Complexity Analysis of Decentralized Application Development Using Integration Tools

PATRIK REK, BLAŽ PODGORELEC, and MUHAMED TURKANOVIĆ, University of Maribor

Decentralized applications development on the Ethereum platform is becoming very popular in last few years. However, it requires developer to have knowledge and skills to integrate large number of components, such as smart contracts programming, IPFS decentralized storage, RPC calls or Truffle for smart contracts management and various JavaScript libraries (e.g. Web3.js, TruffleContract, JS-IPFS). This makes the development process very complex and difficult. At the same time, the developer has multiple front-end frameworks available, which however lack the ability to easily integrate the majority of previously mentioned components. To solve this problem, there are integration tools which address above issues and are intended to support a comprehensive development of decentralized applications (e.g. Drizzle, Vortex, Web3-React). The paper focuses on these integration tools and analyses the code complexity of decentralized application development using such tools. The analysis of code complexity was performed using multiple code complexity metric assessment methods.

1. INTRODUCTION

Due to the current stage of the Decentralized Applications and according to Gartner's 2018 Hype Cycle for Blockchain Technologies [Furlonger and Kandaswamy 2018], there is still lack of studies and tools with which the development and understanding of such applications could be easier. In order to achieve the level of productivity, the development of decentralized applications needs to be well studied, simplified, and documented. The decentralized application development process is currently marked as complicated because of multiple factors (e.g. lack of tools, good practices, and documentation), which are tightly related mainly with the purpose to facilitate the development process. In this article, we focus on the issues of the development process of the decentralized application, based on the Ethereum platform, as it is the most used platform for the development of such applications with the largest number of active developers among other platforms [StateOfTheDApps 2019].

Decentralized applications developer must be acquainted with the Solidity programming language, with which smart contracts on the Ethereum platform are developed. Smart contracts also represent the fundamental building blocks of decentralized applications [Modi 2018]. If we look from the traditional web application development perspective, smart contracts can be considered as a back-end logic of decentralized applications. Therefore, it is also necessary to know how to connect the aforementioned smart contracts within a blockchain network (e.g. permissioned or permissionless) with the end user in a user-friendly way, i.e. with decentralized applications. [Antonopoulos and Wood] The

The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No.P2-0057. This paper is also supported, in parts, by EC H2020 Project CONCORDIA GA 830927.
Author's address: P. Rek, B. Podgorelec, M. Turkanović, University of Maribor, Faculty of Electrical Engineering and Computer Science, Institute of Informatics, Koroška cesta 46, SI-2000 Maribor, Slovenia; email: patrik.rek@um.si; blaz.podgorelec@um.si; muhamed.turkanovic@um.si;

In: Z. Budimac and B. Koteska (eds.): Proceedings of the SQAMIA 2019: 8th Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, Ohrid, North Macedonia, 22–25. September 2019. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

connection between smart contract and the decentralized application front-end, which is usually implemented with the use of different frameworks (e.g. Angular, ReactJS, Vue.js) can be done with the help of the Web3 library or with the use of dedicated frameworks (e.g. Truffle, Embark). In case that the decentralized application includes any additional large data files, there is also an decentralized option for a file storage, i.e. a decentralized storage technology as IPFS, Swarm or StorJ. [Google 2018; Facebook Inc. 2019; Evan You 2018; Consensys 2018; Embark 2019; Protocol Labs 2018b; Ethersphere 2018; Storj Labs 2018]

The whole development process of decentralized applications is complex due to the related and various programming languages, tools, frameworks, and new architectural concepts, which will be described in more detail in the following chapters. The aim of the paper is to present integration tools that simplify comprehensive decentralized applications development and compare the process of development with and without such integration tools. We limit ourselves to the development process of the front-end building blocks of decentralized applications. To better assess the meaningfulness of using such integration tools, we have also performed a code complexity analysis using several metric assessment methods, thus concluding with the results of these.

2. PRELIMINARIES

In this section, we present a short overview of concepts of technologies used within the development of decentralized applications.

2.1 Blockchain technology

The blockchain is a sequence of connected data blocks that contain a complete list of transactions sent to a blockchain network. Each block consists of a head and body. Inside the head, there is the information about the version of the block, tree root hash value, timestamp, number of bits, nonce and hash value of the block are written, while body consists from the transactions and the transaction counter [Zheng et al. 2017]. The blockchain network usually consists of multiple nodes, which have equal privileges, and each of them possesses the same ledger with all blocks and transactions performed through the network [Zheng et al. 2017]. To achieve consistency among all nodes, they need to follow the network protocol, part of which is also the consensus mechanism (e.g., proof of Work, proof of stake, proof of authority) [Bach et al. 2018].

2.2 Smart contracts

A smart contract is a computer program that is stored and implemented on a blockchain network (e.g., Ethereum). Smart contracts represent the business logic of decentralized applications and are programmed arbitrarily. They are instantiated on the blockchain network within the special ”create smart contract” transaction. Once this transaction is included in the valid block, each account inside the blockchain network can use its specified functions. These functions are performed by sending the transactions to the address of the smart contract, and it is crucial to notice that all nodes of the blockchain network with the purpose of validation of this transaction perform the same action inside own virtual machines (e.g. Ethereum Virtual Machine). After that, the transaction results are irrevocably recorded in the blockchain [Aldweesh et al. 2018].

2.3 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a stack-based virtual machine that executes smart contracts. EVM executes the smart contracts based on the byte code and consumes a certain amount of gas for its execution. The gas is determined by cryptocurrency - Ether, which is necessary for performing operations on the blockchain network. The amount of gas required to perform individual operations is

predetermined in terms of the amount of memory, network activity, and processor work. The function which is defined by smart contract and it is called within a transaction sent on the blockchain network is executed inside EVM on all nodes of the blockchain network. [Wohrer and Zdun 2018; Aldweesh et al. 2018]

2.4 Distributed data storage

Data can be stored inside the persistent storage of smart contract, but because of the higher processor complexity and, on the other hand, the higher costs incurred by using the storage space of each network node, it is not suitable for smart contracts to store large number of data or even files inside it. For this purpose, the distributed and decentralized storage systems (e.g., IPFS, Swarm, StoreJ) can be used, which can operate independently of the blockchain or they can be used interchangeably with the blockchain network [Zheng et al. 2018].

2.5 Decentralized applications

Decentralized applications are applications that store all data about the performed operations in a distributed ledger (DL). Their business logic does not rely on any central entity but is unlike the traditional applications defined in smart contracts, which are part of DL. Those smart contacts are later (when the function is called) executed by every node of the distributed and decentralized blockchain network. Decentralized applications are usually open source and have a front-end part for interacting with the user, while the cryptographically signed data is processed within the blockchain network [Furlonger and Kandaswamy 2018]. The high-level decentralized application architecture which consists of four fundamental building blocks is shown in Figure 1, and these building blocks are executed in two different environments (local and network). The user interacts with the decentralized application

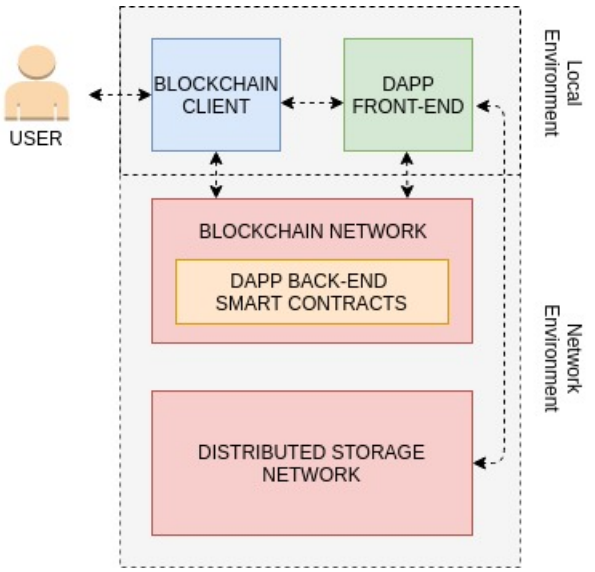

Fig. 1: Decentralized application architecture

through the usage of BC-CLIENT building block. BC-CLIENT store user credentials (cryptographic keys) with which the transactions are cryptographically signed, and provides the connection to the

blockchain network. Decentralized application front-end building block communicates with smart contracts deployed on the blockchain network and provides to the end user the user-friendly usage of decentralized application. If there is a need for additional data storage the decentralized application front-end also communicates with a distributed storage network. Both of the building blocks (decentralized application front-end and BC- CLIENT) mentioned above are executed in the local user environment - usually inside the browser. Blockchain and distributed storage networks are executed in a remote network environment [Pustišek and Kos 2018]. Purpose of both of these building blocks is described in Section 2.

3. DECENTRALIZED APPLICATION DEVELOPMENT

3.1 JavaScript frameworks

In order to connect the logic with the presentation level of web applications, there are many frameworks, including React, Angular, and Vue.js [Facebook Inc. 2019; Google 2018; Evan You 2018].

3.1.1 *React.* React is a JavaScript library for the design and construction of user interfaces, which is based on various components. The components are encapsulated and have their own state, which by combining several different components allows the development of more complex user interfaces. The logic of the components is written with the JavaScript programming language, while the presentation level is written in XML-like syntax called JSX. An application based on the React framework can be run on the server-side as well as on the client-side. Because of the JavaScript-based components, they are flexible and handle the state that is sent to the server or, in our case, into the blockchain network. The more advanced React developers also use Redux, which is the advanced application state management tool. Without the use of Redux, it is necessary to store the application state for each component, which in the case of blockchain means reconnecting with the network on each interaction of the user with the application [Facebook Inc. 2019; Abramov and Redux 2019].

3.1.2 *Angular.* Angular framework is similar to React, but it is based on a strongly typed programming language - TypeScript. It enables the easier development of web applications. Angular uses declarative templates and real-time debugging. The platform itself also allows building applications that will be used on different devices and platforms. For templates, the enriched HTML language is used. Similarly as React, Angular also encourages the development of modules and components, divided into smaller portions of the user interface, since it is possible to achieve easier collaboration between developers with this approach to development [Google 2018].

3.1.3 *Vue.* Vue is a progressive framework for building user interfaces. It is, like React, based on the JavaScript programming language. Vue is designed to be used incrementally, which means it does not need to be used exclusively, but can be used in conjunction with other frameworks and is only used partially. The framework is focused solely on the presentation level, which makes it easy to integrate with other libraries and projects. Despite its simplicity, it is possible to run sophisticated web applications in combination with modern tools and libraries. In Vue, the template is written in an enriched HTML language, just like in Angular. When building applications using the Vue framework, it is also recommended that the entire project be separated into individual components [Evan You 2018].

3.2 Support technologies

There are several tools and JavaScript libraries that simplify development of decentralized applications, which will be described in this section.

3.2.1 *Web3.js.* A dedicated JavaScript library is required to connect a decentralized application to the blockchain network. One of these libraries, which is widely used, is web3.js. This is a collection of libraries that allow interaction with local or remote Ethereum node using HTTP, WebSocket or IPC (inter-process communication) connection. The library collection is divided into three major parts [Ethereum 2016]:

(1) web3-eth, representing an access point to the Ethereum blockchain and smart contracts,
(2) web3-shh, which represents a way to connect to the Whisper protocol for peer-to-peer communication,
(3) web3-utils, which contains useful features for developers of decentralized applications (conversion between different types of encodings, validation of records ...).

With the help of the web3.js library, it is also possible to subscribe to events, which are triggered upon successful completion of a state change in smart contracts. An important fact is that web3.js and its functions are executed synchronously, which adds an additional challenge to the development of decentralized applications, since the above-described frameworks operate asynchronously, so it is necessary to manually implement asynchronous operations in cooperation with the web3.js library [Ethereum 2016].

3.2.2 *Ethereum development frameworks.* There are several frameworks that help full-stack Ethereum developers in decentralized applications development.

Truffle. One of such tools is called Truffle. It is a development and testing framework for the Ethereum blockchain. Truffle provides tools to compile, deploy and migrate smart contracts to your network, automatically test smart contracts, manage networks and load smart contracts on any number of public and private networks, an interactive console for direct communication with smart contracts, and a tool for easy continuous integration of smart contracts. Using Truffle to deploy smart contracts, we can use the JavaScript truffle-contract library to interact with them within decentralized applications, which makes it easier for developers to initialize and use smart contracts. Thus, it is no longer necessary to manually obtain an application binary interface, as Truffle automatically takes care for this. This makes the development cycle a bit shorter and easier because the use of functions is more similar to the standards used by JavaScript [Consensys 2018].

Embark. Embark, like Truffle, is a development framework for building decentralized applications and deploying on distributed networks. It connects to the Ethereum blockchain and IPFS distributed storage. It provides automatic deployment of smart contracts to the blockchain network, testing, distribution of decentralized applications to the distributed IPFS file system and peer-to-peer messaging. It also includes the Cockpit application, which represents an interface for managing smart contracts and blockchain networks. It also comes with the JavaScript library EmbarkJS, which brings similar functionality as the truffle-contract library in the case of the Truffle environment. It allows connection of a decentralized application with a blockchain network, sending transactions and calls, subscribing to events, usage of distributed IPFS file system, and working with user accounts [Embark 2019].

3.2.3 *JS-IPFS.* IPFS is a distributed file system that aims to connect all computers with the same file system. It is a versioned file system that manages files in different places and also tracks their variations. The system is similar to the operation of the Bittorrent protocol [Protocol Labs 2018b].

Connection to IPFS can be established using HTTP calls to one of the nodes or, in the case of decentralized applications, using the js-ipfs JavaScript library, which allows connection to the IPFS node and adding or getting files. The js-ipfs library needs to be initialized and connected to local or remote node after installation. Once the connection is established, the developer gets access to the application

programming interface for the IPFS file storage. It can listen to various events that are triggered by faults or interruptions. At the same time, it can add files or other data to the IPFS network, where it receives their hash value as a return, and the files can also be read from the network by entering their hash value. All the functions of the js-ipfs library are asynchronous [Protocol Labs 2018a].

3.3 Integration tools for a comprehensive development

Due to the specifics of decentralized application development and its many frameworks and libraries, the development process is relatively complicated. There are various tools that aim to simplify and unify the development process of comprehensive decentralized applications. Examples of such tools are Drizzle, Vortex and Web3-React. In the following sections, we will describe these tools and compare them with each other according to the functionalities they provide.

3.3.1 *Drizzle.* Drizzle is a collection of libraries designed to develop the front-end of decentralized applications. The goal of the tool is to make the development of these easier and more predictable. The core of the Drizzle is based on the Redux store. Drizzle ensures the synchronization of smart contract data and transaction data. It provides responsive data obtained from smart contracts, including balances, events, and transactions. These can be used by the developer in a similar way to traditional functions and methods in JavaScript applications. Drizzle also boasts declarativity, where the developer determines what information he needs and thus does not use the computational power for unnecessary data [ConsenSys 2018].

Since Drizzle is developed by ConsenSys within the Truffle Suite, it is desirable to use it in cooperation with Truffle to work properly. Drizzle is installed using the npm package manager and can be used in combination with any Redux-enabled framework. Therefore, the most appropriate framework to use with Drizzle is React, for which Drizzle developers have also developed components that simplify the development of decentralized applications. Drizzle is compatible with version 1.0 of web3 and WebSocket protocol [ConsenSys 2018].

React components for Drizzle include (1) a component that checks blockchain and smart contracts connection status and displays specific views for loading and failure status, (2) a component for making smart contract calls and displaying returned data and (3) a component that generates an input form from smart contract methods. With these components and using the entire React framework, the development of decentralized applications is accelerated [ConsenSys 2018].

3.3.2 *Vortex.* Vortex is a Redux store that deals with transactions, smart contracts, events, accounts, method calls, web3 status, IPFS file retrieval, etc. It is mainly used with the React framework, since it allows better responsiveness without refreshing and makes fewer web3 requests for even better results. It can also be used to read files from a distributed IPFS file system and to cache them in the Redux store [horyus 2019].

It provides fast loading of all smart contract instances within Redux store. It stores information about all user actions in the cache and tracks the user's transactions and the balance of the Ethereum account. All data from smart contracts is consistently obtained from the blockchain and saved into the cache. When the event is emitted on a smart contract, it is automatically retrieved. The development of decentralized applications is thus possible in a more uniform way [horyus 2019].

Vortex can work in conjunction with Embark or Truffle. Installation is possible using the npm package manager. Features or components for React can then be used. The current weakness of the Vortex tool is that the current version is already outdated because it does not fit with some of the more recent libraries it refers to. A new version, called ethvtx, is being developed, but it has not yet been developed to the same extent as the first version of the Vortex tool [horyus 2019].

3.3.3 *Web3-react.* Web3-react is a simple framework for developing decentralized applications of the Ethereum platform using the React library. It supports most of the commonly used tools that establish the web3 connection to the blockchain network (e.g. MetaMask, Infura, Portis). It is developer friendly because it establishes the web3.js instance and manages the settings. It does not offer as much functionality as the previously described Drizzle and Vortex, but provides a basis for the development of advanced functionality that manages every aspect of the decentralized application. It solves the asynchronism issue, which, instead, the developer would have to implement himself by creating promises for all existing Web3.js functions [horyus 2019].

3.3.4 *Comparison.* An exact comparison of functionalities of the described tools is displayed in the table I.

Table I. : Comparison of comprehensive decentralized applications development tools

	Drizzle	**Vortex**	**Web3-react**
Connection establishment	Yes	Yes	Yes
Smart contracts	Yes	Yes	No
Making transactions	Yes	Yes	No
Events subscription	Yes	Yes	No
IPFS connection	No	Yes	No
Asynchronous operation	Yes	Yes	Yes
Redux store	Yes	Yes	No
Limitations	Truffle & Redux	React	React
Development state	Production ready	Production ready, but it does not comply with the latest versions. New version is still in development.	Production ready

4. COMPLEXITY

The complexity of the software is the measurement of the software code quality. It is considered as the amount of hardware capabilities required to interact with the software. For the computer, the complexity can be described as the amount of time and memory needed to perform computational operations. For the developer, the complexity is defined as complexity of performing tasks such as programming, debugging and testing. A high level of complexity in an individual part of the software (function, object, class, etc.) is considered bad. The complexity measurements must be carried out using certain metrics that address different software attributes [Debbarma et al. 2012; Bhatia and Malhotra 2014].

There are many metrics for measuring the complexity of the software. Number of lines of code (LOC) is the oldest and very often used. This counts each line of program code, including comments and blank lines, while Effective Lines of Code (ELOC) ignores comments and blank lines. The advantages of this metric are in its simplicity and general acceptance, with the disadvantage being that it is not necessary that a smaller number of code lines mean a higher quality code [Pawade et al. 2016].

The ABC metric consists of three components, namely Assignment, Branch and Condition. Assignments represent the number of variables, branches represent the number of function calls, and the conditions are the number of conditional statements in the program code. Components represent components of the ABC vector, whose length is the value of the ABC code. Higher value means greater complexity [Pawade et al. 2016].

Halstead's metrics (HSS) extend the LOC metric to treat the program as a set of operators and operands. They define multiple software complexity attributes [Halstead and H. 1977]:

—number of distinct operators: n_1
—number of distinct operands: n_2
—total number of operators: N_1
—total number of operands: N_2
—program length - total number of operators and operands in program code: $N = N_1 + N_2$
—program vocabulary - total number of distinct operators and operands: $n = n_1 + n_2$
—program volume - size of the entire program, calculated according to the following equation: $V = N \times \log_2 n$
—degree of difficulty - the difficulty of writing or understanding the program, calculated by the equation: $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$
—effort - the actual time of programming according to the equation: $E = V \times D$
—time required to develop in seconds: $T = \frac{E}{18}$
—number of bugs in the software: $B = \frac{V}{3000}$

Based on the above attributes, the complexity of the software can be determined according to Halstead principles [Debbarma et al. 2012; Pawade et al. 2016].

The metrics described so far have been quantitative, while the following metrics address the control flow. This is analysed according to the program flowchart. One such metric is called cyclomatic complexity (MCC) and is defined by the equation $CC = e - n + 2$, where e is the number of edges and n is the number of nodes in the graph. Cyclomatic complexity can more easily be calculated using the equation $CC = \Pi - s + 2$, where Π is the number of decision statements and s the number of endpoints. The decision statements are all *if, while, for, case, catch* and other dependency items [Pawade et al. 2016].

For object-oriented programming languages, which include JavaScript, there are specific metrics, and one such is the Weighted Methods per Class metric (WMC). In WMC, we sum complexities of all methods, calculated from the selected metric for computing complexity, such as, for example, cyclomatic complexity [Beranič 2018; Nuñez-Varela et al. 2017].

5. COMPLEXITY ANALYSIS

We have developed an example decentralized chat application in two ways. In both ways, we have manually written the source code, while using a private Ethereum blockchain network. While developing the decentralized application, we used the Truffle tool for smart contracts' management and the React frontend library. In the first case, we have not used any comprehensive decentralized application development tools, whereas in second we have used Vortex. We have selected Vortex due to its highest number of functionalities, as displayed in table I. Both decentralized applications used the same smart contract and had the same functionalities. Users would register to the application where they can see all the users and conduct chat conversations with them. All the messages are stored on the IPFS platform and noted in the blockchain network. Users interact with decentralized applications using transactions to smart contracts. Each chat with another user is separate smart contract.

To calculate the complexity of the program code we used the previously described metrics, namely LOC, ABC, Halstead Metrics, MCC and WMC. Since smart contracts are the same in both examples, we have excluded them from the research. Calculations for both examples are displayed in Table II, where N represents the program length, n is the program vocabulary, V the volume of the program,

D is the difficulty level, E is the effort, T is time to develop in seconds and B number of bugs in the program. LOC, ABC, MCC and WMC are calculations by each metric respectively. As we can see, the method which uses Vortex is only better in difficulty level, MCC and WMC metric of software complexity. According to other metrics, the example without use of comprehensive decentralized application development tool is less complex than the one using Vortex.

Table II. : Comparison of code complexity between example without integration tool usage and example which uses Vortex.

	Without integration tool	**Using Vortex**	**% change**
LOC	352	465	+ 32,1
ABC	132,1	150,9	+ 14,2
N	1314	1469	+ 11,8
n	146	154	+ 5,5
V	9447,4	10674,9	+ 13,0
D	38,3	38,0	- 0,8
E	361904,8	405182,4	+ 12,0
T	20105,8	22510,1	+ 12,0
B	3,1	3,6	+ 16,1
MCC	3,8	2,5	- 34,2
WMC	19	15	- 21,1

6. DISCUSSION

Metrics which measure program code size (i.e. LOC, ABC and Halstead metrics) show better results for example without using integration tools, which is due to the higher number of components used in example using Vortex tool, since the tool provides component for each scenario. However, the structural complexity has decreased, as noted using MCC and WMC metrics in Table II.

7. CONCLUSION

Using Vortex tool, there is a decrease in structural complexity of decentralized application, while code size has increased. We think that existing software complexity metrics are not appropriate for web applications, since they are specific and more components do not always mean more complex software, since components are intended to diversify application and split it into smaller independent parts which make further development easier.

With the use of comprehensive integration tools for decentralized applications development such as Vortex or Drizzle, developers obtain unified access to each functionality and do not have to focus on lower level of decentralized application development. These tools are made with the goal to change developer's focus from decentralized technologies to functionalities.

REFERENCES

Dan Abramov and Redux. 2019. Redux · A Predictable State Container for JS Apps. (2019). Retrieved June 6, 2019 from https://redux.js.org/

Amjad Aldweesh, Maher Alharby, Ellis Solaiman, and Aad van Moorsel. 2018. Performance benchmarking of smart contracts to assess miner incentives in Ethereum. In *2018 14th European Dependable Computing Conference (EDCC)*. IEEE, 144–149.

Andreas M. Antonopoulos and Gavin Wood. *Mastering Ethereum : building smart contracts and DApps*. https://books.google.si/books?id=nJJ5DwAAQBAJ

LM Bach, Branko Mihaljevic, and Mario Zagar. 2018. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 1545–1550.

Tina Beranič. 2018. Identifikacija pomanjkljive kode na osnovi mejnih vrednosti programskih metrik. (2018). https://dk.um.si/IzpisGradiva.php?id=72643

Sonam Bhatia and Jyoteesh Malhotra. 2014. A survey on impact of lines of code on software complexity. In *2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014)*. IEEE, 1–4. DOI:http://dx.doi.org/10.1109/ICAETR.2014.7012875

ConsenSys. 2018. Truffle Suite — Documentation — Drizzle — Drizzle Quickstart. (2018). Retrieved June 6, 2019 from https://truffleframework.com/docs/drizzle/quickstart

Consensys. 2018. Truffle Suite — Sweet Tools for Smart Contracts. (2018). Retrieved June 6, 2019 from https://truffleframework.com/

Mrinal Kanti Debbarma, Nirmalya Kar, and Ashim Saha. 2012. Static and dynamic software metrics complexity analysis in regression testing. In *2012 International Conference on Computer Communication and Informatics*. IEEE, 1–6. DOI:http://dx.doi.org/10.1109/ICCCI.2012.6158825

Embark. 2019. Getting Started — Embark. (2019). Retrieved June 6, 2019 from https://embark.status.im/docs/overview.html

Ethereum. 2016. web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. (2016). Retrieved June 6, 2019 from https://web3js.readthedocs.io/en/1.0/

Ethersphere. 2018. Swarm 0.3 documentation. (2018). https://swarm-guide.readthedocs.io/en/latest/introduction.html

Evan You. 2018. Vue.js. (2018). Retrieved June 6, 2019 from https://vuejs.org/

Facebook Inc. 2019. React – A JavaScript library for building user interfaces. (2019). Retrieved June 6, 2019 from https://reactjs.org/

David Furlonger and Rajesh Kandaswamy. 2018. Hype Cycle for Blockchain Technologies, 2018. *gartner.com* (July 2018).

Google. 2018. Angular. (2018). Retrieved June 6, 2019 from https://angular.io/

Maurice H. (Maurice Howard) Halstead and Maurice H. 1977. *Elements of software science*. Elsevier. 127 pages. https://dl.acm.org/citation.cfm?id=540137

horyus. 2019. Introduction - Vortex. (2019). Retrieved June 6, 2019 from https://vort-x.readthedocs.io/en/develop/

Ritesh Modi. 2018. *Solidity Programming Essentials: A beginner's guide to build smart contracts for Ethereum and blockchain*. Packt Publishing Ltd.

Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, and Carlos Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *Journal of Systems and Software* 128 (jun 2017), 164–197. DOI:http://dx.doi.org/10.1016/J.JSS.2017.03.044

Dipti Pawade, Devansh J. Dave, and Aniruddha Kamath. 2016. Exploring software complexity metric from procedure oriented to object oriented. In *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*. IEEE, 630–634. DOI:http://dx.doi.org/10.1109/CONFLUENCE.2016.7508195

Protocol Labs. 2018a. IPFS Documentation. (2018). Retrieved June 6, 2019 from https://docs.ipfs.io/

Protocol Labs. 2018b. IPFS is the Distributed Web. (2018). Retrieved June 6, 2019 from https://ipfs.io/

Matevž Pustišek and Andrej Kos. 2018. Approaches to front-end iot application development for the ethereum blockchain. *Procedia Computer Science* 129 (2018), 410–419.

StateOfTheDApps. 2019. State of the Dapps Statistics. (2019). Retrieved June 6, 2019 from https://www.stateofthedapps.com/stats

Storj Labs. 2018. StorJ: A Decentralized Cloud Storage Network Framework. (2018). https://storj.io/storjv3.pdf

Maximilian Wohrer and Uwe Zdun. 2018. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2–8.

Qiuhong Zheng, Yi Li, Ping Chen, and Xinghua Dong. 2018. An Innovative IPFS-Based Storage Model for Blockchain. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 704–708.

Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, 557–564.