Inheritance software metrics on smart contracts

Ashish Rajendra Sai

17053145@studentmail.ul.ie

Lero, University of Limerick, Ireland

Jim Buckley

Lero, University of Limerick, Ireland

Conor Holmes

CSIS

University of Limerick, Ireland

Andre Le Gear

Horizon Globex Ireland DAC

Limerick, Ireland

ABSTRACT

Blockchain systems have gained substantial traction recently, partly due to the potential of decentralized immutable mediation of economic activities. Ethereum is a prominent example that has the provision for executing stateful computing scripts known as Smart Contracts. These smart contracts resemble traditional programs, but with immutability being the core differentiating factor. Given their immutability and potential high monetary value, it becomes imperative to develop high-quality smart contracts. Software metrics have traditionally been an essential tool in determining programming quality. Given the similarity between smart contracts (written in Solidity for Ethereum) and object-oriented (OO) programming, OO metrics would appear applicable. In this paper, we empirically evaluate inheritance-based metrics as applied to smart contracts. We adopt this focus because, traditionally, inheritance has been linked to a more complex codebase which we posit is not the case with Solidity based smart contracts. In this work, we evaluate the hypothesis that, due to the differences in the context of smart contracts and OO programs, it may not be appropriate to use the same interpretation of inheritance based metrics for assessment.

CCS CONCEPTS

• **Software and its engineering** → **Software reliability**; **Software design techniques**; **Inheritance**.

KEYWORDS

smart contracts, software metrics, inheritance, complexity

ACM Reference Format:

Ashish Rajendra Sai, Conor Holmes, Jim Buckley, and Andre Le Gear. 2020. Inheritance software metrics on smart contracts. In *28th International Conference on Program Comprehension (ICPC '20)*, October 5–6, 2020, Seoul, Republic of Korea. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3387904.3389284>

1 INTRODUCTION

Blockchain is an immutable database shared by the participants of a distributed network [35]. The main application of blockchain, thus far, has been cryptocurrencies such as Bitcoin and Ethereum. These blockchain-based cryptocurrencies are estimated to have a market capitalization of over 200 Million USD [36].

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

ICPC '20, October 5–6, 2020, Seoul, Republic of Korea

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-7958-8/20/05...\$15.00

<https://doi.org/10.1145/3387904.3389284>

Some blockchains have the provision for executing user programs in a decentralized, stateful fashion. These are known as smart contracts [9]. Smart contracts were first implemented in Bitcoin, but were limited in functionality due to the lack of Turing completeness of the programming environment [37]. This limitation was overcome with the introduction of the Turing complete execution environment in the Ethereum network. Ethereum smart contracts can be written in a high-level OO programming language known as Solidity [29]. Solidity has a syntax that resembles that of C++, Java, C#, and JavaScript.

Given the monetary value associated with smart contract transactions, quality assessment and evaluation [25] becomes vital but, even in traditional software systems, such quality assessment techniques can be quite informal [4]. Although some argue for instant feedback regarding the quality control [8], another approach is to apply well-established software metrics. Software metrics have conventionally been used to determine the quality, maintainability, and testability of programs [16], and many programming-paradigm specific software metrics have been proposed [12, 33, 34]. As Solidity adheres to the OO programming paradigm, the metrics proposed by [12] have been identified as potentially useful [22, 44]. These metrics may help the developers produce improved quality contracts that are easier to test and maintain.

Despite the syntactical and conceptual similarities between Solidity and conventional OO programming languages, they vary significantly. The immutability imposed on smart contracts by the target blockchain platform is the prime differentiating factor. Another significant difference is the cost of execution, where each byte code-level operation in Ethereum carries a dynamically calculated fee influenced by market forces within the network. Other differences include the functional limitations of Solidity. For example, the inability to return an array from a function [2, 38]. Developers have found several workarounds to these limitations. For example, where inheritance is favoured over composition [18].

These differences may warrant a recalibration of existing OO metrics when applied to smart contracts. In this study, we aim to evaluate two such OO metrics empirically. Given the assertion in [18], that Solidity may favor inheritance over composition, we restrict our focus to the inheritance-based metrics proposed by [12]. This early research aims to provide preliminary evidence towards answering the following research question:

Does moving from traditional OO programming contexts to Solidity contracts impact inheritance-based practice, as measured by inheritance-based metrics?

In conventional software systems, a high degree of inheritance is often linked with greater complexity, resulting in more faults [11].

It has been argued that misuse of such functionality should also be investigated [5]. However, as functional and economic constraints limit the complexity of smart contracts, the codebase remains relatively simple. We anticipate that this relative simplicity of code might mitigate the potential complexity introduced by inheritance and may result in greater use. Additionally, as inheritance favours code reuse, we believe that developers will *favour inheritance more in smart contracts than in traditional OO programs*. We address this hypothesis through an empirical analysis.

The paper makes the following contributions:

- We present an illustrative case study, as a motivating example, suggesting increased use of inheritance in Solidity smart contracts.
- We present a small empirical study that captures inheritance-metric data for prominent smart contracts on Ethereum.
- We present a comparison between inheritance in smart contracts and traditional systems based on our provisional findings (Section 5).

2 SOFTWARE METRICS AND SMART CONTRACTS

Software metrics are commonly used for cost estimation, quality assessment, reliability testing, security evaluation, and complexity measurement [16]. Research suggests that such metrics should be employed in the evaluation of smart contracts owing to the financial value of the transactions they mediate [22, 44]. The paper by [30] analyzed 10000 smart contracts and reported that the software code metrics such as line of code had lower values on average with high variances when compared with standard software. However, the research in specific metrics for smart contracts is in the early stages [15, 28, 31, 44, 45]. Due to the preliminary nature of our work in this area, we restrict our focus to the inheritance-based metrics proposed by [12].

In OO programming, inheritance is a technique by which one class or object can reuse code from another class or object [24]. Due to the significance of inheritance in OO programming, its implication on cost, quality, and maintainability have been thoroughly examined [13, 32, 39, 41, 46]. A high degree of inheritance is often linked with greater complexity [11], which translates to more faults and higher costs. This adverse impact has led to the paradigm of *"Composition over Inheritance"* [20], but it is still widely used [26]. Thus, it becomes crucial to capture and measure different aspects of inheritance to work towards minimizing adverse effects. In this study, we restrict our investigation to the inheritance metrics proposed by [12]:

- (1) **Depth of Inheritance Tree (DIT)** calculates the maximum length of a path from a class to the root class. [12] suggests that *"if the depth of Inheritance tree has more than five steps, the code is too complex"*.
- (2) **Number of Children (NOC)** indicates the number of immediate subclasses that depend on a class in the class structure. A large number of children classes are considered favorable due to greater code reuse.

3 SMART CONTRACTS AND INHERITANCE

Smart contracts were created in the early 1990s [7, 42] to specify an enforceable agreement for a transaction. Smart contracts are programs that can be correctly executed by a network without an external trusted authority [6].

An Ethereum smart contract is converted into bytecode instructions [19] and this byte code is then appended to a block on the Ethereum Blockchain. Most Ethereum Smart Contracts are written in Solidity, a JavaScript-like programming language [14].

Solidity supports multiple inheritance. It follows the guidelines of the Python language and uses C3 Linearization [14], which forces a specific order in the inheritance of classes.

Every byte code level operation in Ethereum carries an execution weight for every execution cycle. This weighted number is known as gas in the Ethereum ecosystem [47]. Each unit of gas has a variable price, driven by demand on the network, and developers tend to design smart contracts that can minimize the cost associated with execution. Inheritance in Solidity plays a significant role in reducing the gas cost associated with deployment and execution. Additionally, the use of inheritance as 'aggregation' is not a language feature of Solidity. One workaround to implementing aggregation is through inter contract communication between separately deployed contract addresses ("internal transactions") which results in an additional cost for execution [40].

To demonstrate the impact of inheritance on the cost, we deployed two toy smart contracts on the Ethereum Testnet [23] and measure gas consumption. The first toy example illustrated in Listing 1 exploits inheritance for implementation and the second, functionally identical example (Listing 2), does not.

Listing 1: Example of Solidity contract with inheritance

```
function fit(string _a) public { a = _a; }
function display(string name) internal view
returns(string)
{ return string(abi.encodePacked(a, name)); }
contract derived is Base {
    function retrieve() public view returns(string) {
        return display("Alice"); } }
```

Listing 2: Example of Solidity contract without inheritance

```
contract Display { string w;
    function Set(string _a) public { a = _a; }
    function display(string name) public view
    returns(string)
    { return string(abi.encodePacked(a, name)); } }
contract Request { address display;
    constructor(address _display) public { display = _display; }
    function Output(string a) public
    { Display(display).Set(a); }
    function Get() public view returns(string)
    { return Greeting(greeting).display("Alice"); } }
```

The results from our cost analysis are reported in Table 2. Listing 1, using inheritance, yields a 54.24% lower deployment cost and a 5.4% lower execution cost. We expect that this significant difference in cost may impact the design choices made by Solidity developers in favor of inheritance usage.

This example suggests two hypotheses: 1) Developers will favor a high DIT to limit the *cost of deployment*. 2) As inheritance induces functional dependencies between a child and a parent class, it may

Table 1: Cost associated with inheritance

Contract Type	Deployment Cost	Execution Cost
Inheritance (Listing 1)	264164 gas	42966 gas
No Inheritance (Listing 2)	575809 gas	45412 gas

Figure 1: Study Design

result in a high cost of execution if used extensively. To reduce the *cost of execution*, developers will favor a low number of children per class resulting in a low width of the inheritance tree.

4 EMPIRICAL STUDY

Unlike traditional commercial software systems, source code for a substantial proportion of deployed smart contracts is accessible in the public domain, as trust is established through independent verification of information (transaction and/or code). However, the blockchain only provides a bytecode level source code for each deployed smart contract. Hence, we use existing public repositories to retrieve the Solidity source code. Our study design is based on the software metric evaluation guidelines put forth by [16] and we segment our study design into four phases (Figure 1) which are now described :

4.1 Smart Contract Selection

Given that all records of smart contract deployment, are publicly accessible, an archive node¹ may be used to access this public data from the Ethereum network. To extract publicly available smart contracts from Ethereum, we deploy a full node. The node communicates with other nodes in the network to retrieve the history of all blocks in Ethereum.

Once the deployed node has retrieved the history of the Ethereum blockchain, we use an open-source parser to extract data from the full node [1]. For the scope of our study, we are only interested in two fields present in the parsed block: the address and bytecode.

The resultant data set contained over one million smart contracts. We restrict our analysis to a specific type of smart contract: ERC-20, due to the high monetary value associated with ERC-20 tokens [17]. ERC-20 is a standard defined by Ethereum to issue digital tokens on the platform [10]. We also restrict inclusion in our sample to prominently used ERC-20 smart contracts to limit the sample size in this preliminary study. We define prominence based on the market capitalization of the ERC-20 token during the observation.² The

¹An archive node refers to a computing device that stores all the data present in the blockchain.

²The observations were conducted for a period of 24 hours on 10-01-2020. The value of market capitalization was retrieved from [3]

restriction on the type (ERC-20) and prominence (contracts with market capitalization greater than 5000 USD) allow us to capture potentially high-quality smart contracts. After the selection of these 244 smart contracts, we extract their associated source code.

4.2 Smart Contract Extraction

Smart contracts are available in the public domain in the form of byte code. To extract information related to inheritance, this byte code must be converted to a high-level language such as Solidity. This conversion is traditionally done by using a decompiler to regenerate the high-level source code from byte code. However, as outlined by [21], all the decompilation approaches for Solidity are significantly limited and produce an intermediate representation, which might not be an accurate mapping to the actual source code. To overcome this limitation, we obtain verified smart contracts on Etherscan [3]. Again, due to the open nature of the blockchain platform, most smart contract developers are incentivised to publish their Solidity code online. Etherscan provides one such repository of verified smart contracts [3].

For our study, we were able to extract source code of 229 of 244 shortlisted smart contracts by using the Etherscan API. Once the source code of shortlisted smart contracts is successfully obtained, we perform the metric calculation.

4.3 Metric Calculation

For each shortlisted smart contract, we calculate the inheritance metrics outlined in Section 2. To aid the process of metric calculation, we use the open-source static code review tool proposed by [22]. However, that static tool does not allow for the calculation of the number of children. We accomplish this through a manual investigation.

4.4 Data Analysis

The 229 extracted Solidity files employed in this study are available in the project repository³. To calculate the number of children, we generate the inheritance tree diagram for all the shortlisted smart contracts. Figure 2 is an illustration of an inheritance tree diagram for a contract from our test sample. We can extract both the value of the DIT and the number of children from the diagram. However, a smart contract often contains multiple classes, to accommodate multiple inheritance paths per contract, we report the *highest* DIT in the tree, along with an average number of children per class.

5 RESULTS

We report findings from our analysis in Table 2. We observe that smart contracts, on average, tend to have a DIT average of 3.29. The highest value of DIT observed in our test sample was 7, with the lowest value being 0. It should be noted that the average reported value for DIT is still below the upper threshold of 5 set by [12, 13].

In empirical results from [43], they reported an average DIT value of 1.25, 1.54, and 0.89 for systems with 5.6k, 21.3k, and 16.0k lines of code, respectively. We obtained an average of 3.29 for DIT for the Solidity contracts we studied, where the contracts were, on average, 328 LOC long. This suggests a preference towards using

³Solidity source code and respective depth of inheritance graphs are available at <https://anonymous.4open.science/r/a35ba37f-9e20-4073-9309-bcf8d74b75d5/>.

Figure 2: Inheritance Tree Diagram for CRO contract

Table 2: Inheritance Metrics for Smart Contracts

Metric	Average	Median	Standard Deviation
Depth of Inheritance	3.29	3.6	1.40
Number of Children	0.99	1.0	0.45

inheritance in smart contracts when compared to conventional programming languages, particularly when normalized for LOC. Despite the overall trend towards favoring inheritance, we report that the lowest value of DIT observed for our data sample is 0, which, according to [12], violates the OO paradigm. This implies that the lower bound on DIT values set by [12] is not suitable for all smart contracts, as inheritance is not always required in order to design an executable smart contract: 4% of smart contracts in our data sample reported a DIT value of 0.

The higher average inheritance finding points to differences between traditional OO software and smart contracts: possibly because of the cost associated with smart contract execution, as presented in Section 2. Surprisingly, the average number of children (NOC) for our data sample was 0.99, which seems to oppose the recommendation of [12], where they advocate for a higher number of children, for more code reuse. We speculate that the execution cost of employing extensive code reuse may be the reason behind a low number of children per class, as argued in hypothesis 2. Code reuse is usually achieved by calls to existing code, which in smart contracts may result in higher cost, as every call carries an execution cost. It may be more cost-efficient to rewrite the piece of code rather than employing extensive code calls. In the case of NOC, the trade-off is between a high degree of code reuse, i.e., a high value of NOC, and a lower cost of execution, which, overall, seems to result in the observed low value of NOC. Our study reports that the smart contract with the highest number of children per class had a value of 2.12. This is quite low in comparison to the highest number of children per class of 16, as reported in [43].

6 CONCLUSION

Measurement of quality, reliability, and security are some of the application areas of software metrics [16]. As software systems tend to vary significantly depending on development and deployment environment, a single interpretation of a suite of metrics may not apply to all types of software systems.

In this paper, we performed an evaluation of prominent smart contracts to assess the applicability of traditional inheritance based software metrics in a smart contract context (Section 4). We observe that the metrics for this environment do not equate to the same metrics for their more traditional environment, suggesting that factors related to context are at play: For example, the smaller source code size of the Ethereum smart contract. This smaller size may be a result of the functional limitations imposed by the decentralized nature of the Ethereum platform and, along with a significant difference in cost associated with execution, may result in different interpretations of, and different values for that metric suite.

Results from the static and manual code investigation reported a trend towards a higher depth of inheritance in examined smart contracts in accordance with hypothesis 1. But we also note that the observed average value DIT (2) still falls below the higher threshold of 5 proposed by [12]. We argue that due to the less complex nature of smart contracts, and the lower cost associated with inheritance, the reported values of DIT may signal a deliberate design choice of developers. We also analyzed and reported the average number of children per smart contract. The results suggest a lower NOC in accordance with our hypothesis 2, arguably due to the cost of execution associated with more child classes, as illustrated by the toy example in Section 2.

As the execution of these smart contracts attracts an execution cost known as a transaction fee, the design decisions may be driven by the need to minimize the cost. We also argue that the economics of smart code execution may drive the choice of preferring inheritance over alternate design choices such as composition. The opportunity cost associated with the higher width of inheritance (more child per class) may also be the reason behind the low value of NOC. Hence, we argue that a direct application of traditional OO metrics to smart contracts may not yield useful results. The notion of complexity and inheritance is significantly different in smart contracts due to the resource constraints of the execution environment.

However, the study presented in this ERA paper is limited by the choice of examined smart contracts. We have only studied a small sample of contracts. Our argument against the use of traditional inheritance metrics is based on the results from this sample, which may be biased based on the prominent use-case of smart contracts. That is, the analyzed smart contracts are sampled from pool of ERC-20 contracts and share a significant degree of similarity. We also acknowledge that the static parser and manual analysis phase may suffer from false positives. We intend to conduct the study on a larger data-set in the future. We also aim to expand our metric evaluation to other metrics. Examining more traditional metrics for smart contracts may guide new thresholds for metrics specific to smart contract context, and prompt new metrics. A further investigation into this field may prove to be vital for the smart contract development. We also intend to examine security metrics for smart contracts. The security of smart contracts has been extensively explored [27] and can be used as a measure of quality.

ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094

REFERENCES

- [1] [n.d.]. Ethereum Blockchain Parser. https://github.com/alex-miller-0/Ethereum_Blockchain_Parser
- [2] [n.d.]. Solidity Documentation. <https://solidity-doc-test.readthedocs.io>
- [3] 2019. verify & publish contract source code. <https://etherscan.io>
- [4] Nour Ali, Sean Baker, Ross O’Crowley, Sebastian Herold, and Jim Buckley. 2018. Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering* 23, 1 (2018), 224–258.
- [5] Sven Amann, Hoan Anh Nguyen, Sarah Nadi, Tien N Nguyen, and Mira Mezini. 2018. A systematic evaluation of static api-misuse detectors. *IEEE Transactions on Software Engineering* 45, 12 (2018), 1170–1188.
- [6] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security*. Springer.
- [7] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. 2015. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE.
- [8] Jim Buckley, Sean Mooney, Jacek Rosik, and Nour Ali. 2013. JITTAC: a just-in-time tool for architectural consistency. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 1291–1294.
- [9] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3 (2014), 37.
- [10] Vitalik Buterin and Fabian Vogelsteller. 2015. ERC20 Token Standard. URL: https://theethereum.wiki/w/index.php/ERC20_Token_Standard (2015).
- [11] Michelle Cartwright. 1998. An empirical view of inheritance. *Information and Software Technology* 40 (1998).
- [12] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, 6 (1994), 476–493.
- [13] John Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1, 2 (1996).
- [14] Chris Dannen. 2017. *Introducing Ethereum and Solidity*. Springer.
- [15] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. 2018. Smart contracts vulnerabilities: a call for blockchain software engineering?. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 19–25.
- [16] Norman Fenton and James Bieman. 2014. *Software metrics: a rigorous and practical approach*. CRC press.
- [17] Gianni Fenu, Lodovica Marchesi, Michele Marchesi, and Roberto Tonelli. 2018. The ICO phenomenon and its relationships with ethereum smart contract environment. In *International Workshop on Blockchain Oriented Software Engineering*. IEEE.
- [18] Ferit and Steve Ellis. 2018. Composition Over Inheritance - Gas Efficiency. <https://ethereum.stackexchange.com/questions/59994/composition-over-inheritance-gas-efficiency>
- [19] Christopher K Frantz and Mariusz Nowostawski. 2016. From institutions to code: Towards automated generation of smart contracts. In *IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE.
- [20] Erich Gamma. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Education India.
- [21] Neville Grech, Lexi Brent, Bernhard Scholz, and Yannis Smaragdakis. 2019. Giga-horse: thorough, declarative decompilation of smart contracts. In *41st International Conference on Software Engineering*. IEEE Press.
- [22] Peter Hegedus. 2019. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. *Technologies* 7, 1 (2019), 6.
- [23] Kedar Iyer and Chris Dannen. 2018. The ethereum development environment. In *Building Games with Ethereum Smart Contracts*. Springer.
- [24] Ralph E Johnson and Brian Foote. 1988. Designing reusable classes. *Journal of object-oriented programming* 1 (1988).
- [25] Tara Kelly and Jim Buckley. 2006. A context-aware analysis scheme for bloom’s taxonomy. In *14th IEEE International Conference on Program Comprehension (ICPC’06)*. IEEE, 275–284.
- [26] Kirk Knoernschild. 2002. *Java design: objects, UML, and process*. Addison-Wesley Professional.
- [27] Iuon-Chang Lin and Tzu-Chun Liao. 2017. A Survey of Blockchain Security Issues and Challenges. *IJ Network Security* 19, 5 (2017).
- [28] M. Marchesi. 2018. Why software engineering is important for blockchain software. In *International Workshop on Blockchain Oriented Software Engineering*. <https://doi.org/10.1109/IWBOSE.2018.8327564>
- [29] Reza M Parizi, Ali Dehghantanha, et al. 2018. Smart contract programming languages on blockchains: An empirical evaluation of usability and security. In *International Conference on Blockchain*. Springer, 75–91.
- [30] Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Tonelli, and Michele Marchesi. 2019. A massive analysis of Ethereum smart contracts empirical study and code metrics. *IEEE Access* 7 (2019), 78194–78213.
- [31] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. 2017. Blockchain-oriented software engineering: challenges and new directions. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 169–171.
- [32] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter Tichy. 2003. A controlled experiment on inheritance depth as a cost factor for code maintenance. *Journal of Systems and Software* 65, 2 (2003).
- [33] Chris Ryder. 2004. *Software measurement for functional programming*. Ph.D. Dissertation. Computing Lab, University of Kent.
- [34] Chris Ryder and Simon J Thompson. 2005. Software metrics: measuring Haskell. In *Trends in Functional Programming*. 31–46.
- [35] Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. 2019. Assessing The Security Implication Of Bitcoin Exchange Rates. *Computers & Security* (2019).
- [36] Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. 2019. Privacy and Security Analysis of Cryptocurrency Mobile Applications. In *Fifth Conference on Mobile and Secure Services*. IEEE.
- [37] DN Salter. 2016. *Bitcoin: How Can a Virtual Currency Attain Real Market Value?* Lulu Press, Inc.
- [38] Ilya Sergey and Aquinas Hobor. 2017. A concurrent perspective on smart contracts. In *International Conference on Financial Cryptography and Data Security*. Springer, 478–493.
- [39] Pradeep Singh, KD Chaudhary, and Shrish Verma. 2011. An investigation of the relationships between software metrics and defects. *International Journal of Computer Applications* 28 (2011).
- [40] StackOverflow. 1968. Composition over inheritance in Solidity - Gas efficiency. <https://stackoverflow.com/questions/52515714/composition-over-inheritance-in-solidity-gas-efficiency>
- [41] Ramanath Subramanyam and Mayuram S. Krishnan. 2003. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on software engineering* (2003).
- [42] Nick Szabo. 1997. The idea of smart contracts. *Nick Szabo’s Papers and Concise Tutorials* 6 (1997).
- [43] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. 1999. An empirical study on object-oriented metrics. In *Proceedings sixth international software metrics symposium*. IEEE.
- [44] Roberto Tonelli, Giuseppe Destefanis, Michele Marchesi, and Marco Ortu. 2018. Smart Contracts Software Metrics: a First Study. *arXiv preprint arXiv:1802.01517* (2018).
- [45] Farshad Toosi, Jim Buckley, Ashish Rajendra Sai, and Andrew Le Gear. 2018. Reverse engineering the blockchain as illustrated using eigen decomposition. (2018).
- [46] Barbara Unger, Lutz Prechelt, and Michael Philippsen. 1998. *The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications*. Univ., Fak. für Informatik.
- [47] Gavin Wood et al. [n.d.]. Ethereum: A secure decentralised generalised transaction ledger. ([n.d.]).