PASO: A Web-Based Parser for Solidity Language Analysis

Giuseppe Antonio Pierro, Roberto Tonelli

PASO: A Web-Based Parser for Solidity Language Analysis

Giuseppe Antonio Pierro
Dep. of Mathematics and Computer Science
University Of Cagliari
Cagliari, Italy
antonio.pierro@gmail.com

Roberto Tonelli
Dep. of Mathematics and Computer Science
University Of Cagliari
Cagliari, Italy
antonio.pierro@gmail.com

***Abstract*—Smart Contracts are computer programs which implement and execute transactions and manage business logic on a decentralized public ledger. Smart Contracts can be written in different programming languages and for different Blockchains. Currently the most used language for Smart Contracts is Solidity and the most used platform is the Ethereum Blockchain. Assessing the quality of Smart Contract programs is an important task required to professional programmers, especially when a programming language has so powerful economic implications. It is therefore crucial to provide professional programmers with tools for the evaluation of Smart Contracts. In software engineering, software metrics has been defined and used to measure software quality and, more in general, to qualify software under the principle "You Can't Manage What You Don't Measure". For the Solidity programming language there are only a few Standalone Applications to analyse the Smart Contract metrics. The aim of this paper is first to build a tool for the practical computation of a specific set of Solidity source code metrics, so that the set will be extensible in the future according also to Solidity compiler evolution, second to fully enable a web-based usage of the tool to access the metrics of the Solidity programming language. The tool, PASO, differently from the existing application, is able to give software metrics values for Smart Contracts written in Solidity programming language just using a web browser.**

***Index Terms*—Smart Contracts, Ethereum Blockchain, Solidity Programming Language, Web-based tool, Solidity Grammar, Parser Generator, PASO**

I. Introduction

The Ethereum Blockchain is a public ledger that keeps record of all Ethereum related transactions and stores and execute software code implementing the so called "Smart Contracts". It is shared among all participants to the Ethereum Blockchain and it is based on a reward mechanism as an incentive for miners to execute the transactions network [1]. Smart Contracts are computer programs that implement and execute transactions on the Ethereum Blockchain. The Smart Contracts can be written in different programming languages and Solidity is the most used and the most recent one [2], [3]. Since its birth in May 2015, a number of versions of the Solidity programming language have been released. In particular, in the last year, thirteen new versions of Solidity have been released and for each new release, several bugs have been fixed. Because of the very high release rate of new versions with several bugs fixed, it is plausible to think that the Solidity programming language is not mature especially when compared to other languages. It is still in evolution and new constructs are introduced once in a while to cover new needs and programming practices specific for Blockchain oriented software and accordingly to such changes also software metrics change. In addition, because of different use cases in Industry applications with high impact on economy, code analysis tools are needed to help developers to check whether Smart Contracts comply with Solidity coding rules [4].

Assessing the quality of Smart Contract programs, as for any other program written in any other language, is a difficult and often subjective process . Having some heuristics and metrics that measure the properties of an applications source code provides a useful starting point, and observing these metrics over time can identify important trends [5], [6], [7].

Software metrics are a useful tool also to monitor software evolution in time. Even if in the specific case of Smart Contracts deployed in the Ethereum Blockchain software changes require a new deployment, there are questions about software code which are quite common for developers and that can be answered to by looking at software metrics also for Blockchain Oriented software development. Given that Smart Contract deploy is costly, it is very practical to ask and get answers to these questions before the Smart Contract is deployed on the Blockchain. A Smart Contract (Solidity) Developer could ask: 1) Is the program becoming harder or easier to maintain? 2) how closely connected two functions or two Contract are? 3) How much repetitive code is there? 4) How big and complex are the Smart Contract structures? Metrics and heuristics methods can provide Smart Contract software developers with answers to these questions, and some tools can help to ensure that the software developers are notified if their code exceeds certain thresholds for any metrics they deem to be important.

Recently, in the market, new tools are coming out to help developers to access Solidity programming language metrics. In academic research, examples of these Standalone Applications tools are Pharo Solity Parser [8] and SolMet [9], a static solidity code analysis written in Java. The main constraint of the previous Standalone Applications is that they depend on other server programming languages or framework that usually are not installed in a personal computer. For example, to use Pharo Solidity Parser, software developers need to install Pharo Smalltalk IDE [10] or, to use the SolMet, software developers need to install a specific version of the Java Virtual Machine. Only after installing a specific version of the software required, the already existing tools provide metrics of a Solidity Smart Contract code, such as the number of code lines for each Smart Contract, the Cyclomatic Complexity, the number of functions, the number of parameters for each function. Furthermore these tool do not follow the evolution of the Ethereum Virtual Machine (EVM) where new Solidity constructs can be introduced during time to tackle new and unexpected problems arising when Smart Contracts are set at work in real world environments. Unfortunately, at the present time, there are no modular and updatable tools neither web-based and easy to use tools to calculate software metrics for Solidity Smart Contract. For these reasons, this paper presents an updatable web-based tool, PASO (a PArser for SOlidity), which detects errors and provides the computation of important software metrics in any web browser available in any operating system. Furthermore, the paper presents and discusses the general components of the online PASO tool. The advantage of the online tools is that the Solidity developers does not need to install any software in their operating system. PASO is indeed a web-based tool which provides programmers with better insight into the Smart Contract code they are developing. By taking advantage of code metrics provided by PASO, Solidity developers can understand which part of the Smart Contract should be modified or tested in a comprehensive manner. Single Solidity developers or Smart Contract development teams can identify potential risks, understand the current state of a Smart Contract, and track progress during software development. Developers can use PASO to generate code metrics data that measure the complexity and maintainability of the Smart Contract code.,Code metrics data can be generated for a single Smart Contract, but also for a set of Smart Contracts. Finally, Solidity metrics can be useful to compare a given Smart contracts with others whose source code is already available.

The outline of the article is the following: Section II reviews the related work on blockchain analysis platforms. Section III presents the hypothesis of the paper. Section IV presents the general components of the online tool PASO and the approach we embraced to define the metrics. Section V describes the threats to validity for the research. Finally, in Section VI, we make our final remarks and draw some conclusion.

II. Related Work

As mentioned in section I, at the date hereof, a web-based and updatable tools to measure source code metrics of Solidity Smart Contract do not exist yet. For other programming languages, such as JavaScript, there are instead a lot of web-based tools to measure the metrics. JSHint [1] is a Static Code Analysis Tool for JavaScript that detects errors and potential problems in JavaScript code.

[1]https://jshint.com/about/

For what regards Solidity there are only Standalone Applications. SolHint [2] is a command-line tool to analyse the Solidity code for potential errors. It also provides both security and style guide validations. Pharo Solidity Parser uses SmaCC (Smalltalk Compiler-Compiler) and relies on Solidity grammar specification to build a parser that can be used to measure metrics for Smart Contract written in Solidity [11].

Zhang et al. [12] proposed metrics for measuring the Web Ontology Language (OWL). Although the OWL language is different from Solidity, the underlying concepts are similar. The paper [12] is also inspired by the concept of software metrics. The proposed metrics were analytically evaluated against Weyuker's criteria [13]. It also performed empirical analysis on public domain ontologies to show the characteristics and usefulness of the metrics.

III. Motivation

The Solidity language grammar definition changes very often and, consequently, the tools to measure the software metrics needs to be updated very frequently. Every time a new version of the Solidity program language is released, the existing Standalone Applications need to be updated accordingly from both the authors and the end users perspectives. This extra work, from the end-user point of view, could be avoided by using a web-based tool, which requires to be updated just by refreshing the web page. Of course also the engine behind the computation needs to be updated according to the new Solidity release. The PASO tool, presented in this paper, accomplishes to all these requirements. A practical option, which is implemented in PASO, is the idea to have all components on the client side with no need to have a server. This solution has the advantage that there is no need to have any server to maintain and manage. What is needed to run the program is just a web browser, which is installed in every operating system. In addition, PASO has offline functionality, i.e. it can work completely offline once all the PASO components, coded in CSS, HTML and JavaScript, have been downloaded.

IV. PASO Components

PASO is available and can be tested at this link. [3]

The main components needed to build and run PASO are:

- Solidity Grammar,
- PASO Parser,
- PASO Metrics,
- PASO GUI (Graphical User Interface).

The following sections give a general definition for each main component for a better understanding of the work made to realize the tool PASO.

A. Solidity Grammar

A programming language is a set of commands, strings of characters readable by programmers but easy to translate

[2]https://protofire.github.io/solhint/

[3]https://aphd.github.io/paso/

to machine code. It has grammar and semantic rules. The grammar is a set of rules that define how the commands have to be arranged to make sense and to be correctly translated to the machine code. Semantics is a set of meanings assigned to every command of the language and it is used to correctly translate the program to machine code [14].

Figure 18 shows a piece of Solidity grammar according to the ANTLR rules. Each ANTLR rule consists of a name, followed by a colon, followed by its definition, and terminated by a semicolon. The sourceUnit symbol is the entry node of the grammar. Nonterminal nodes in ANTLR have to be lowercase. Terminal nodes have capitalized names, like EOF. EOF is a special terminal node, defined by ANTLR, meaning the end of the input. In particular, it stands for the end of the file, even though the input may also come from a string or a network connection rather than just from a single file.

The symbol "|" represents the alternation operator, the symbol "*" is the repetition operator, and parentheses are used for grouping, in the same way we have been using for a natural language grammar reading. Optional parts can be marked with the symbol "?". In ANTLR, Terminal nodes can be defined using regular expressions, but fixed strings are not permitted. For example, here are some Terminal patterns used in the Solidity grammar written in ANTLR syntax and with ANTLR naming convention: COMMENT → "/*" ".*?" "*/".

The Solidity grammar definition is available at this web site. [4]

```
grammar Solidity;

sourceUnit
  : (pragmaDirective | importDirective |
      contractDefinition)* EOF ;

  ...

functionDefinition
  : natSpec? 'function' identifier? parameterList
       modifierList returnParameters? ( ';' |
      block ) ;

returnParameters
  : 'returns' parameterList ;

modifierList
  : ( modifierInvocation | stateMutability |
      ExternalKeyword
    | PublicKeyword | InternalKeyword |
        PrivateKeyword )* ;
```

Listing 1: This code shows how the Solidity grammar looks like as an ANTLR source file

B. PASO Parser

The PASO Parser is generated from a Parser Generator. Figure 1 shows the input and the output of a Parser Generator. A parser generator is an application which generates a parser: it takes the Solidity grammar as input and automatically generates a source code named Parser. The parser is a function that takes the sequence of characters of a Smart Contract as input, attempts to match the sequence with the grammar and produces a parse tree as output. Figure 2 shows the input and the output of the PASO Parser.

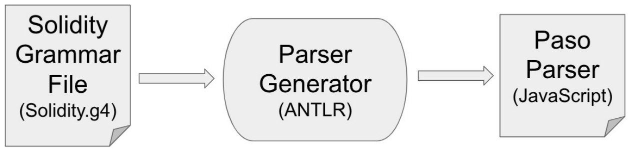
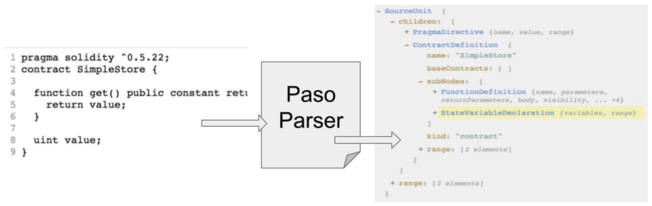


Fig. 1: The Parser Generator takes a file containing the Solidity grammar rules. It produces a PASO Parser, i.e. a parser in JavaScript computer language that can be run in a client browser.

A parse tree or parsing tree is an ordered, rooted tree that represents the syntactic structure of the source code according to the grammar. The root of the parse tree is the starting Nonterminal node of the grammar. In a parse tree, a Nonterminal node is a node of the parse tree which is either a root or a branch of the tree, whereas a Terminal node is a node of the parse tree which is a leaf.

There are different parser generator applications for various programming languages. To the aims of this paper, it is necessary to use a Parser Generator that can generate a Parser in a client-side scripting language, like JavaScript. Among the different Parser Generators, we chose ANTLR4 (ANother Tool for Language Recognition), precisely because it can produce a Parser in JavaScript programming language that can run on the client part together with the GUI part.



```
pragma solidity ^0.5.22;
contract SimpleStore {

  function get() public constant retu
    return value;
  }

  uint value;
}
```

```
- SourceUnit {
  - children: [
     + PragmaDirective {name, value, range}
     - ContractDefinition {
          name: "SimpleStore"
          baseContracts: [ ]
        - subNodes: [
            + FunctionDefinition {name, parameters,
              returnParameters, body, visibility, ... +4}
            + StateVariableDeclaration {variables, range}
          ]
          kind: "contract"
        + range: [2 elements]
       }
    ]
  + range: [2 elements]
  }
```

Fig. 2: Example of input and output of the PASO Parser.

C. PASO Metrics

In software engineering software metrics has been defined and used to measure software quality and, more in general, to qualify software under the principle "You Can't Manage What You Don't Measure". Code metrics can be used to detect any characteristic in the source code that possibly indicates a major problem of the code. They therefore act as a useful alert to detect a problem and improve the overall design of the code. An example of problem to detect is duplicated code, i.e. identical or very similar code which exists in one or more parts of the program [15], [16]. Code metrics can also be used to identify functions or a Smart Contracts formed by many lines of code (LOC), or to measure the Cyclomatic complexity, i.e. the existence of too many branches or loops. A high value of Cyclomatic complexity metric or/and the LOC metric may indeed indicate that a function needs to be broken into smaller functions, or that it can be simplified [17], [18].

[4]https://github.com/solidityj/solidity-antlr4/blob/master/Solidity.g4

Many object-oriented metrics have been proposed over the last decade [19] and most of the metrics used in the tool are derived from "C&K" metrics. C&K metrics were proposed by Chidamber and Kemerer in 1991 for object-oriented software [20] and details of them can be found in [21]. The metrics discussed in the paper include C&K metrics and add further metrics. The overall set of metrics displayed in the PASO tool are therefore divided into two categories: 1) Object oriented metrics used to measure properties of object oriented programming languages, such as java, smalltalk, C++ [22] 2) Solidity metrics, which are specific for Solidity programming language. PASO displays some of the metrics taken from the two categories 3

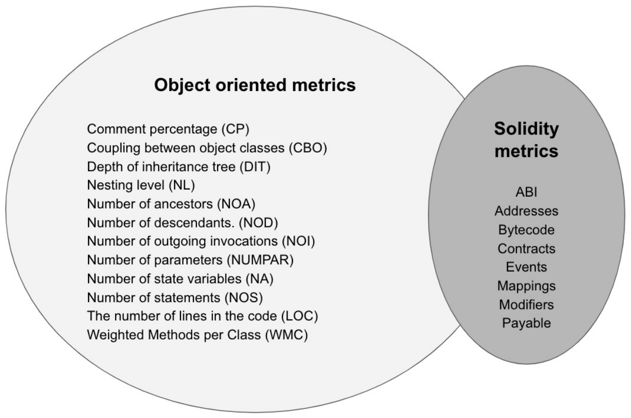

Object oriented metrics
Comment percentage (CP)
Coupling between object classes (CBO)
Depth of inheritance tree (DIT)
Nesting level (NL)
Number of ancestors (NOA)
Number of descendants. (NOD)
Number of outgoing invocations (NOI)
Number of parameters (NUMPAR)
Number of state variables (NA)
Number of statements (NOS)
The number of lines in the code (LOC)
Weighted Methods per Class (WMC)
Solidity metrics
ABI
Addresses
Bytecode
Contracts
Events
Mappings
Modifiers
Payable

Fig. 3: The two ovals respectively represent the set of Object oriented metrics (on the left) and the set of metrics that are specific to Solidity Language (on the right).

1) Object Oriented Metrics: In the previous literature, static code analysis tools generate metrics for OOP (object oriented programming) languages. Some of the most used metrics are: 1) the number of code lines, 2) coupling, i.e. the number of connections a file or a class has to other files or classes, 3) the number of arguments for a function, etc. Table I lists the most used metrics in the scientific literature which are implemented in PASO tool.

2) Solidity Metrics: Solidity programming language has some peculiarities that makes it unique when compared to other programming languages [23].

Table II lists the metrics that it is possible to obtain by parsing the Smart Contract code with the PASO tool. These metrics are defined only for Solidity programming languages.

D. PASO GUI

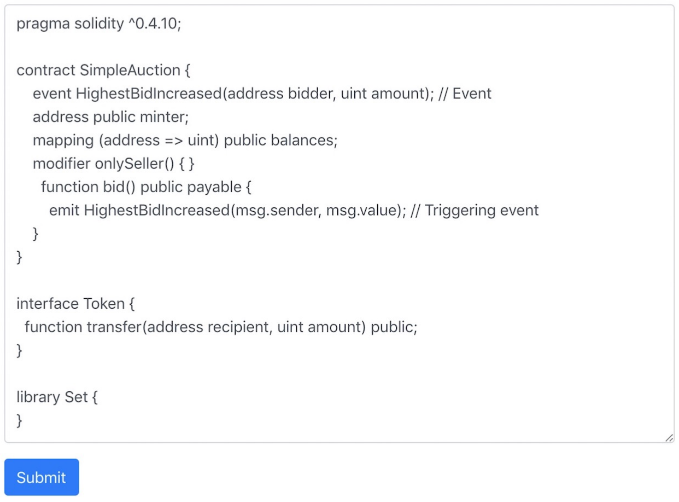
The PASO GUI is the PASO component that allows users to interact with the PASO Parser. The GUI (Fig. 4) is divided into two sections: 1) a textarea where the user can write or paste the Smart Contract or several Smart Contracts, 2) a section for results where the user can see the values of the metrics displayed. Figure 4 presents the two sections: the textarea is shown on the left (Fig. 4a), while on the right some metrics value are shown (Fig. 4b), corresponding to the Smart Contract written in the textarea.

TABLE I: Some object oriented metrics

Metric name	**Description**
LOC	It indicates the number of lines in the code. A high number might indicate that a contract is trying to do too much work and should be split up. It might also indicate that the contract or method is hard to maintain.
Methods	Number of Methods. Contracts with too many methods may be trying to do too much, or in any case may be more difficult to maintain.
NL	The nesting level metric denotes the sum of the deepest nesting level of the control structures within the functions of a class, library or interface.
NUMPAR	Number of parameters. It counts how many parameters a function has.
NOS	Number of statements. The number of statements metric counts how many statements there are in a class, library or interface.
DIT	Depth of inheritance tree. The depth of inheritance metric measures how deep a class, library or interface is in the inheritance tree.
NOA	Number of ancestors. The number of ancestors metric counts all the different direct or transitive ancestors of a class, library or interface.
NOD	Number of descendants. The number of descendants metric measures how many different direct or transitive descendants a class, library or interface has.
CBO	Coupling between object classes. The coupling between object classes metric in the OO paradigm measures the number of classes that the actual class is connected to (by using the class as an attribute type, method parameter or return value, etc.).
NA	Number of state variables.
NOI	Number of outgoing invocations (i.e. fan-out). The number of outgoing invocations metric measures how many different functions are called from a function in a class or library.
WMC	Weighted Methods per Class (WMC) is an object-oriented metric to measure complexity in a class.
CP	Comment percentage.

V. LIMITATION

Studying the quality of a software might be a difficult, often subjective process. Having some metrics value that measure an applications source code provides a useful starting point to improve the existing code. Static code analysis tools such as PASO can spot many different kinds of mistakes, but cannot detect if the Solidity Smart Contract produces the correct system behavior. A solidity software developer should always combine tools like PASO with unit and functional tests as well as with code reviews. Indeed, PASO can check the correctness of the grammar written by the programmer and compute the associated metrics, but cannot understand whether the meanings assigned by the programmer are coherent, i.e. actually correspond to what the programmer wanted to achieve.

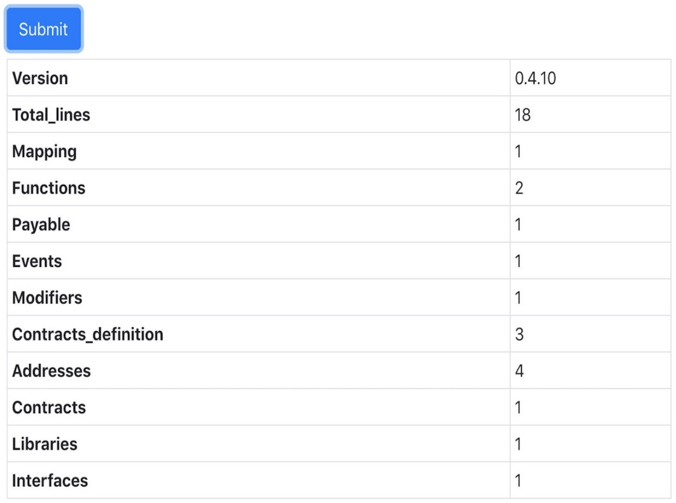

```
pragma solidity ^0.4.10;

contract SimpleAuction {
    event HighestBidIncreased(address bidder, uint amount); // Event
    address public minter;
    mapping (address => uint) public balances;
    modifier onlySeller() { }
      function bid() public payable {
        emit HighestBidIncreased(msg.sender, msg.value); // Triggering event
    }
}

interface Token {
  function transfer(address recipient, uint amount) public;
}

library Set {
}
```

(a) PASO GUI Textarea.



Version	0.4.10
Total_lines	18
Mapping	1
Functions	2
Payable	1
Events	1
Modifiers	1
Contracts_definition	3
Addresses	4
Contracts	1
Libraries	1
Interfaces	1

(b) PASO GUI Metrics.

Fig. 4: PASO GUI. Figure 4a shows the textarea where the user can write or paste the Smart Contract or several Smart Contracts. Figure 4b shows some metrics value corresponding to the Smart Contract written in the textarea.

TABLE II: Some Solidity Metrics

Metric name	**Description**
Payable	The number of Payable Functions.
Mappings	The number of Mapping types. Mappings, in Solidity programming language, can be seen as hash tables.
Modifiers	The number of Function Modifiers.
Addresses	The number of addresses.
Events	The number of Events.
Contracts	The number of Contracts.
ABI	The size of the ABI (Application Binary Interface).
Bytecode	The size of the Bytecode.

The PASO tool is limited in the interface design. For instance, it displays the metrics value in numerical form. An improvement of the tool could be the visualization of metrics by using a treemap, as for example in the work by Balzer [24]. PASO is also limited in the number of metrics it computes but is structured to be easily updated, also accordingly to the eventual future evolution of the Solidity EVM. Example of missing metrics in the PASO tool are: 1) the coupling metric that describes the number of connections a file or a contract has to other files or contracts, and 2) The Cyclomatic Complexity of a function. However, thanks to a modular design, it can be used as a basis for a richer implementation that gives more precise information to the developer and a more user-friendly graphical interface to the user.

VI. Conclusion and Future Work

The paper presents a fully web-based tool able to compute the Smart Contract Metrics. The goal has been achieved by using the ANTLR (ANother Tool for Language Recognition) parser generator. We gave the Solidity grammar as an input to the ANTLR Parser Generator, which has been used to create a JavaScript Parser. Finally we wrote the Solidity Code in the PASO GUI textarea, thus giving it as an input to the JavaScript Parser, which calculated and displayed the metrics values on the screen. Before implementing PASO, there were only Standalone Applications, such as Pharo Solidity Parser and SolMet, which have allowed to parse and to generate metrics for a Smart Contract written in Solidity program language. The research assessed the hypothesis that it is possible to build a completely web-based tool, PASO, able to achieve at least the same results of the previous Standalone Applications. The main advantage of having such web-based tool - when compared to the previous ones - are: 1) users have no need to install a third-party software, like Java or Smalltalk Pharo, 2) PASO is able to cope in a more efficient way with the countless updates of Solidity programming language: by using PASO, there is no need to update the Standalone Application, it is only needed to update the web page.

The number of metrics discussed in the paper are just a few: a complete list of metrics to be implemented in the PASO tool would indeed require more in-depth research that can be developed in future works. The aim of the present research was indeed limited to test the hypothesis that it is possible to build a fully and updatable web based tool to compute the metrics value of a Smart contract written in Solidity without installing any tools on users local computer. At the time of writing (December 2019), PASO is the only fully web-based tool that allows to parse and to generate metrics for a Smart contract written in Solidity program language. It represents a starting point for a future richer implementation, able to give more relevant information to the developers and a more user-friendly graphical interface to the user.

References

[1] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 254–269, New York, NY, USA, 2016. ACM.

[2] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, pages 91–96, New York, NY, USA, 2016. ACM.

[3] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. Blockchain-oriented software engineering: Challenges and new directions. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C 17, page 169171. IEEE Press, 2017.

[4] Giuseppe Antonio Pierro and Henrique Rocha. The influence factors on ethereum transaction fees. In *2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*, WETSEB '19, pages 24–31, Piscataway, NJ, USA, 2019. IEEE Press.

[5] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.

[6] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi. A massive analysis of ethereum smart contracts empirical study and code metrics. *IEEE Access*, 7:78194–78213, 2019.

[7] Roberto Tonelli, Giuseppe Destefanis, Michele Marchesi, and Marco Ortu. Smart contracts software metrics: a first study, 2018.

[8] Henrique Rocha, Stéphane Ducasse, Marcus Denker, and Jason Lecerf. Solidity parsing using smacc: Challenges and irregularities. In *Proceedings of the 12th Edition of the International Workshop on Smalltalk Technologies*, IWST '17, pages 2:1–2:9, New York, NY, USA, 2017. ACM.

[9] Péter Hegedűs. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, WETSEB '18, pages 35–39, New York, NY, USA, 2018. ACM.

[10] David Röthlisberger, Oscar Nierstrasz, Stéphane Ducasse, and Alexandre Bergel. Tackling software navigation issues of the smalltalk ide. In *Proceedings of International Workshop on Smalltalk Technologies (IWST'09)*. ACM Digital Library, 2009.

[11] John Brant and Don Roberts. The smacc transformation engine: How to convert your entire code base into a different programming language. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 809–810, New York, NY, USA, 2009. ACM.

[12] Hongyu Zhang, Yuan-Fang Li, and Hee Beng Kuan Tan. Measuring design complexity of semantic web ontologies. *J. Syst. Softw.*, 83(5):803814, May 2010.

[13] E. J. Weyuker. The evaluation of program-based software test data adequacy criteria. *Commun. ACM*, 31(6):668675, June 1988.

[14] Man Leung Wong and Kwong Sak Leung. Evolutionary program induction directed by logic grammars. *Evol. Comput.*, 5(2):143–180, June 1997.

[15] Giulio Concas, Michele Marchesi, Alessandro Murgia, Sandro Pinna, and Roberto Tonelli. Assessing traditional and new metrics for object-oriented systems. pages 24–31, 01 2010.

[16] G. Concas, C. Monni, M. Orr, and R. Tonelli. A study of the community structure of a complex software network. In *2013 4th International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 14–20, May 2013.

[17] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. Would you mind fixing this issue? an empirical analysis of politeness and attractiveness in software developed using agile boards. volume 212, 05 2015.

[18] Alessandro Murgia, Roberto Tonelli, Michele Marchesi, Giulio Concas, Steve Counsell, Janet McFall, and Stephen Swift. Refactoring and its relationship with fan-in and fan-out: An empirical study. *Proceedings of the Euromicro Conference on Software Maintenance and Reengineering, CSMR*, pages 63–72, 03 2012.

[19] Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., GBR, 1991.

[20] Norman E. Fenton and Martin Neil. Software metrics: Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE 00, page 357370, New York, NY, USA, 2000. Association for Computing Machinery.

[21] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA 91, page 197211, New York, NY, USA, 1991. Association for Computing Machinery.

[22] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. A relational approach to software metrics. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1536–1540, New York, NY, USA, 2004. ACM.

[23] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, WETSEB '18, pages 9–16, New York, NY, USA, 2018. ACM.

[24] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 165–172, New York, NY, USA, 2005. ACM.