A Preliminary Empirical Exploration of Quality Measurement for JavaScript Solutions

DAVID KOSTANJEVEC, MAJA PUŠNIK, MARJAN HERIČKO, BOŠTJAN ŠUMAK,
University of Maribor
GORDANA RAKIĆ and ZORAN BUDIMAC, University of Novi Sad

Contrary to the increasing popularity of JavaScript programming language in the field of web application development, the numerical expression of evidence about the quality of solutions developed in this language is still not reliable. Based on the preliminary literature review, which is the main subject of this paper, this area has not yet been fully explored. Measurement is done by application of general and object-oriented metrics, which can reflect only general characteristics of the solution, while the specifics related to the programming language are not expressible by existing metrics. Due to the popularity of the language and the increasing number of JavaScript projects, the idea is to determine appropriate metrics and approach to measurement for their application in practice. Finally, the measurement approach will be implemented in the SSQSA Framework to enable its application. The preliminary research presented in this paper was conducted during a student course of Empirical research methods at the University of Maribor and therefore is limited in number of included papers, depth of research, and analysis of its contents, which restricts us to preliminary conclusion only, but places the foundation and justifies the described forthcoming research.

Categories and Subject Descriptors: **H.0. [Information Systems]:** *General*; **D.2.8 [Software Engineering]:** ***Metrics*** — *Complexity measures; Product metrics*; **D.2.9. [Software Engineering]:** *Management — Software quality assurance (SQA)*

General Terms: *Software quality assurance*

Additional Key Words and Phrases: *software metrics, quality metrics, JavaScript, software analysis, JavaScript analysis*

1 INTRODUCTION

JavaScript is an increasingly popular programming language that is dynamically interpreted and has a simple syntax. The Angel List Job Posting (USA) Agency recorded 30.6% jobs as JavaScript oriented in 2016 (Chen 2017). Popularity however brings a lot of changes and innovations in the development of web applications and services. JavaScript was primarily used only for client-side functionalities and was running in a browser. However, it is now running on the servers as well (Capan 2013). Applications and services are written using the JavaScript language and executed on the Node.js platform, a relatively new technology which is continually evolving. JavaScript is also becoming the main programming language for developing hybrid mobile applications using frameworks such as Cordova, Phonegap, Titanium, Facebook's React Native, etc. (ValueCoders 2017).

Today Node.js is one of the most innovative solutions for building servers and web/mobile applications. The field is growing rapidly with the valuable contributions of other developers and technological giants (Abhishek 2015). Node.js also provides a platform for publishing third party packages that are available through the Node Packet Manager (NPM) public repository. At the time of writing, there are more than 540,000 packages, which can be simply installed, managed, and updated through package management (B. Pfretzschner in L. B. Othmane 2016) and (E. Wittern, P. Suter 2016). Today, there are many solutions based on Node.JS framework that constantly evolving. The most well-known Node.js frameworks are Express.js, Hapi.js, Socket.io, Mojito, Mean.js, Sails.js, Koa and others (Noeticsunil 2017). Following all frameworks as well as their success in terms of quality is

Authors address: David Kostanjevec, Maja Pušnik, Marjan Heričko, Boštjan Šumak, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia; email: kostanjevec.david@gmail.com maja.pusnik@uni-mb.si marjan.hericko@um.si bostjan.sumak@um.si
Gordana Rakić, Zoran Budimac University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovica 4, 21000 Novi Sad, Serbia; email: gordana.rakic@dmi.uns.ac.rs, zjb@dmi.uns.ac.rs

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Belgrade, Serbia, 11-13.09.2017. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

very difficult, as new Node.js frames are constantly appearing (Abhishek 2015), (Noeticsunil 2017). Because of the large number and how fast different frameworks are being developed, and updated, it would be necessary to have appropriate tools enabling for analysis and comparison of these frameworks in terms of different quality dimensions. The analysis should include benchmarking criteria for (1) the framework, (2) what the framework is intended for, and (3) quality of the software developed on the framework. The quality of the developed projects would be tested by using JavaScript software metrics that are still not precisely defined.

The overall goal of this research is to establish and to apply an appropriate measurement to monitor quality of applications written using the JavaScript programming language. Upon the establishment of an measurement approach, the SSQSA Framework (Gordana Rakić 2015) could be exploited to measure JavaScript solutions according to it. Existing frameworks so far extract different metrics for measuring quality of solutions written in different languages, including JavaScript, but with no processing of extracted numbers that would provide the user with useful information about the quality of the analyzed solution. In order to employ an analytic technique to generate a certain information about the quality of a JavaScript solution, we have to define which of extracted metrics we will take into account and how they measure the quality attributes in JavaScript solutions as well as provide quantitative data about the quality variables or factors.

Following described goals, we conducted a preliminary research to identify software metrics potentially appropriate for assessing the quality of JavaScript solutions. The goal of this paper is to review approaches for measuring complexity and other quality attributes applied to JavaScript solutions, in order to select applicable metrics. Based on the literature review, we tried to identify metrics comparable to classic OO metrics that are, according to the existing literature, used to measure quality attributes of JavaScript solutions. In the following sections, we first describe the review approach. Next, results of the review are presented and discussed, where we focus on the selection of the metrics and evaluate their suitability to the specified purpose. Finally, the last two sections provide limitations of this preliminary research and conclusions with the plans for future work.

2 REVIEW APPROACH

Within the overall examination, several research questions will stand in the foreground as guidelines for further research:

1. Which existing software metrics are appropriate for measuring the quality of JavaScript solutions?
2. What are the shortcomings of the existing JavaScript metrics?
3. What JavaScript metrics already exists and what are their reference values?
4. Is it possible to extend the existing set of JavaScript metrics for comprehensive software quality measurement of JavaScript based solutions?
5. What tools for static analysis of JavaScript code already exists and which metrics do the support?

Guided by these research questions, preliminary research was focused on determination of the current situation in measuring the quality of JavaScript solutions. Specific research questions that should be answered by this preliminary literature review are:

1. Which software metrics have been used in existing literature for measuring the quality of JavaScript solutions?
2. Are these metrics appropriate to measure the quality of JavaScript solutions?

Following databases of scientific papers were used for searching of relevant literature: (1) Web of Science, (2) Science Direct and (3) IEEE Xplore Digital Library. The search was focused on research published in the past 5 years (from 2012 to 2017) in the field of computer science and informatics. The

used keywords were “JavaScript Metric”, “static analysis”, “JavaScript analyse” and “software analyse”. Table *I* presents combination of keyword sequence for each database and number of results (conducted at the end of May 2017).

Table I. Set of keyword combinations

	A combination of a keyword sequence	**# of results**
IEEE Xplorer	((javascript) OR Javascript) AND ((metric)OR (static analysis*) OR (software analys*)OR ("Abstract":staticanalys*))	28
ScienceDirect	(TITLE-ABSTR-KEY(javascript)) AND (TITLE-ABSTR-KEY(metric) OR TITLE-ABSTR-KEY(static analys*) OR TITLE-ABSTR-KEY(software analys*))	27
Web of Science	TS=(javascript AND metric*) OR SU=(static analys* OR software analys*)	38

13 papers were found and included in this preliminary research, addressing listed general questions, without a specific selection criterion. However, the preliminary research in this paper is not a systematic literature review and focuses on discovery of software metrics applied on JavaScript solutions and exploration of their usability. Other research questions will be addressed in more depth in future work.

3 REVIEW RESULTS

Authors in (Y. Ko, H. Lee, J. Dolby 2015) present a new approach for analyzing large JavaScript applications with static analysis. The research describes their tool and focuses on the analysis of products written in the JavaScript programming language.

The authors in (S. Mirshokraie, A. Mesbah 2013) focused on the JavaScript programming language and proposed a set of mutations that are specific to web applications. They suggest a technique that complements the static and dynamic analysis of the program to guide the process of generating mutations, on sections of the program code, where there is a greater likelihood of errors or could affect the output of the program. The paper presents the MUTANDIS tool and gives an assessment of effectiveness (S. Mirshokraie, A. Mesbah 2013).

Similarly, the paper (S. Rostami, L. Eshkevari, D. Mazinanian 2016) presents the JSDeodorant tool, a plug-in for Eclipse. The tool allows us to observe classes in JavaScript, it can identify the divergence between modules and utilities when examining objects in a program code. The main purpose of the paper is the presentation of the techniques, provided by the JSDeodorant tool, the comparison of the tool with the more familiar tool JSClassFinder, and the quantitative and qualitative evaluation of the results to recognize their limitations and possibilities for future improvements (S. Rostami, L. Eshkevari, D. Mazinanian 2016). The authors in (Mesbah 2013) present the JSNOSE tool that uses the technique of detecting bad programming patterns. The tool compares the program code with a set of 13 samples of JavaScript to find "smelly" parts of the code. Parts of bad code can have a poor effect on the whole project, maintenance or understanding. The JSNOSE tool in the paper is also tested on eleven web applications (Mesbah 2013). By increasing the use of JavaScript frameworks for web applications, there is an increasing demand for the quality of the written code including fast maintenance, reliability and speed.

Based on the preliminary literature review, we concluded that there are already several metrics defined, mostly connected to object-oriented paradigm and all of them are not completely suitable for JavaScript program code. However, JavaScript metrics were included in a research conducted by (Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez 2017), where a total of 190 different metrics was identified for the object-oriented paradigm. The most common metrics and total occurrences of specific metric, are presented in the Table II for programming languages Java, AspectJ, C++, C, C#, Jak, Ada, COBOL, Pharo, PHP, Python, Ruby, as well as for JavaScript.

Table II. Commonly applied metrics

Metric	Total occurrences
Weighted Methods per Class (WMC)	89
Coupling Between Objects (CBO)	89
Lack of Cohesion in Methods (LCOM)	86
Depth of Inheritance Tree (DIT)	81
Lines of Code (LOC)	79
Number of Children (NOC)	77
Response for a Class (RFC)	72
Number of Methods (NOM)	57
Cyclomatic Complexity (V(G))	55
Number of Attributes (NOA)	43
Fan-out (FANOUT)	27
Fan-in (FANIN), Number of Public Methods (NOPM)	22
Lines of Comments (LCOMM)	21
Afferent Couplings (Ca), Efferent Couplings (Ce)	20
Lack of Cohesion in Methods 2 (LCOM2)	18

Metric	Total occurrences
Lines of Source Code (SLOC)	17
Lack of Cohesion in Methods 3 (LCOM3)	16
Cohesion Among Methods (CAM), Number of Classes (NCLASS), Number of Parameters (NPAR)	15
Nesting level (NEST), Message Passage Coupling (MPC)	14
Number of Overridden Methods (NMO), Number of Public Attributes (NOPA)	13
Lack of Cohesion in Methods 4 (LCOM4)	12
Effort (E)	11
Instability (I)	11
Lack of Cohesion in Methods 5 (LCOM5)	11
Tight Class Cohesion (TCC)	11
Abstractness (A) Loose Class Cohesion (LCC) Number of Statements (STAT) Number of Methods Inherited (NMI)	10
Attribute Hiding Factor (AHF), Class Cohesion (CC), Data Abstraction Coupling (DAC), Data Access Metric (DAM), Method of Aggregation (MOA), Method Hiding Factor (MHF), Normalized Distance from Main Sequence (Dn)	Less than 10

All listed metrics (Table *II*) however cannot be applied to JavaScript program code. According to the research (L. H. Silva, D. Hovadick, M. T. Valente, A. Bergel, N. Anquetil 2016), which presents the JSClassFinder tool, only few metrics are suitable. The mentioned tool creates a structure model from the program code and is object-oriented, enabling visualization of the code (UML class diagram) and presenting information about classes, methods, attributes, deductions, and relationships. Metrics obtained with the tool and suitable for JavaSript measurement according to (L. H. Silva, D. Hovadick, M. T. Valente, A. Bergel, N. Anquetil 2016) are the following:

1. Number of classes,
2. Number of method,
3. Number of attributes,
4. Number of subclasses,
5. Depth of inheritance tree.

The selected five metrics are not enough for comprehensive measuring of JavaScript solution quality or complexity; therefore, extended research will be conducted, focusing on additional research questions.

4 LIMITATIONS AND THREATS TO VALIDITY

This research has limitations that have to be identified and discussed. Since the research was conducted by students within the course Empirical research methods it is not complete and other papers of existing research must be taken into consideration. Furthermore, this is only a preliminary

research from the empirical perspective. After comprehensive empirical research, confirmed conclusions have to be challenged in practical application to examine real usability of selected metrics and discover their weaknesses. Final conclusions should lead to answers to the research questions.

5 CONCLUSION AND FUTURE WORK

Several metrics have been identified and are already commonly used for software quality measurement; however most of them are reportedly not suitable for JavaScript source code measurements. A set of metrics dedicated to measuring the quality of JavaScript solutions must be defined and evaluated. The preliminary research in this paper is an initial effort to examine the field of JavaScript metrics, providing basic insight into the research field.

The future work will include a systematic literature review of the field, practical examination of selected metrics and tools and definition of metrics suitable for JavaScript program code measurement in terms of quality and complexity according to defined research questions. Extended formation of appropriate measurements for JavaScript source code will be implemented by integrating them with the SSQSA Framework.

6 ACKNOWLEDGMENTS

This joint work is enabled by bilateral project “Multidimensional quality control for e-business applications” between Serbia and Slovenia (2016-2017). Furthermore, the two authors from University of Novi Sad were partially supported by the Ministry of Education, Science, and Technological development, Republic of Serbia, through project no. OI 174023. The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. (J5-8230).

7 REFERENCES

Abhishek. 2015. “10 Best Node.js Frameworks For Developers.” https://www.devsaran.com/blog/10-best-nodejs-frameworks-developers.

Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, Carlos Soubervielle-Montalvo. 2017. “Source Code Metrics: A Systematic Mapping Study.” *The Journal of Systems and Software* 128: 164–97.

B. Pfretzschner in L. B. Othmane. 2016. “Dependency-Based Attacks on Node.js.” *IEEE Cybersecurity Development*: 66.

Capan, Tomislav. 2013. “Why The Hell Would I Use Node.js? A Case-by-Case Tutorial.” *Toptal*. https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js.

Chen, Yi-Jirr. 2017. “What Programming Language Should a Beginner Learn in 2017?” *Codmentor community*. https://www.codementor.io/codementorteam/beginner-programming-language-job-salary-community-7s26wmbm6.

E. Wittern, P. Suter, in S. Rajagopalan. 2016. “A Look at the Dynamics of the JavaScript Package Ecosystem.” In *IEEE/ACM 13th Working Conference on Mining Software Repositories*, 351–61.

Gordana Rakić. 2015. SSQSA: Set of Software Quality Static Analysers, Extendable and Adaptable Frameworkfor Input Language Independent Static Analysis “SSQSA Ontology Metrics Front-End1.” University of Novi Sad.

L. H. Silva, D. Hovadick, M. T. Valente, A. Bergel, N. Anquetil, in A. Etien. 2016. “JSClassFinder: A Tool to Detect Class-like Structures in JavaScript”.

Mesbah, A. M. Fard A. 2013. “JSNOSE: Detecting JavaScript Code Smells.” In *IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 116–25.

Noeticsunil. 2017. “Node.js Frameworks: The 10 Best for Web and Apps Development.” http://noeticforce.com/best-nodejs-frameworks-for-web-and-app-development.

S. Mirshokraie, A. Mesbah, in K. Pattabiraman. 2013. “Efficient JavaScript Mutation Testing.” In *Verification and Validation 2013 IEEE Sixth International Conference on Software Testing*, 74–83.

S. Rostami, L. Eshkevari, D. Mazinanian, N. Tsantalis. 2016. “Detecting Function Constructors in JavaScript.” In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 488–492.

ValueCoders. 2017. “9 Top JavaScript Frameworks For Mobile App Development.” *Valuecoders -Expert Remote Teams for your Web & Mobile Needs*. https://www.valuecoders.com/blog/technology-and-apps/top-javascript-frameworks-for-mobile-app-development/.

Y. Ko, H. Lee, J. Dolby, in S. Ryu. 2015. “Practically Tunable Static Analysis Framework for Large-Scale JavaScript Applications.” *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*: 541–51.