An Empirical Study on Evolution of Open Source JavaScript pr Projects

Everton da S. Maldonado and Shahriar Rostami Dovom
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
everton.maldonado@gmail.com, shahriar.rostami@gmail.com

Abstract*—*[Shahriar: hello maldonado]

I. Introduction

JavaScript is object-oriented to its core, with powerful, flexible high level programming capabilities. This language also supports functional and imperative programming styles. It is ubiquitous, it is fast and getting faster as compare to other web programming languages. Developers can craft it manually or they can target it by compiling from another language [1] [2]. It has been few years since JavaScript [3] is competing with other server side languages like (PHP, Ruby and etc.)

Source code analysis in object oriented and generally statically typed languages has been the interest of researchers for decades. However measuring object oriented metrics in JavaScript is scarce [1] [2].

II. Related Work

[Everton: use this paragraph in the related work for JavaScript. Note that it was just measuring metrics not evolution. but we have to mention it: Starts here: Previous works on JavaScript calculate trivial metrics such as number of attributes (NOA), number of methods (NOM), Depth of Inheritance Tree (DIT) and number of children. Possible actions can be taken for: (1) recalculating and evaluating those metrics, (2) calculating advanced metric complexity of a function or file.]

III. Approach

We decide to examine the evolution of five JavaScript server-side packages known as Node Packages and five Java projects to compare how might the similarities and differences on how these two popular programming language can affect evolution of software systems. We want to measure various aspects of the growth of these applications by having metrics such as number of files, lines of code, number of functions and statements. We also measure amount of duplications known as clones in terms of lines of codes, blocks and files. We measure the cyclomatic complexity over time which the metric is calculated as following. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. The control flow can split by conditional statements like if/else, switch case and so on. This metric is also known as also known as McCabe metric We use the term source file to mean any file whose name ends with .js and also we removed folders containing external libraries which is usually located at *lib* or *node_modules*.

[1] TypeScript: http://www.typescriptlang.org/Tutorial
[2] CoffeeScript: http://coffeescript.org/
[3] NodeJS: http://nodejs.org/

IV. Research Questions and techniques

Lehman suggests using the number of modules as the best way to measure the size of a large software system [3]. However, we decided to use the number of uncommented lines of code (uncommented LOC) like the way Godfrey et al [4] did the evolution study on Linux Kernel. On the other hand we measure the comment lines and the ratio of comments to lines of codes, and based on that we can infer how much developers tend to put comments within their codes. We have to consider hidden corners that can mislead results, for example descriptive comments are totally different to the lines of codes that got commented because of refactoring or changes which consider as light-weight code smells within the code. We are going to answer following research questions:

V. Research Questions

- How and in what extent can we measure object oriented metrics in JavaScript?
- By admitting the fact that JavaScript developers emulate object oriented style programming, we can empirically compare the differences between this language and mainstream programming languages.
- Studying the evolution of JavaScript libraries in terms of software metrics to find out how mature developers are writing code in terms of obeying object oriented best practices.

VI. Milestones

References

[1] G. Richards, S. Lebresne, B. Burg, and J. Vitek, "An analysis of the dynamic behavior of javascript programs," *SIGPLAN Not.*, vol. 45, no. 6, pp. 1–12, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1809028.1806598

[2] W. Gama, M. Alalfi, J. Cordy, and T. Dean, "Normalizing object-oriented class styles in javascript," in *Web Systems Evolution (WSE), 2012 14th IEEE International Symposium on*, Sept 2012, pp. 79–83.

[3] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski, "Metrics and laws of software evolution-the nineties view," in *Software Metrics Symposium, 1997. Proceedings., Fourth International*, Nov 1997, pp. 20–32.

TABLE I
PROPOSED EXPERIMENT PROJECTS WITH PRELIMINARY RESULTS OF LATEST RELEASE

Name	Description	# of JS files in latest release	Number of Directories	Number of Functions	Number of Statements
NPM	Google's common JavaScript library	165	32	1217	5329
Node MySQL	RequireJS is a JavaScript file and module loader	140	11	667	3317
Esprima	ECMAScript parsing infrastructure for multipurpose analysis	34	6	4862	29002
Grunt	HTML enhanced for web apps	31	9	251	1245
Node Redis	Google chrome encryption extension implemented in JavaScript	18	6	457	2537

[4] M. Godfrey and Q. Tu, "Evolution in open source software: a case study," in *Software Maintenance, 2000. Proceedings. International Conference on*, 2000, pp. 131–142.