearing in a single line, the modified version of the tool considers all the smart contract's lines (and not only the ones related to a particular function). Besides, to collect data on gas consumed during smart contract deployment, we actually deployed each smart contract in our dataset on a private Ethereum blockchain by leveraging the Truffle Suite [18]. Through the Ethereum client Ganache [17], we were able to obtain information about the actual gas consumed for the deployment of each smart contract in the dataset. To evaluate whether significant relationships between (i) each readability metric and gas consumption and (ii) each pair of readability metrics can be found, we used the Spearman rank correlation coefficient [12], fixing the p-value ≤ 0.05. To deal with multiple comparisons, we adopted Holm's p-value correction procedure [19]. As recommended by Cohen's standard [11], we interpret the strength of the correlation as (i) *small* for $0.10 \leq |\rho| \leq 0.29$, (ii) *medium* for $0.30 \leq |\rho| \leq 0.49$, and (iii) *large* for $|\rho| \geq 0.50$.

5 RESULTS

In this section, we discuss the main results achieved in our study to answer the research questions posed.

5.1 *RQ_1: What are the main differences between traditional software and smart contracts in terms of code readability?*

To gain empirical evidence of the differences in readability metrics between traditional software and smart contracts, we tested the following null hypothesis:

H_0: traditional software exhibits a value of m_i equal to the one exhibited by smart contracts

Table 2: Results of Mann-Whitney test and Cliff's d effect size (100 code snippets from traditional software - 100 code snippets from smart contracts).

Metric	p-value	cliff-delta
Avg.assignments	2.09e-07	0.4229 / medium
Avg.blank.lines	1.06e-12	0.5029 / large
Avg.commas	0.3921925	—
Avg.comments	0.08170996	—
Avg.comparisons	0.007548249	-0.2018 / small
Avg.identifiers.length	1.12e-06	0.3986 / medium
Avg.conditionals	2.49e-08	0.4012 / medium
Avg.indentation.length	0.02472583	0.1838 / small
Avg.keywords	1.87e-22	0.7974 / large
Avg.line.length	0.04342982	-0.1654 / small
Avg.loops	0.000328727	0.197 / small
Avg.number.of.identifiers	2.46e-31	0.953 / large
Avg.numbers	0.01830024	-0.1705 / small
Avg.operators	0.001564026	0.2383 / small
Avg.parenthesis	4.19e-09	-0.4808 / large
Avg.periods	0.000207089	0.3033 / small
Avg.spaces	0.07465014	—
Max.identifier.length	5.36e-14	0.6147 / large
Max.indentation.length	4.62e-09	0.4633 / medium
Max.keywords	8.21e-07	0.3705 / medium
Max.line.length	0.3639308	—
Max.number.of.identifiers	0.4866922	—
Max.numbers	0.000902801	-0.239 / small
Max.occurrences.single.character	0.004613365	0.2318 / small
Max.occurrences.single.identifier	9.87e-07	0.3911 / medium

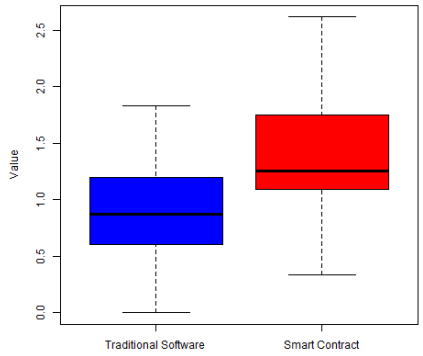

Figure 1: Boxplots of Avg.parenthesis

H_0 has been tested through Mann-Whitney test (fixing *p-value* to 0.05) for all the metrics m_i defined in Table 1. Table 2 reports the test results, while Figures 1, 2, 3, 4, 5, 6, 7, 8, and 9 illustrate, through box plots, the main differences observed between traditional software and smart contracts. In particular, Table 2 reports the p-values obtained for the 25 metrics considered and the corresponding Cliff's *d* results for which the Mann-Whitney test returned $p \leq 0.05$.

For the sake of space, in the following, we mainly look at the statistically significant differences for which we obtained *large* and *medium* effect size. In particular, as shown in Table 2, for 5 metrics we observed statistically significant differences with *large* effect size, while for 6 metrics the differences have *medium* effect size.

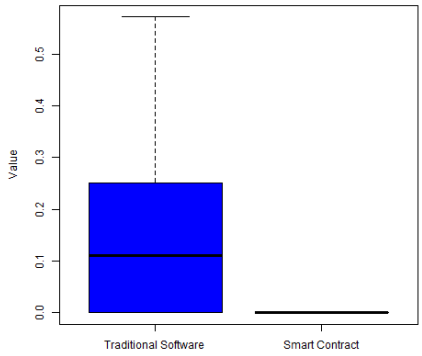

Figure 2: Boxplots of Avg.blank.lines

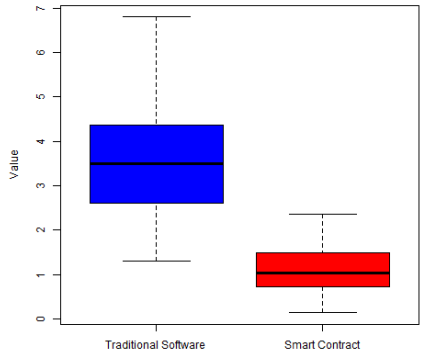

Figure 4: Boxplots of Avg.number.of.identifiers

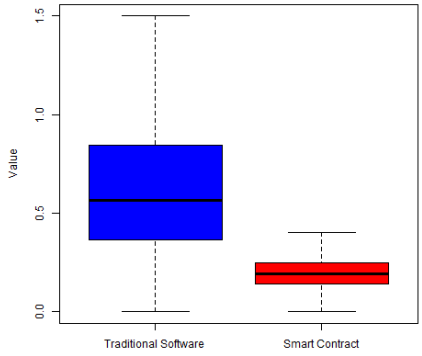

Figure 3: Boxplots of Avg.keywords

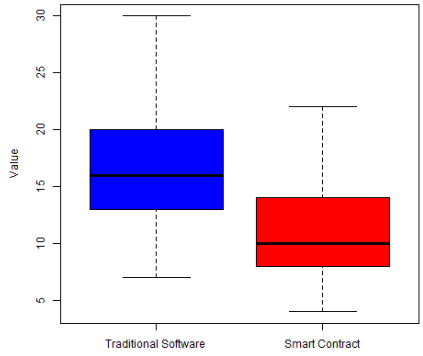

Figure 5: Boxplots of Max.identifier.length

Concerning the differences for which a *large* effect size was observed, on the one hand, it emerged that smart contracts exhibit higher values of average parentheses than traditional software (see Figure 1). This metric (i.e., average parenthesis) has a relevant predictive power when used for estimating code readability [5]. In particular, during smart contract development, the intensive use of parentheses might deteriorate readability significantly. In traditional software, such as in Java programs [39], using blank lines improves readability as it binds logically related sections of code. Considering Figure 2, we observe that this convention is not frequently used during smart contract development, if compared to traditional software. Therefore, as also demonstrated by Buse & Weimer [5], the usage of blank lines might improve smart contract readability.

On the other hand, we noticed that smart contracts show lower values of average keywords (see Figure 3) and identifiers (see Figure 4) compared to traditional software. Although Java and Solidity languages have many possible keywords that can be used in code writing, in the case of smart contracts they are not largely used. This can be due to the lack of a complete documentation for smart contract development [8], and the lower maturity of Solidity [20] with respect to Java. Besides, since identifiers usually give names to memory locations, the lower usage of identifiers in smart contracts compared to traditional software could be due to the fact that operations involving memory or storage are gas-expensive [1]. Previous findings [5] showed that both these metrics might reduce the readability, and a lesser usage of identifiers and keywords could lead to improve the readability. Finally, for smart contracts, we report lower values of the maximum identifiers length (see Figure 5) in a single line than traditional software. Since this metric is negatively correlated with readability, lower values of this metric correspond to higher readability [5].

Concerning the statistically significant differences with *medium* effect sizes, our results showed that Solidity smart contracts exhibit lower values of average identifier length (see Figure 6) than traditional software. In addition, smart contracts also exhibit lower values of average assignments and average conditionals. While average identifiers length shows low relevance in estimating the

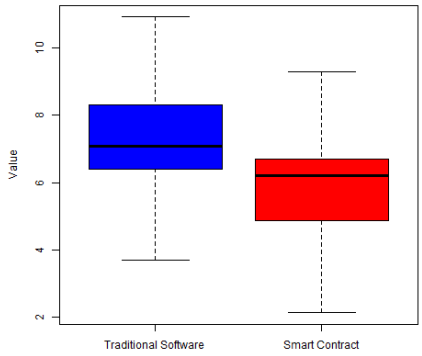

Figure 6: Boxplots of Avg.identifier.length

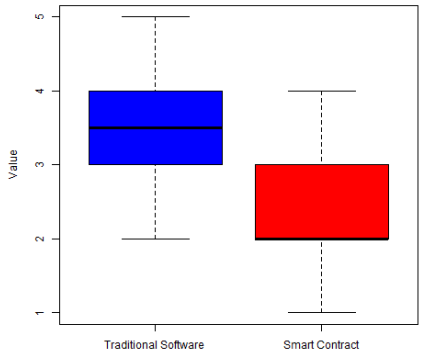

Figure 7: Boxplots of Max.occurrences.single.identifier

code readability, lower values of average assignments and average conditionals might lead to higher readability [5]. Furthermore, we observed that smart contracts show lower values of the maximum indentation and maximum keywords than traditional software. While maximum indentation would affect readability more, the *Max.keywords* metric exhibits a lower relationship with readability [5]. Smart contracts also show lower values than traditional software for the *Max.occurrences.single.identifier* metric (see Figure 7), likely leading to higher readability.

We also noticed that traditional software is more frequently commented (see Figure 8), and more attention is paid to proper indentation (see Figure 9) compared to smart contracts, likely due to the use of more mature tools and IDEs to write traditional programs. Interestingly, concerning these aspects, the differences between smart contracts and traditional software are *small* (*Avg.indentation.length*) or not statistically significant (*Avg.comments*).

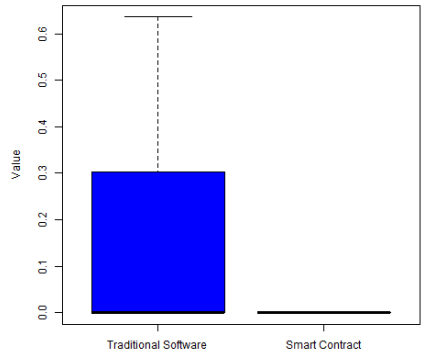

Figure 8: Boxplots of Avg.comments

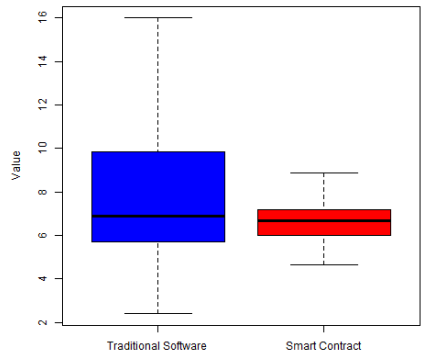

Figure 9: Boxplots of Avg.indentation.length

RQ_1 Summary: *The rare usage of blank lines, the intensive use of parentheses, and the absence of inline comments reduce smart contract readability, compared to traditional software. However, we also notice that smart contracts make use of lower average numbers of keywords, identifiers, assignments, and conditionals than traditional software, likely favoring code readability.*

5.2 *RQ_2: How do readability metrics correlate with the gas consumed for smart contract deployment?*

Figure 10 reports the results of the Spearman correlation between each pair of the readability metrics and between the readability metrics and gas consumption. All the results considered in the following discussion correspond to an adjusted p-value ≤ 0.05. In Figure 10 we consider the 17 average and the 8 maximum values of metrics. These readability metrics were computed on the initial dataset of 2,186 Solidity files. As a smart contract may contain more than one contract within, we considered 2,783 contract as input.

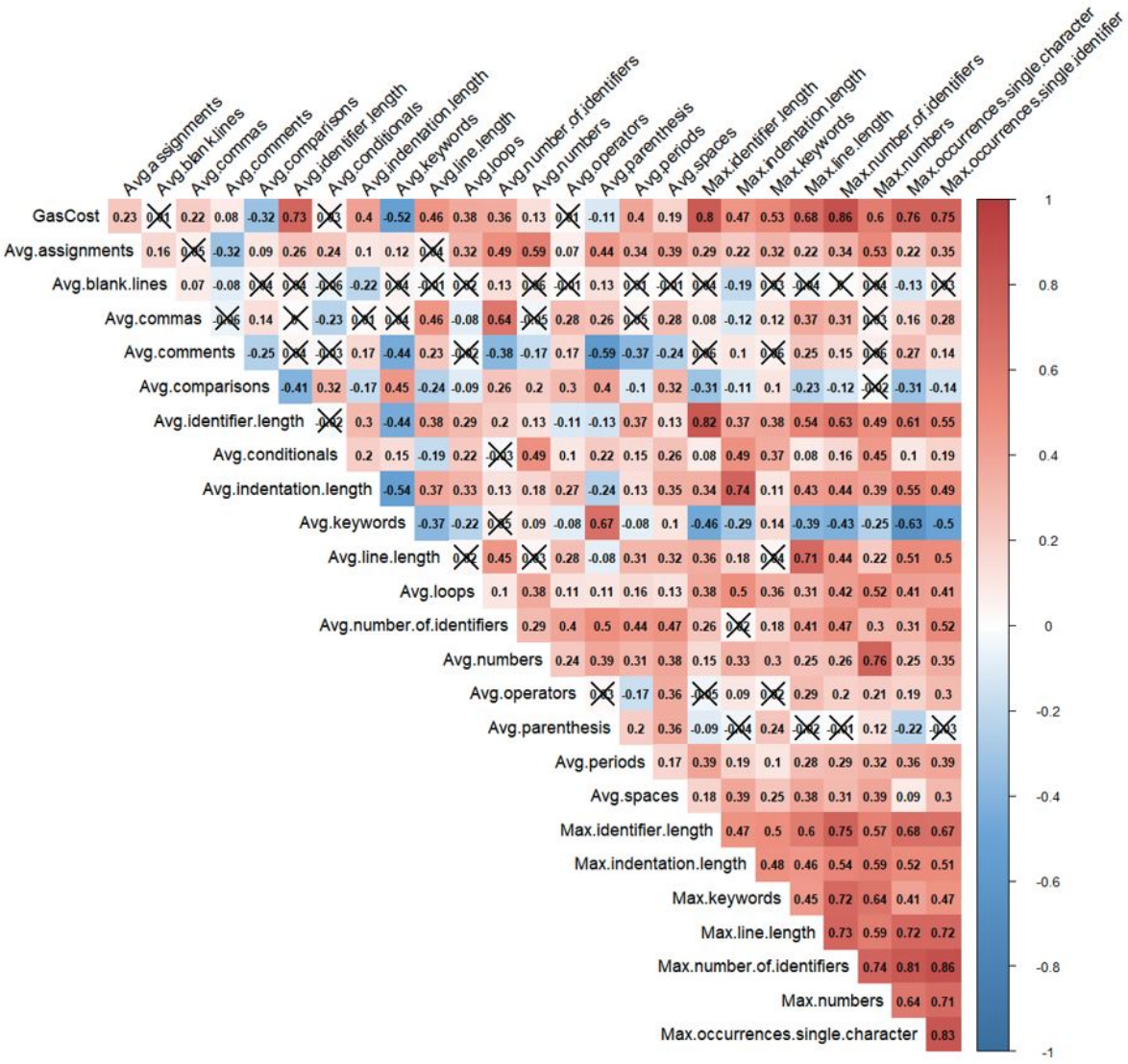

Figure 10: Spearman correlation results among the readability metric and the gas cost (X is used to indicate the correlations that are not statistically significant).

The values of the correlation indices vary between -1 and +1; both extreme values represent perfect relationships between variables, while 0 represents the absence of a relationship. A positive relationship means that higher values in a variable correspond to higher values on the second variable, too. Conversely, a negative relationship indicates that lower scores on one variable co-occur with higher scores on the other variable.

It is important to recall that we do not observe any strong correlations between gas consumption and the average numbers of comments, blank lines, and parentheses. Having a well-commented and well-structured code is essential to facilitate code understanding. Thus, to encourage code reuse and maintainability without influencing gas consumption, we first recommend smart contract developers increment the usage of (i) inline comments (to provide insights about the code) and (ii) blank lines (to separate logically-related code portions), (iii) simplifying code structures as well. Interestingly, the *Avg.parenthesis* metric exhibits *large* effect size correlations with (i) the average number of comments ($\rho = -0.59$) and (ii) the average number of keywords ($\rho = 0.67$). This indicates that code involving more complicated structures is rarely commented, while the usage of specific keywords might lead to a more intensive use of parentheses.

As shown in Figure 10, two readability metrics are more strongly linked to gas consumption: the average identifiers length (*Avg. identifier.length*, with $\rho = 0.73$) and the average keywords (*Avg.keywords*, with $\rho = -0.52$). The average number of keywords is negatively

correlated with gas consumption. Thus, an increase in the usage of keywords might lead to gas savings.

In general, (i) a poor usage of keywords (i.e., *Avg.keywords - GasCost*, $\rho = -0.52$), in conjunction with the use of (ii) longer identifiers (i.e., *Avg.identifier.length - GasCost*, $\rho = 0.73$), probably to facilitate code comprehensibility (i.e., more self-explanatory identifiers), and, consequently, (iii) longer code lines (i.e., *Avg.line.length - GasCost*, $\rho = 0.46$), (iv) deeper indentation (i.e., *Avg.indentation.length - GasCost*, $\rho = 0.40$), and (v) more intensive use of *dot notation* (i.e., *Avg.periods - GasCost*, $\rho = 0.40$), probably for referencing complex data structures (such as structs), might lead to higher gas consumption. This result is corroborated by the fact that we observe an anti-correlation with *medium* effect size between the average number of keywords and the average length of identifiers (i.e., *Avg.keywords - Avg.identifier.length*, $\rho = -0.44$). Similarly, we notice correlations with *medium* effect size between (i) the average length of the identifiers and the average length of the lines of code (i.e., *Avg.identifier.length - Avg.line.length*, $\rho = 0.38$), (ii) the depth of indentation and the average length of lines of code (i.e., *Avg.indentation.length - Avg.line.length*, $\rho = 0.37$), and (iii) the average number of *periods* and the average length of identifiers (i.e., *Avg.periods - Avg.identifier.length*, $\rho = 0.37$). It is important to note that longer lines of code, as well as more intensive use of dot notation and deeper indentation might have a significant negative impact on readability, as reported in [5].

Clearly, gas consumption grows as the average numbers of identifiers (i.e., *Avg.number.of.identifiers - GasCost*, $\rho = 0.36$) and loops used (i.e., *Avg.loops - GasCost*, $\rho = 0.38$) increase. Indeed, repeated operations and those involving storage are among the most expensive in terms of gas consumption. In particular, a larger average number of identifiers can have a negative impact on readability, as demonstrated by Buse & Weimer [5]. We also report (i) a correlation with *medium* effect size between the average number of identifiers and the average number of *periods* (i.e., *Avg.number.of.identifiers - Avg.periods*, $\rho = 0.44$), likely symptom of intensive use of complex data structures (e.g., structs), and a correlation with *medium* effect size between the average number of loops and the average indentation depth (i.e., *Avg.loops - Avg.indentation.length*, $\rho = 0.33$). In addition, we highlight the correlation with *medium* effect size between the average number of loops and the average number of assignments (i.e., *Avg.loops - Avg.assignments*, $\rho = 0.32$). This is a likely indication that assignment operations (that are expensive if they involve storage) are often incorporated within loops.

Correlations with *medium* effect sizes are also obtained between (i) average numbers and average loops (i.e., *Avg.numbers - Avg.loops*, $\rho = 0.38$) and between (ii) average numbers and average conditionals (i.e., *Avg.numbers - Avg.conditionals*, $\rho = 0.49$). These correlations demonstrate the frequent involvement of numbers in loops (e.g., to specify loop stopping conditions) and conditions (e.g., number checks). We also highlight correlations with *medium* and *large* effect sizes between the average commas and (i) the average line length (i.e., *Avg.commas - Avg.line.length*, $\rho = 0.46$) and (ii) the average number of identifiers (i.e., *Avg.commas - Avg.number.of.identifiers*, $\rho = 0.64$), as, when many identifiers are present on the same line, they are separated by commas (e.g., `event Transfer(address indexed from, address indexed to, uint256 value);`). Therefore, simplifying smart contract lines of code and discouraging the use of complex data structures and loops, might have positive effects not only on code readability but also on gas consumption.

Considering the metrics representing maximum values, 7 out of 8 metrics exhibit correlations with gas consumption having *large* effect sizes (i.e., $\rho \geq 0.50$). For just one metric (i.e., *Max.indentation.length*) we obtain a correlation with gas consumption having a *medium* effect size. As demonstrated in previous work [5], all these metrics also exhibit negative correlations with readability, and higher values of these metrics could lead to a lower readability. Therefore, especially for what concerns *Max.number.of.identifiers* and *Max.line.length* metrics, lowering the values associated with these indicators might not only correspond to improvements in terms of readability but also to optimizations in terms of gas consumption.

RQ_2 Summary: *In general, lower values in all the eight metrics representing maximum values correspond to higher readability and lower gas consumption. Besides, more intensive uses of keywords and comparisons co-occur with lower gas consumption but also with degradations in readability. On the contrary, reduced average length/depth of identifiers, lines, and indentation and lower usages of identifiers, loops, and dot notation co-occur with both higher readability and lower gas consumption.*

6 THREATS TO VALIDITY

Threats to construct validity concern the relation between measured properties and the theoretical concepts they should represent. To carry out our study, we considered a suite of metrics that could be insufficient to exhaustively assess the level of readability of a piece of code. For partially mitigating this issue, we leveraged a set of well-known metrics previously used for similar purposes [4, 5]. We chose to consider smart contracts at the source code level since previous work [8, 43] highlighted a trade-off between code readability and code optimizations in the blockchain domain. Thus, our work is a first attempt to deal with this issue, providing an analysis of the structural aspects of source code (that previous work has proven to correlate with code readability) co-occurring with higher (lower) costs. In particular, developers mainly deal with source code (and not opcodes) when performing software maintenance, testing, and evolution tasks. Thus, to ease these tasks, the main aim of the work is to explore ways to obtain a more readable, even if optimized, source code. However, it is important to highlight that gas consumption is computed at the bytecode level, and different Solidity compilers might lead to different bytecode versions for the same source code. To partially mitigate this threat, in our RQ_2, we used the same Solidity compiler for assessing the costs associated with each smart contract considered.

Threats to conclusion validity concern the relationship between treatment and outcome. Appropriate statistical procedures have been adopted to draw our conclusions. In particular, we used the Mann-Whitney test to investigate the statistically significant differences occurring in the values of readability metrics obtained on a set of 100 code snippets extracted from traditional software and 100 code snippets related to smart contracts. We also adopted

Cliff's delta effect size measure to assess the magnitude of the statistically significant differences. We used non-parametric statistical tests as the variables of interest are not well-modeled by normal distributions (as verified through the Shapiro-Wilk test [34]). In addition, we used the Spearman rank correlation coefficient to investigate (i) the relationships between readability metrics and gas consumption, as well as (ii) the correlations between each pair of readability metrics. As we performed multiple tests, we adjusted our p-values using Holm's correction procedure. Since some of the obtained correlations could be altogether casual, we only discussed statistically significant (p-value ≤ 0.05) correlations.

Threats to internal validity concern any confounding factors that can affect our results. For comparing the readability of traditional software with the one of smart contract functions, we leveraged a set of 100 Java code snippets inherited from previous work [5] and selected a set of 100 functions extracted from Solidity files. Some of the features that most effectively characterize readable code might be different for different programming languages [15]. However, in Solidity, a contract is similar to a Java class. Both can have constructors, public and private methods, global and local variables, and can be instantiated. Notable differences in how the code looks in these two languages are modifiers and events. Both are usually stated at the beginning of a contract. Remarkably, Buse and Weimer's model focuses on structural characteristics of code rather than specific programming features and most of the concepts behind the model apply to most common programming languages. Nevertheless, an important limitation of our study, is that the model treats Solidity-specific constructs (like the transaction-reverting statements [23]) as the other compound statements. Further research is needed to comprehend whether such kinds of statements might particularly hinder code readability.

The different application domains for which Java and Solidity are used might represent an important threat to our study's validity. While Java is a general-purpose language, Solidity is specifically used to develop smart contracts (that mainly deal with the management of financial transactions). Although one of the goals of our work is to better understand the readability aspects for which smart contracts and traditional software differ, we believe that further investigations are needed to explore whether our results also apply when considering traditional software with specific resource constraints.

Threats to external validity concern the generalization of the findings. Further research is needed to more in-depth understand the Solidity-specific coding practices that reduce code readability. In particular, more comprehensive models involving a broader set of readability indicators are necessary to catch further smart contracts' peculiarities that could hamper readability. However, our work jointly attempts to (i) highlight a first set of differences between the readability features occurring in traditional software and smart contracts and (ii) identify Solidity code practices that could negatively impact gas consumption. As regards RQ_1, the selected Solidity functions might not be representative of all the smart contract functions. However, to guarantee a fair comparison, we selected smart contract functions by using the same selection criteria adopted in previous work [5] to collect the 100 code snippets (extracted from java programs) considered in our study. Concerning RQ_2, our experiment has been carried out by considering a data collection comprising 2,783 small to medium size real-world Solidity smart contracts. These smart contracts might be not representative of all the smart contracts deployed on the Ethereum blockchain, and some of the findings may depend on the specific data we considered. For partially mitigating this threat, we chose a data collection that is (i) a statistically significant sample of the smart contracts for which the source code is available on Etherscan (i.e., a confidence level of 99% and a margin of error smaller than 3%), and (ii) sufficiently large to be representative of the smart contracts deployed on Ethereum [28]. In the future, we plan to perform experiments at a larger scale to further verify the generalizability of our findings.

7 CONCLUSION

Readability is one of the most important aspects to consider when assessing the quality of software, as it fosters desirable properties of code, such as portability, maintainability, and reusability. However, the gas system and blockchain immutability complicate the development and maintenance of smart contracts, as developers often apply optimizations aimed at minimizing the gas required for smart contracts' deployment and execution. Previous work [8, 43] found that these optimizations might negatively impact code readability.

Stemming from these previous findings, in this study, we proposed the first qualitative and quantitative investigation aimed at assessing the specific readability aspects (i) for which traditional software (that is not affected by the gas system optimizations) and smart contracts differ, and (ii) the ones that exhibit correlations with gas consumption.

Our results show that relevant differences exist between traditional software and smart contracts that might reduce smart contract readability, especially for what concerns the usage of parentheses, inline comments, and blank lines. Moreover, we found that improvements in such aspects would not significantly impact costs, as the corresponding metrics do not exhibit relevant correlations with gas consumption.

Although correlation does not necessarily imply causation, our work mainly aims at first looking at the patterns and relations existing between source code readability aspects and gas consumption to better comprehend the source code metrics to focus on for further analyses. We believe that our work has the potential to stimulate further research in this direction, to finally address the specific challenge (highlighted in previous surveys involving blockchain developers [8, 43]) of developing reliable source-code-level gas optimization tools considering code readability. Besides, our work could be useful to drive improvements in the Solidity programming language toward obtaining more readable/understandable smart contracts (e.g., reducing the number of parentheses required for complex compound statements or forcing indentation and blank lines). As future work, we plan to extend our analysis by considering further readability-related aspects (not only related to structural aspects of code) and languages for smart contract development. In particular, based on our results, we aim at developing tools for automatically estimating the code readability of smart contracts that could also recommend optimizations in low readable code portions. Finally, we also plan to involve smart contract developers to understand the perceived importance of the various readability aspects in the blockchain domain.

REFERENCES

[1] Elvira Albert, Jesús Correas, Pablo Gordillo, Guillermo Román-Díez, and Albert Rubio. 2020. Gasol: Gas analysis and optimization for ethereum smart contracts. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Cham, 118–125.

[2] Elvira Albert, Jesús Correas, Pablo Gordillo, Guillermo Román-Díez, and Albert Rubio. 2021. Don't run on fumes—Parametric gas bounds for smart contracts. *Journal of Systems and Software* 176 (2021), 110923.

[3] Santiago Bragagnolo, Henrique Rocha, Marcus Denker, and Stéphane Ducasse. 2018. SmartInspect: solidity smart contract inspector. In *2018 International workshop on blockchain oriented software engineering (IWBOSE)*. IEEE, 9–18.

[4] Raymond PL Buse and Weimer Westley R. 2009. Learning a metric for code readability. *IEEE Transactions on Software Engineering* 36, 4 (2009), 546–558.

[5] Raymond PL Buse and Westley R Weimer. 2008. A metric for software readability. In *Proceedings of the 2008 international symposium on Software testing and analysis*. 121–130.

[6] Gerardo Canfora, Andrea Di Sorbo, Michele Fredella, Anna Vacca, and Corrado A. Visaggio. 2021. iSCREAM: a suite for Smart Contract REAdability assessMent. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 579–583.

[7] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen. 2020. Defining smart contract defects on ethereum. *IEEE Transactions on Software Engineering* (2020).

[8] Jiachi Chen, Xin Xia, David Lo, John Grundy, and Xiaohu Yang. 2021. Maintenance-related concerns for post-deployed Ethereum smart contract development: issues, techniques, and future challenges. *Empirical Software Engineering* 26, 6 (2021), 1–44.

[9] Ting Chen, Youzheng Feng, Zihao Li, Hao Zhou, Xiapu Luo, Xiaoqi Li, Xiuzhuo Xiao, Jiachi Chen, and Xiaosong Zhang. 2020. Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts. *IEEE Transactions on Emerging Topics in Computing* (2020).

[10] Xiangping Chen, Peiyong Liao, Yixin Zhang, Yuan Huang, and Zibin Zheng. 2021. Understanding Code Reuse in Smart Contracts. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 470–479.

[11] Jacob Cohen. 1988. Statistical power analysis for the behavioral sciences second edition. *Lawrence Erlbaum Associates, Publishers* (1988).

[12] Wayne W Daniel. 1990. Spearman rank correlation coefficient. *Applied nonparametric statistics, 2nd ed. PWS-Kent, Boston* (1990), 358–365.

[13] Andrea Di Sorbo, Sonia Laudanna, Anna Vacca, Corrado A. Visaggio, and Gerardo Canfora. 2022. Profiling gas consumption in solidity smart contracts. *Journal of Systems and Software* 186 (2022), 111193.

[14] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1085–1100.

[15] Jonathan Dorn. 2012. A general software readability model. *MCS Thesis available from (http://www. cs. virginia. edu/weimer/students/dorn-mcs-paper. pdf)* (2012).

[16] Sarah Fakhoury, Devjeet Roy, Adnan Hassan, and Vernera Arnaoudova. 2019. Improving source code readability: theory and practice. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2–12.

[17] Truffle Blockchain Group. 2021. ganache. https://www.trufflesuite.com/ganache. Online; accessed November 22, 2021.

[18] Truffle Blockchain Group. 2021. truffle. https://www.trufflesuite.com/. Online; accessed November 22, 2021.

[19] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[20] Henry M Kim and Marek Laskowski. 2018. Toward an ontology-driven blockchain design for supply-chain provenance. *Intelligent Systems in Accounting, Finance and Management* 25, 1 (2018), 18–27.

[21] John C Knight and E Ann Myers. 1991. Phased inspections and their implementation. *ACM SIGSOFT Software Engineering Notes* 16, 3 (1991), 29–35.

[22] Aija Leiponen, Llewellyn DW Thomas, and Qian Wang. 2021. The dApp economy: a new platform for distributed innovation? *Innovation* (2021), 1–19.

[23] Lu Liu, Lili Wei, Wuqi Zhang, Ming Wen, Yepang Liu, and Shing-Chi Cheung. 2021. Characterizing Transaction-Reverting Statements in Ethereum Smart Contracts. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 630–641.

[24] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 254–269.

[25] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf Accessed: 2015-07-01.

[26] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. 2018. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 653–663.

[27] Gustavo Ansaldi Oliva, Ahmed E. Hassan, and Zhen Ming (Jack) Jiang. 2020. An exploratory study of smart contracts in the Ethereum blockchain platform. *Empir. Softw. Eng.* 25, 3 (2020), 1864–1904.

[28] Giuseppe Antonio Pierro, Roberto Tonelli, and Michele Marchesi. 2020. An Organized Repository of Ethereum Smart Contracts' Source Codes and Metrics. *Future Internet* 12, 11 (2020), 197.

[29] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A simpler model of software readability. In *Proceedings of the 8th working conference on mining software repositories*. 73–82.

[30] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys. In *Annual meeting of the Florida Association of Institutional Research*, Vol. 13.

[31] Sara Rouhani and Ralph Deters. 2019. Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access* 7 (2019), 50759–50779.

[32] Simone Scalabrino, Mario Linares-Vásquez, Rocco Oliveto, and Denys Poshyvanyk. 2018. A comprehensive model for code readability. *Journal of Software: Evolution and Process* 30, 6 (2018), e1958.

[33] Simone Scalabrino, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. 2016. Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 1–10.

[34] S. S. Shapiro and M. B. Wilk. 1965. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika* 52, 3/4 (1965), 591–611.

[35] Yahya Tashtoush, Zeinab Odat, Maryan Yatim, and Izzat Alsmadi. [n. d.]. Impact of Programming Features on Code Readability. *International Journal of Software Engineering and Its Applications* 7, 6 ([n. d.]), 441–458.

[36] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 9–16.

[37] Anna Vacca, Andrea Di Sorbo, Corrado Aaron Visaggio, and Gerardo Canfora. 2021. A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. *J. Syst. Softw.* 174 (2021), 110891.

[38] Shuai Wang, Yong Yuan, Xiao Wang, Juanjuan Li, Rui Qin, and Fei-Yue Wang. 2018. An Overview of Smart Contract: Architecture, Applications, and Future Trends. In *2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018*. 108–113.

[39] Xiaoran Wang, Lori Pollock, and K. Vijay-Shanker. 2011. Automatic Segmentation of Method Code into Meaningful Blocks to Improve Readability. In *2011 18th Working Conference on Reverse Engineering*. 35–44. https://doi.org/10.1109/WCRE.2011.15

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

[41] Xiao Liang Yu, Omar Al-Bataineh, David Lo, and Abhik Roychoudhury. 2020. Smart contract repair. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–32.

[42] Abdullah A Zarir, Gustavo A Oliva, Zhen M Jiang, and Ahmed E Hassan. 2021. Developing Cost-Effective Blockchain-Powered Applications: A Case Study of the Gas Usage of Smart Contract Transactions in the Ethereum Blockchain Platform. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–38.

[43] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach D Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. 2019. Smart Contract Development: Challenges and Opportunities. *IEEE Transactions on Software Engineering* (2019).