AJJSME

Advanced Java JavaScript Metrics Extractor
Final project report

Anastasia Trimasova
Vyacheslav Bikbaev

April 2017

1 Problem statement

Software metrics play an important role in the software development area. Correct metrics allow to make the process of the software development more suitable for the current needs of an organization. Therefore, many authors try to evaluate the shortcomings of existing metrics and the need for new metrics especially designed for object-orientation. [3]
The goal of this project is to extract metrics from a JavaScript code.
The whole project was divided into three parts:

- The first part of the project was devoted to acquaintance with Java, the study of code metrics and thinking over the design of the solution. Also, we had to find or implement JavaCC grammar for our language - JavaScript, write Unit Tests for a few metrics and try to implement them.
- In the second part of the project it was necessary to write a sufficient number of Unit Tests for a metrics extraction, implement these metrics extraction, write statistics output to a file and test our extractor on files with real JavaScript code.
- And the final part is to finalize our solution, test it on a rather big data set of source code and present our results.

2 System specification

During our project development the following technologies have been used:

- JavaCC 6.0
- Java 8
- JavaScript ES5
- GitHub
- IntelliJ IDEA

3 First phase

In the first phase we have spent our time on reading some articles and sites about metrics, JavaCC and its usage [6, 7, 5, 2, 4, 1]. As JavaSript is a very popular programming language, we first tried to find existing implementation of a grammar to use in our implementation. Fortunately, we have found several implementations. After we've revised and tested them, we've pick one for our use. It is one of the newest versions of JavaScript - EcmaScript 5. (Names JavaScript and EcmaScript are used interchangeably).

Link: EcmaScript 5 grammar

Later, we have written a simple wrapper class to play with parser and understand, how it works. Also, we decided to write several Unit Tests to check if the grammar works correctly.

Further we read articles about metrics, in particular for not object oriented programing languages, to decide which ones can be applied in case of JavaScript. Then we implemented unit tests for some of them.

4 Second phase

In this phase we've implemented more methods to extract metrics and now we have 8 metrics that characterize code written in JavaScript.

We've made **AdvancedJavaMetrics.jar** file to run the parser without any redundant movements. Our metrics parser reads ***.js** files available in particular directory and outputs metrics in **metrics.csv** file.

File **metrics.csv** contains metrics represented as columns and ***.js** files represented as rows. At the intersection of the file and the metric there is counted value for it.

4.1 Metrics

Some of the metrics calculated recursively because of the nested functions.

1. linesOfCode
2. numberOfComments
3. numberOfGlobalVars

   counts number of global variables
4. numberOfVars

   counts number of all variables
5. numberOfFunctions
6. averageFunctionSize
7. numberOfLoops

   counts *for, foreach, while, do..while* statements
8. numberOfConditions

   counts *if-else* statements

Link: archive with metrics pareser

In zip-archive, we provide metrics extractor **AdvancedJavaMetrics.jar** and folder **js** with some examples of JS files to test our metrics extractor with.

5 Third phase

In the third phase we have tested our metrics extractor on real data set, which contains JavaScript source code. Source code is taken from popular JavaScript libraries, namely Vue.js and jQuery. Because these files are prepared for web usage, we have modified them to allow metrics parser extractor to work with them. It was an important step but it has no side impact on the performance of our program. Each file contains about 10000 lines of code.

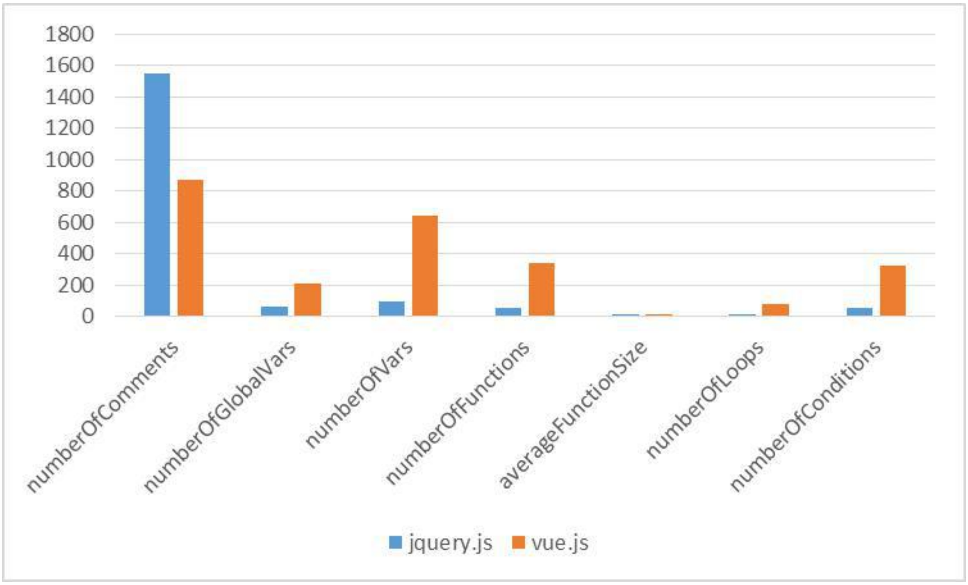

Figure 1: Visualized metrics

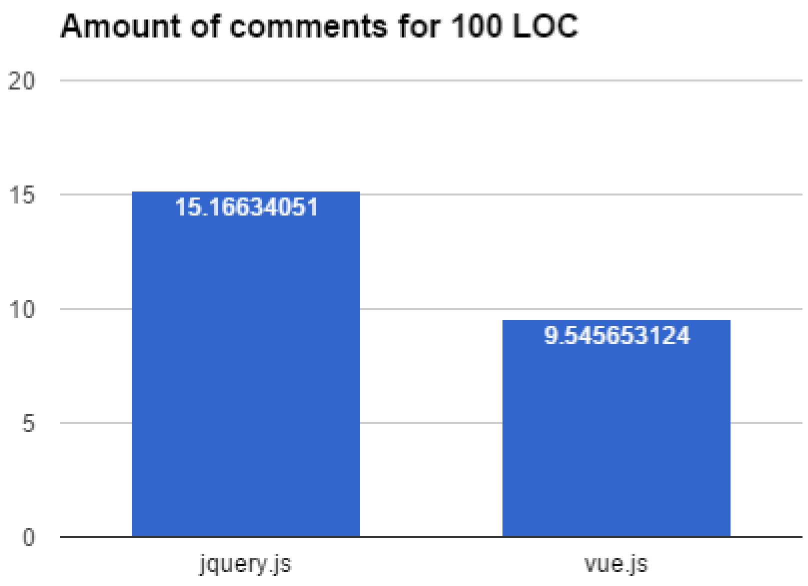

Figure 2: Comments

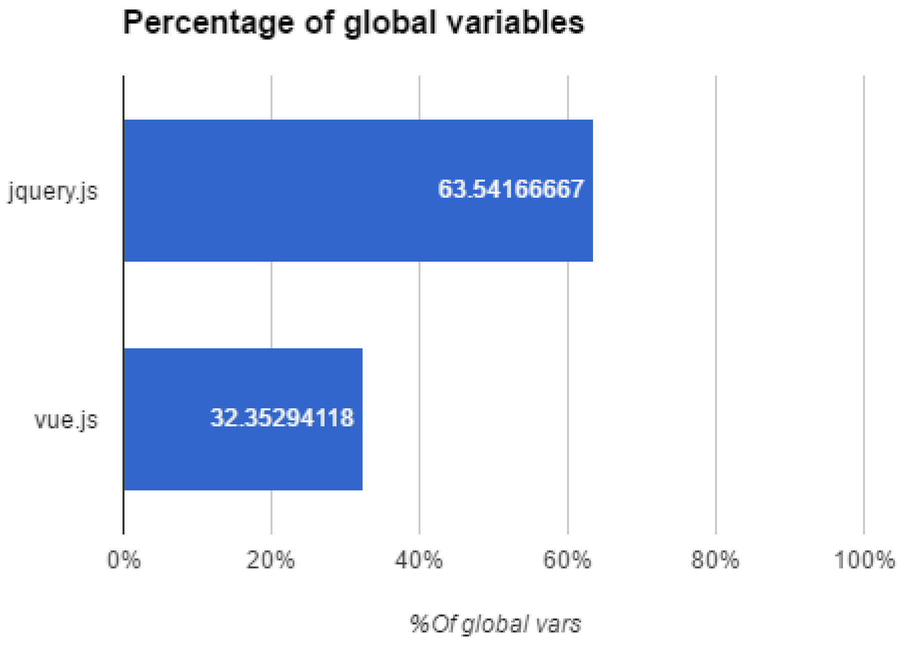

Figure 3: Global variables

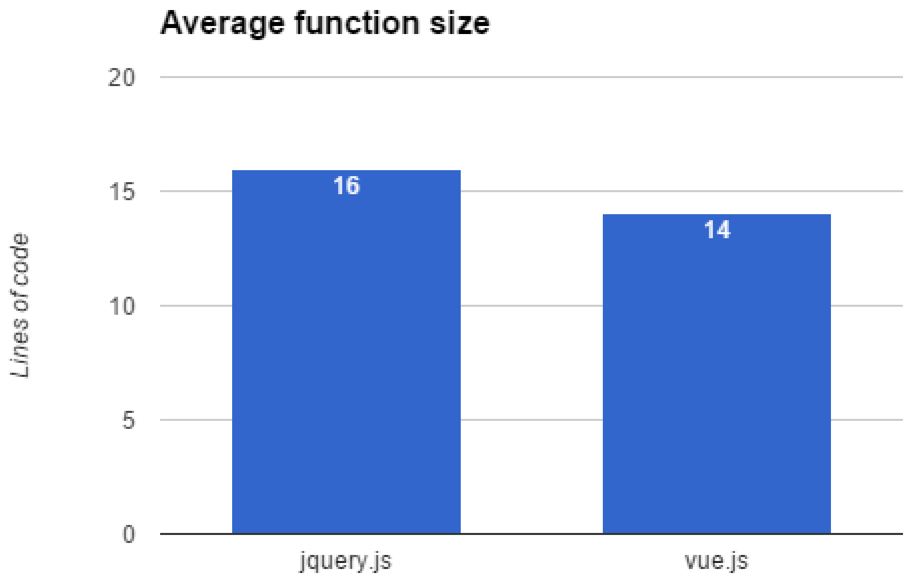

Figure 4: Function size

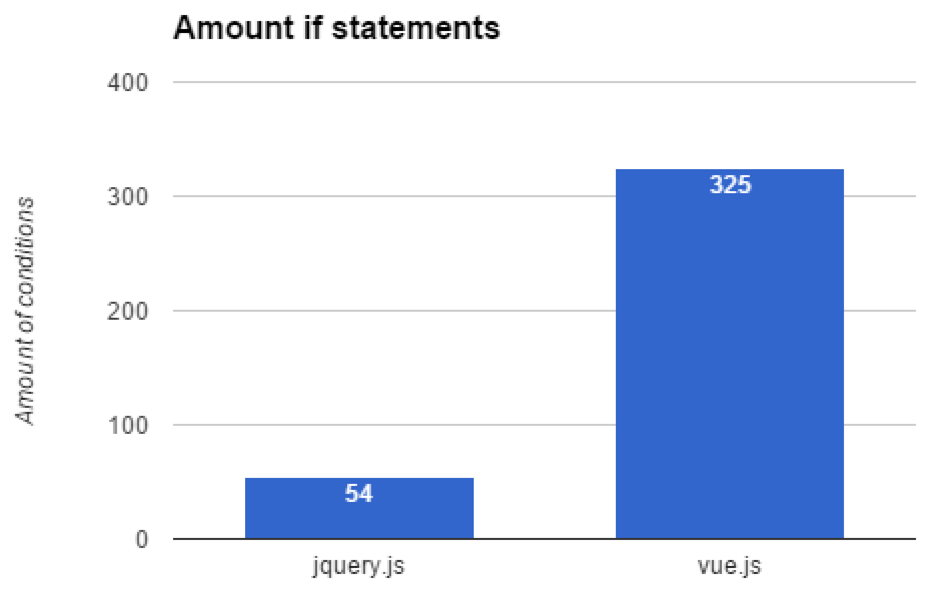

Figure 5: If statements

From these analytics we can judge that jQuery is more documented than Vue.js. Also, we can see that average function size is virtually the same, but in jQuery the chance of error is slightly lower. However, the amount of global variables in jQuery is a little bit larger, and this is bad for security. Finally, there are more IF statements in Vue.js, which statically increases chance of mistake.

6 Software launching

Link: archive with metrics parser

In zip-archive, we provide metrics extractor **AdvancedJavaMetrics.jar** and folder **js** with source code files to test our metrics extractor with.

To run our program (AdvancedJavaMetrics.jar file), type in your console

```
java␣-jar␣AdvancedJavaMetrics.jar␣"/path/to/folder/with/sources"
```

7 Github

Link: https://github.com/Alcovaria/AdvancedJavaMetrics

References

[1] An Introduction to JavaCC. https://www.codeproject.com/Articles/35748/An-Introduction-to-JavaCC.

[2] The JavaCC Tutorial. http://www.engr.mun.ca/~theo/JavaCC-Tutorial.

[3] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[4] Viswanathan Kodaganallur. Incorporating language processing into java applications: A javacc tutorial. *IEEE software*, 21(4):70–77, 2004.

[5] Michele Marchesi, Giancarlo Succi, and Williams Laurie. Traditional and agile software engineering, chapter 24 – measuring development.

[6] Sanjay Misra and Ferid Cafer. Estimating quality of javascript. *The International Arab Journal of Information Technology*, 9(6), 2012.

[7] Di Wu, Lin Chen, Yuming Zhou, and Baowen Xu. A metrics-based comparative study on object-oriented programming languages. In *SEKE*, pages 272–277, 2015.