Performance and Quality Evaluation of jQuery Javascript Framework

Andreas Gizas, Sotiris P. Christodoulou, Tzanetos Pomonis

HPCLab, Computer Engineering & Informatics Dept., University of Patras Rion, Patras

Received Jun 10, 2013; Revised Jun 21, 2013; Accepted Mar 12, 2014; Published Jun 12, 2014

Abstract

The scope of this work is to provide a thorough methodology for quality and performance evaluation of the most popular JavaScript framework, the jQuery Framework, by taking into account well established software quality factors and performance tests. The JavaScript programming language is widely used for web programming and increasingly, for general purpose of computing. Since the growth of its popularity and the beginning of web 2.0 era, many JavaScript frameworks have become available for programming rich client-side interactions in web applications. The jQuery project and its community serve today as a major part of web programmers. The main outcome of this work is to highlight the pros and cons of jQuery in various areas of interest and signify which and where the weak points of its code are.

Keywords

JavaScript; Frameworks; Metrics; Comparison; jQuery; Quality

Introduction

The most popular programming language for the browser today is JavaScript (Chuan and Haining, 2009). JavaScript was specifically designed to run in a Web browser and, thus, JavaScript is the most suitable for the development of client-side programs. In particular, JavaScript is well-suited for programming event-based user interfaces in the Web browser. All main-stream Web browsers today support the execution of JavaScript natively. Along with the growth of demands for more comprehensive user interfaces, the size and the complexity of web applications are increasing. On the other hand, JavaScript is also becoming a general purpose computing platform for browsers (Richards, et al, 2010), for office applications, for RIA frameworks (like Google Web Toolkit, Qooxdoo.org, Cappuccino.org) and even program development environments (like lively-kernel.org). In the last few years, the Web was becoming more accessible by portable and wireless devices. Mobile web is the name of this new field of web applications and JavaScript is expected to play a major role in its development with the evolution of new devices and standards (ex. iPhone, Android) or as the heart of cross platform applications (like phonegap.com). There are also proposals for employing JavaScript in server-side applications (Server-Side JavaScript Reference v1.2).

Due to the plethora of applications that JavaScript serves and the variety of programming needs, frameworks have been created in order to help both programmers and end-users. These frameworks aim to be a useful tool for simplifying JavaScript code development and repeat blocks of code by using just a few lines and symbols. Moreover they provide clean structure, new features, cross-browser support, pre-build applications and plug-ins ready for use. For these reasons, JavaScript libraries and/or frameworks, have become most popular (The State of Web Development, 2010), and gain more and more friends nowadays. The diversity of web applications as far as the functionalities that web 2.0 and Ajax propose, the end- user GUI, the different data collections and repositories, make these frameworks a useful tool for creating rich web applications. As JavaScript language is used for more ambitious projects, frameworks to support these projects become increasingly important. Today, the most popular JavaScript frameworks (called JFs from now on) for creating applications for browsers are these six: jQuery, Prototype (with Scriptaculous library support), Dojo, MooTools, ExtJS and YUI. Nowadays jQuery serves over than 60% of JavaScript framework users and is by far, the most well known and used framework in this category. Especially with the "War of browsers" that have arouse, browsers develop continuously. The end-user demands change (html5, video on demand, Ajax etc.), so these libraries have to develop again and again. Evolving of technologies, different demands of users,

new needs for new functionalities, many operating systems and the most distinctive mobile web make the choice difficult for choosing the appropriate framework. Towards this direction, are we sure that jQuery is the best choice for our needs? Does it follow the continuous evolvement of web technologies or needs some more additions?

Mainly, we focus on the code quality of the basic library of JavaScript frameworks and we conduct a static analysis research by using different metrics each time. Quality is being a basic factor for consideration in the case of these frameworks, especially as they are being used to everyday projects nowadays. For the first time these JavaScript frameworks are being tested from the quality point of view, trying to find which of them are mature enough for production of big projects and satisfy the criteria of portability, maintainability, reusability and testability (Calero, et al, 2004). Second, we measure the performance of these frameworks and try to analyze the basic factors that may affect it. The most popular combinations of operating systems, browsers, access devices (pc stations, embedded devices, mobile) are taken into account in order to measure size of data exchanged, speed of Dom Access. Third, we conduct validation tests and we look for security vulnerabilities for the framework. These tests highlight pros and cons in every case. So, this paper proposes new additions and probable improvements for the core libraries. It tries to summarize the work that is being done in the area of JavaScript frameworks and we conclude with the useful facts on what library we should use in every case.

In the following section, some related work will be presented, first regarding the quality metrics of software and web applications on which our analysis is based. Second, we will mention the efforts which are being done in the performance analysis of these frameworks, and we will present related works that have been done. Chapter three starts with an overview of definitions as far as the terms and technologies for each framework is concerned and summarizes how frameworks have evolved until today. In this chapter we refer to jQuery more thoroughly, and discuss some key characteristics of JavaScript programming that have made it so special. Then, chapter four presents our methodology and the basic axes of our analytical comparison based on previous efforts in the area of software and web analysis and metrics. The next section provides our measurements of several metrics regarding the jQuery frameworks. Furthermore we analyze the results; we outline the most important outcomes and discuss them. In Chapter six, we propose some new metrics for in order to estimate the quality of jQuery selectors, which we believe to be the heart of jQuery framework. The paper is concluded with some intended future work.

Related Work

In this section we discuss research efforts related directly or indirectly to our work. **Software quality** is a major research area and there is substantial literature on **quality metrics** and on methods to assess the quality of given software. Numerous studies confirm that many software metrics aggregated in software quality prediction models and testing methodologies, are valid predictors for quality characteristics of general interest like maintainability and correctness. There are plenty of papers in software quality research field, starting from the pioneer works of (Kafur et al. 1985, LiLi and Cheung, 1987, Mata-Toledo and Gustafson, 1992), which suggest quality metrics or even approaches to improve software development in general. Such metrics, or combination of them, measure certain aspects of code and they can be collected and used throughout the life cycle of software in order to give us the ability to predict or detect poor software quality. Web engineering is a software-related field, thus these metrics can also be employed to measure the quality of web software like JavaScript code and of course JavaScript frameworks.

There are a plethora of quality metrics that have been proposed and evolved during time. Some of them are obsolete or some others have specific weight, depending on what we want to measure (IEEE Std. 1061-1998, Kaur, et al, 2009). **Choosing the right metrics** is crucial in order to assess accurately. The authors in (Barkmann, et al, 2009) studies and evaluates several software quality metrics and finds correlation among them, indicating that some of them seem to measure the same properties. Moreover, other researchers (Calero, et al, 2004, Gayatri, et al, 2009) suggest that depending on the category of software, we should choose different quality tests, in order to improve the efficiency of metrics collected. Furthermore, some researchers (Olsina and Rossi 2002, Jiang, et al, 2008) focus on the analysis of **performance and quality of web applications**. We base our own methodology on such outcomes, in order to choose the right metrics relevant to JavaScript programming

language and framework development.

In the literature we can find some more generic efforts on comparison, evaluation and performance testing of programming languages like PHP and JSP (Trent et. al. 2008). As far as **JavaScript performance** testing is concerned, the author in (Ratanaworabhan et al., 2010) deals with it, giving us an evaluation of benchmarks that should be used. Especially for **JavaScript frameworks** comparing, some efforts have been made (Compare JavaScript frameworks, 2010, Crandall, 2006, Evaluation of JavaScript Libraries, and Rosales-Morales, et al, 2011) but none is focused on the software quality factors of the frameworks. None of these works are based on previous scientific analysis and there are not up to date.

Quality Metrics

As stated before, the aim of this work is to specify the quality metrics that are important to collect when we consider open source JavaScript Frameworks, and measure them for jQuery. To achieve that, we conduct different quality, performance and validation tests. As far as quality is concerned, we measure the size, the complexity and the maintainability of code as a whole and per function.

The **Size Metrics** are a number of metrics attempt to quantify software "size"; how big the program or function is. The most representative size metric is *lines of code (LOC)*. The bigger the number of LOC is, the harder to find and correct errors. Additional metrics here are the *number of statements*, the *comment lines* and the *ratio* between comment lines and lines of code. This ratio should be at least 10% and can indicate a function that is poorly commented and thus hard to understand, maintain and evolve especially within an open source community with thousands of diverse developers.

The **Complexity Metrics** measure the difficulty to understand the code and find a logical error within it. The most important metric in this category is *McCabe's cyclomatic complexity*, which measures the number of linearly-independent paths through a program module. It is one of the most widely-accepted software metrics; it is intended to be independent of language and language format. *Branches* (the number of paths in logical flow through the execution of a function) and *depth* (the maximum "nesting" of if-then-else and loop structures within a function) are important complexity factors of a function and indicate part of code which can be better re-programmed. Branches are caused by the use of if, else, case, default, catch, finally, && and ||. The Cyclomatic Complexity (C) we measure here is one more than the number of edges formed by a branch (E), minus the number of branch nodes (N), plus the number of exit points in the function (R) (return or throw).(C = E - N + R + 1). Finally, a complexity metric for the whole software is its *number of modules or functions*.

The **Maintainability Metrics** are used primarily to determine whether the program or a function has a high, medium or low degree of difficulty to maintain. (Halstead and Maurice, 1977) Halstead metrics (Program Volume and Program Level were selected) and Maintainability Index are popular metrics in this category. MI is a single-number value for estimating the relative maintainability of the code. Maintainability Index is calculated with certain formulae from lines-of-code measures, McCabe measure and Halstead measures. (Oman and Hagemeister, 1992) The measurement and track maintainability are used primarily to determine if code has a high, medium or low degree of difficulty to maintain, and to indicate when it becomes cheaper and/or less risky to rewrite the code instead to change it. Maintainability Index is calculated on each function, on each file and on all files (if they exist) together level.

JS FRAMEWORKS

Definition and Overview

So, what do JavaScript Frameworks especially offer and have become so popular for programming for web browsers? The key principle behind many frameworks is DRY principle - Don't Repeat Yourself. They aim to reduce code duplication and instead provide you with a set of tools you can build upon that are known to be stable and robust and tested. This is the biggest benefit of JFs that facilitates programmers to produce applications quickly and easily. They typically provide a library of classes that will do a multitude of things from managing DOM traversal and manipulation, effects, Ajax manipulation, layout, at the same time impose an architecture that provides an unformed way to extend the framework (e.g. plug-ins, modules etc.). They also aim to smooth over the functional differences between web browsers, thus prevent the web developer from having to write the same browser specific code over and over again. They reduce the amount of code that

the programmer writes and make the classical orders of JavaScript shorter and simpler to write and understand. Also, they cover a wide range of different applications and therefore are subject to use by many developers.

They reduce the amount of code that the programmer writes and make the classical JavaScript instructions shorter and simpler to write and understand. We summarize the most basic functionalities of these frameworks:

1. *DOM Traversal and Manipulation*: Select DOM elements, search elements by ID or class name, find element's position, parent or child elements etc. Manipulations of DOM elements like hide or remove an element, add elements, copy them, change properties such as color, width, height, etc.

2. *Selectors Engines*: Most of the available JF implement a mechanism for rapid element selection. These selectors make the process of obtaining a reference to an HTML element much faster, and allow you to search for the element by ID, class name, element type, or by using a series of pseudo-selectors.

3. *Event handling*: Each JavaScript framework implements cross-browser event handling support that encourages you to move from the old-style inline attaching of events to a streamlined method. Now, in one line we can add a sequence of events that one follows the other. Basic event handling is supported by all JFs, while most of them provide some extra event functions.

4. *Ajax support*: One of the most compelling reasons to use a JavaScript framework is for standardized cross-browser Ajax requests.

5. *Compatibility*: JFs functions guarantee the compatibility of your code in every OS platform, version of browser, and type of device, thus the developer doesn't have to worry about such issues.

6. *Utility functions*: Many JFs come with a set of various utility functions that make developing JavaScript applications easier and quicker, especially those that focus on rich user interfaces. Utilities functions can be categorized as UI Widgets, visual effects and other utilities. Visual effects contain functions which deal with animations, positioning, resize and move. Typical UI Widgets include menus, buttons, progress bars, auto complete snippets, grids, tree views and tab views of information. Other utilities commonly used, are Cookies manipulation, date picker, rich editors, localization for different languages and dialects, Drag & Drop functionalities, history support, JSON support, etc. In most frameworks we can even add plug-ins to additional UI utility functions.

Overall, by using the term JavaScript Framework we end to mean a basic core implementation of pre-written JavaScript code (many times supported with some more Libraries for extra functionality) which allows for easier development of JavaScript-based applications, aiming to be robust, with cross browser functionality especially for web-centric technologies like Ajax, Mobile Web etc.

There are plenty of JS Frameworks. The most popular are: *Dojo, ExtJS, MooTools, YUI, Prototype* and of course *jQuery*. Most of them come with a minified and/or compact version (minimum size, no comments) and a development version. Some of them (jQuery and ExtJS) have mobile versions with special mobile UI functions, aiming to speeding mobile accessibility. In each framework we can distinguish the basic part of code, that we call the core and a list of addons (called plugins), which come to enhance the basic functionalities of the core. The core usually contains functions for selectors, Dom manipulation, Ajax functionalities, implementations of effects and animations and basic elements of forms. Additionally, plugins deal with interactions with other programs, theming, new functionalities and widgets. Of course these things can differ from version to version and between frameworks. YUI has the idea of add-ons more basic. The core is smaller. ExtJS and Dojo come with full packages of addons when you download them, especially for those who want to develop big projects. The jQuery UI is the basic add-on package for interface, theming and effects for jQuery. For Prototype, we have the Scriptaculous library, which offers new functionalities, effects and animations. MooTools More is the basic addon library for MooTools. Additionally, many widgets and pluggins exist in every framework in order that the user can enhance their functionalities.

A Closer Look to Jquery

JQuery started in August of 2005, when John Resig first made mention of an improvement on Prototype's "Behavior" library. This new framework was formally released as jQuery on January 14, 2006. The jQuery library provides a general-purpose abstraction layer for common web scripting, and is therefore useful in

almost every scripting situation. As mentioned above, it gathers all the basic characteristics of JavaScript frameworks. It is here to make DOM traversal and manipulation very easy, simplifies the modification of appearance of web pages (supports CSS 1-3 selectors), and alters the content of a document (with a single easy-to-use API). The jQuery library also offers an elegant way to intercept a wide variety of events, it animate changes being made to a document with beautiful effects added to the core lib, it makes uses of AJAX and in addition it provides enhancements to basic JavaScript constructs such as iteration and array manipulation.

But why jQuery is so famous today? By basing the mechanism for locating page elements on **CSS selectors**, and the **support of extensions are the two basic characteristics** that have made this library popular. The jQuery relegates special-case uses to **plugins**. The method for creating new plugins is simple and well-documented, which has spurred the development of a wide variety of inventive and useful modules. Even most of the features in the basic jQuery download are internally realized through the plugin architecture, and can be removed if desired, yielding an even smaller library. Today the final user can search among thousands of plugins which may serve his needs. The diversity is wide.

Another feature that we underline is that the programmer is allowed to have multiple actions in one line. To avoid overuse of temporary variables or wasteful repetition, jQuery employs a programming pattern called **chaining** for the majority of its methods. This means that the result of most operations on an object is the object itself, ready for the next action to be applied to it. In addition, when we instruct jQuery, for example "*Find all elements with the class* example *and hide them*", there is no need to loop through each returned element. Instead, methods such as .hide () are designed to automatically work on sets of objects instead of individual ones. This technique, called **implicit iteration**, means that many looping constructs become unnecessary, shortening code considerably.

The vibrant community that has sprung up around the project is the actual life of the library. Users of jQuery gather to discuss not only the development of plugins, but also enhancements to the core library. The library has its own mobile version (which has reached version 1.3) and many plugins have aroused. Microsoft, Nokia and Adobe have adopted jQuery as part of their official application development platforms. Facebook and Twitter are using extensively JavaScript and especially jQuery. The facts above point out how hot jQuery is and in how many different categories of interest can be adopted.

The Testing and Evaluation Process

Quality Tests

In order to measure the quality metrics, we used the following tools: Cloc (cloc.sourceforge.net), Jsmeter (jsmeter.info), and Understand (scitools.com). Some metrics were measured by more than one tool in order to validate the measurement.

Cloc counts lines of code in files, comment lines, comment blank lines and physical lines of source code. Given two versions of a code, Cloc can also compute changes in blank, comment and source lines. It is written entirely in Perl and is licensed under the GNU. It is used to collect the size metrics.

The **JsMeter** software is a JavaScript application licensed under New BSD License that calculates code metrics for JavaScript programs. The following metrics are collected with this tool: statement count, branch count, cyclomatic complexity, lines of code, lines of comments, % of lines containing comments, code depth, Halstead Program Volume / Level and Maintainability Index.

Understand is a static analysis tool for maintaining, measuring and analyzing JavaScript Code among others. We use it to calculate both Complexity Metrics and Size Metrics.

TABLE 1 SIZE METRICS

Frameworks	Line	Comment Lines	Statments	% Comments
jQuery 1.9.0	9555	1290	7124	13,50%
jQuery 1.8.3	9472	1272	7199	13,43
jQuery 1.7.2	9301	1242	7252	13,35%
jQuery 1.6.2	8981	1160	7022	12,92%
jQuery 1.6.1	8936	1152	6997	12,89%
jQuery 1.5.1	6490	1073	6490	12,9%
jQuery 1.4.1	6078	671	5168	11,04%

Table 1 summarizes the Size Metrics for jQuery Framework and for the 4 last versions. Our purpose for that is to observe how the metrics vary from version to version. We examine only the core of the framework and of course we use the non-minified version.

Table 2 summarizes the Complexity Metrics for

jQuery.

TABLE 2 COMPLEXITY METRICS

Framework	Depth	Branches	Number of Functions	Cyclomatic	Sum Cyclomatic
jQuery 1.9.0	9	471	402	600	1837
jQuery 1.8.3	9	459	400	572	1811
jQuery 1.7.2	9	385	400	474	1803
jQuery 1.6.2	9	351	508	431	1721
jQuery 1.6.1	9	339	507	420	1719
jQuery 1.5.1	9	306	462	379	1614
jQuery 1.4.1	9	174	403	242	1337

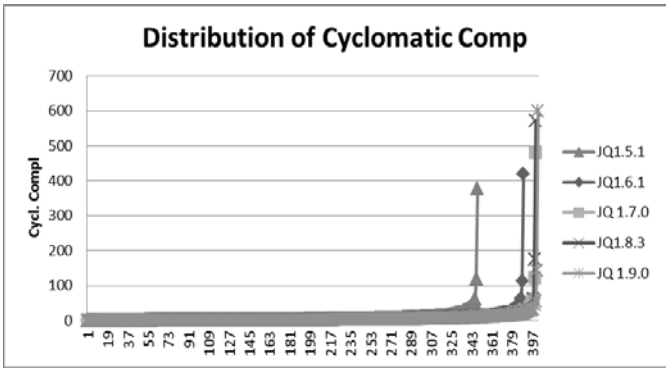

FIGURE 2 DISTRIBUTION OF CYCLOMATIC COMPLEXITY FOR JQ VERSIONS

Figure 2 shows the functions' distribution of cyclomatic complexity for the five last major versions of jQuery. These figure illustrates how bigger CC becomes as versions grow but as we demonstrate the number of functions is almost the same from 1.7 and later.

Moreover, considering that10 is a threshold for cyclomatic complexity of the functions of the framework, we give a list of 49 functions that have CC more than 10. Especially those that have more than 20 CC must be checked. These functions are considered difficult to maintain. The full list of the worst functions as far as Cyclomatic Complexity is provided, with the line on which they begin.

TABLE 3 HALSTEAD AND MAINTAINABILITY METRICS

Frameworks	Program Volume	Program Level	Main. Index
jQuery 1.9.0	6880	0,00169	1293,8
jQuery 1.8.3	6920	0,00168	1276,1
jQuery 1.7.2	5730	0,00202	1146,5
jQuery 1.6.2	5160	0.00225	1166.1
jQuery 1.6.1	5090	0.00228	1158.2
jQuery 1.5.1	4440	0.00261	1081.2
jQuery 1.4.1	2850	0.00407	688.30

The Table 3 summarizes the Halstead and Maintainability Metrics of each version of jQuery. The measures were taken thanks to JsMeter. Moreover Table 4 summarizes those problematic functions that exceed the ideal thresholds for cyclomatic complexity, depth and MI for the last versions of jQuery. For each version we have a detailed list with these problematic functions in order that the community have a closer look into specific parts of code that we consider will cause problems in future usage of the framework.

TABLE 4 NUMBER OF FUNCTIONS WITH PROBLEMATIC CYCLOMATIC COMPLEXITY, DEPTH AND MI

Framework	Cyclomatic Complexity (CC)			Depth	Maintenance Index (MI)		
	11-1!	16-2(	20+	>5	<100	<85	<65
jQuery 1.9.0	25	11	12	6	53	7	0
jQuery 1.8.3	24	12	12	4	49	10	0
jQuery 1.7.2	26	10	18	6	55	14	1

Considering that 85 is a threshold for maintainability index of the functions of the framework, we give a list of *all functions* that have MI less than 85. These functions are considered difficult to maintain. The full list of the worst functions as far as Maintainability Index is provided, with the line on which they begin.

For example (Anonymous1).(Anonymous17) .Sizzle. filter function (starts in line 3965) has 24 CC, 69,129 MI! (JQ1.6.2). Or (Anonymous1).domManip (in JQ 19.0) has 25 CC, 85 Statements and 74 MI. Those functions that appear in both lists are the most crucial ones for new check. We propose 60 functions for jQuery 1.9.0, 59 for 1.8.3 and 70 for jQ 1.7.2!

The full details and measurements can be found in our web server http://150.140.142.212:100/JFmetrics/jq.html.

We underline the effort of jQuery foundation to produce code with more functionality and less volume as versions grow. More effort should be done in the future.

Furthermore, we examine a relation of bug history of jQuery with the list of functions we propose in this work. In jQuery's bug tracker (http://bugs.jquery.com) a user can find or upload a bug for a specific version. Sizzle function for example, is a function that has big CC and MI in all jQuery versions and we can find many tickets referring to it. We argue here that our proposed methodology for finding problematic functions of a specific version might be useful for predicting and solving future issues in any JF. More

tests will be conducted in this area.

Validation Tests

The **validation tests** have been made by using Yasca (sourceforge.net/projects/yasca) software utility. It is an open source program which looks for security vulnerabilities, code-quality, performance and conformance to core DOM. We moreover incorporate JavaScript Lint into Yasca and run the tests accordingly. Table 4 summarizes the errors found for every version of jQuery.

TABLE 5 VULNERABILITY AND CONFORMANCE OF JQUERY FRAMEWORK

Frameworks	Critical Severity Errors	High Severity Errors	Overall
jQuery 1.9.0	**33**	**35**	**1210**
jQuery 1.8.3	**30**	**31**	**1191**
jQuery 1.7.0	**29**	**28**	**1124**
jQuery 1.6.2	**29**	**28**	**1109**
jQuery 1.6.1	**28**	**28**	**1107**
jQuery 1.5.1	**19**	**29**	**1095**
jQuery 1.4.1	**13**	**17**	**839**

The complete list can be found in our server. We provide all error points, the exact source code position and explanation of why they are being spotted. We notice that the number of errors increases from version to version. Nevertheless, although jQuery adds more functions and statements to latest versions, the critical errors remain almost the same.

This is a notice for consideration. In version 1.6.2, we have 10 critical errors from IE incompatibility, 7 errors come from Firefox incompatibilities, 4 of them include Firefox, Safari and Chrome and 7 come from older versions of browsers. The four critical mistakes that come from incompatibilities of Firefox, Safari and Chrome are considered general and have to be solved. These are the errors with code 12 (line 4661), 13 (4662) which are in the function called unnamed_function_195. This function is between lines 3738-5150! The other two mistakes with number 25(line 5877) and number 26(line 5883) belong to cloneFixAttributes function. Moreover, the first 8 errors (codes 1-8) are due to the command setAttribute, which is not fully supported by IE.

Performance Tests

In order to conduct **the performance tests**, we used the SlickSpeed Selectors test framework, which is open source (code.google.com/p/slickspeed/). First of all we had selected the most popular sets of settings that end-users have today (w3c.org and gs.statcounter.com). With the term "set of settings", we mean a specific regulation for a web browser, an operational system, and the device the user has. We tested the performance of each JF under 6 different browsers, 4 different operating systems (Win7, WinXP, WinVista, and Ubuntu) and 6 computers, with the following hardware configurations:

- PC1/Windows 7 Ultimate: Intel R Pentium 4 CPU 2.8GHz, 1.5GB RAM
- PC2/Windows 7 Ultimate: Intel Core 2 Duo TB50@1.83 GHz, 2GB RAM
- PC3/Ubuntu 11: AMD Athlon 64 4000+@2.46 GHz, 2GB RAM with Ubuntu 11.04 Desktop Edition OS
- Laptop1/Windows Vista: Intel Core 2 Duo P8400@2.26 GHz, 2GB RAM,
- Laptop2/Windows XP SP3: AMD Turion 64X2 @ 1.66 GHz, 2GB RAM.
- Netbook1/Windows 7 Starter: Intel R Atom CPU N455 @ 1,66GHz, 1GB RAM.

Figure 1 illustrates the total execution times (the average among five separate runs) for each JF, on each browser, on each OS. Before every test, cache memory was cleared. During the tests, no other services were running. The detailed test results for each run can be found on our web server. The version of jQuery tested was v1.7.2, which was the last version of jQuery at that point of time. The time numbers is in ms. In order to have the right results we conducted more than 1000 measurements with all the above OS, different browsers and JavaScript frameworks. Before every test, cache memory was cleared. During the tests, no other services were running. The detailed test results for each run can be found on our web server. The main conclusions on these performance results:

- In average ExtJS, Prototype and jQuery are slightly faster than the others.
- Opera 11 is 40% faster than Opera 10, due to the introduction of the new JavaScript Engine Carakan, who is 2,5 faster than the old one (Futhark).
- Opera is lightly fastest in overall, meaning that Carakan JavaScript engine is performing better than the others.
- FF7 is 10% faster than FF4, while FF10 is about 15% slower than FF7!
- Chrome 17 is 30% faster than Chrome 15.

- Based on performance tests on previous versions (in 2011) of the JFs, we observed clear improvements (from 10 to 60%) in the performance between JFs versions released the last year.
- IE9 decrease the execution time to 60% than IE8 in windows 7
- IE9 is slower than IE8 in vista about 50% for jQuery 1.6.1
- IE 9 is faster about 70% than IE8 for jQuery 1.5.1
- Chrome is the fastest in overall

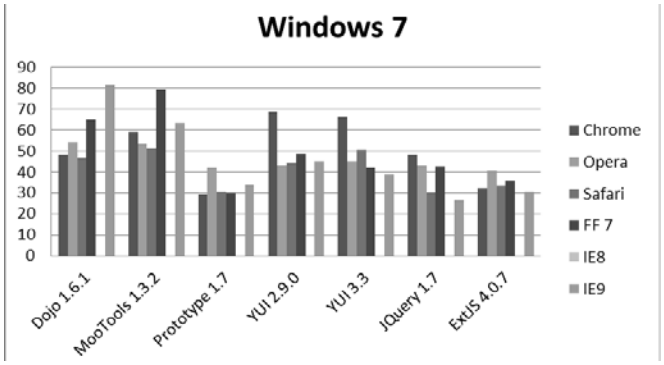

FIGURE 3 PERFORMANCE OF JQUERY 1.7.2 IN COMPARISON TO OTHER POPULAR JFS IN WINDOWS 7

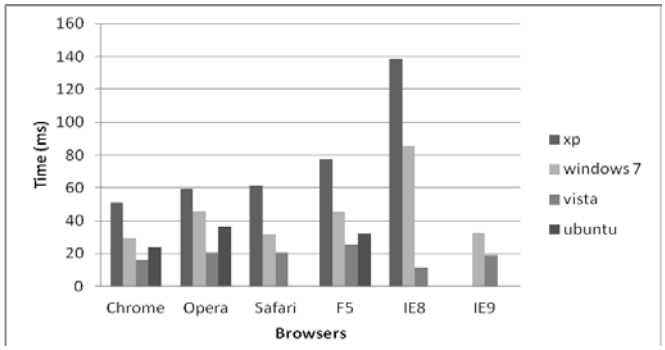

FIGURE 4 PERFORMANCE OF JQUERY 1.7.2 PER OS AND BROWSER

The results in Figure 4 offer comparison clues for the performance of jQuery Framework on different OS and different browser.

The results in Figure 4 show that in the same OS (windows vista), two different versions of jQuery have different performance. The newer version is much quicker than the older one, and this could be an indication for the development community of jQuery that probably it can be executed even faster. **The tests can be found in our server http://150.140.142.212:100/JFmetrics/index4.html.**

We also conducted these tests with mobile operational systems and we saw big latency and much bigger times. Simulators tests are not actual true as far as times and latencies are concerned. They only give a phenomenal outcome.(Figure 6) We used BlackBerry Smartphone Simulators and MobiOne Studio. We underline that we have to test the mobile versions of frameworks in order to see how they actually perform in mobile OS.

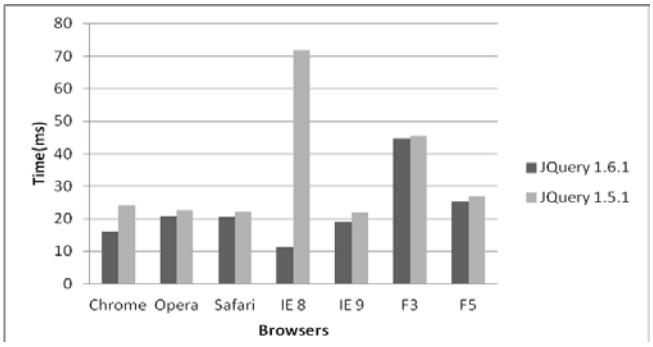

FIGURE 5 COMPARISON OF TWO JQ VERSIONS

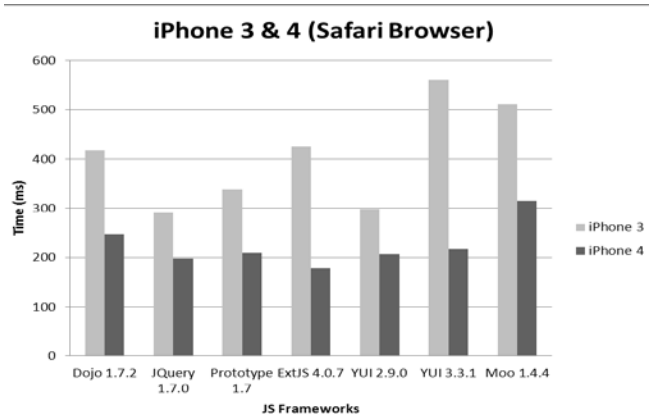

FIGURE 6 PERFORMANCE IN MOBILE DEVICES

Proposed Metrics

The above tests revealed to us that the heart of jQuery is the usage of selectors by the users. Many good practices are proposed to users in order to use selectors by ids' or classes. Jsperf (http://jsperf.com/) summarizes these tests for someone who likes to know how to start with jQuery. But how a user can finally test if a program needs more improvements and what parts can he examine in more details?

Except the preceded tests, we think that a developer must have more tools available in his hands. So, selectors are the big power of this JF and the way a user treats them reflects the quality and performance of the actual program. We examined in details:

- the syntax of jQuery programs
- the most commonly used statements and operators and, we propose some new metrics, especially selected for jQuery code. Thus, in this section we propose new facts that should be measured when we consider the quality of a jQuery code, many of which are not

measured by existing tools. It is obvious that these measurements are connected to the quality of the code. What is missing is to exhaustively measure several JQuery codes and try to specify the weak quality sections in code, and associate this weakness with specific thresholds on the values of metrics.

As the name implies, jQuery focuses on queries. The core of the library allows you to find DOM elements using CSS selector syntax and run methods on that collection. Typically, access to and manipulation of multiple DOM nodes begins with the $ function, being called with a CSS selector string, that uses node elements name and node elements attributes (id and class), as criteria to build selectors of HTML elements.

No tool measures anything about selectors. We strongly believe that some metrics about the volume, the variety and the complexity of the selectors could give worthy indications about the quality of jQuery code. In the scope of this paper we propose some metrics about selectors, based on our extensive experience on jQuery programming. We aim to evaluate and demonstrate their connection with quality characteristics in future work. These metrics are:

- **Metric M1: #Selectors** = Total number of selectors referenced inside jQuery code statements (i.e. the volume of selectors)

- **Metric M2: #DistinctSelectors** = Total number of distinct selectors referenced inside jQuery code statements (i.e. the vocabulary of selectors)

- **Metric M3: #DistinctTypesOfSelectors** = Total number of distinct types of selectors referenced inside jQuery code statements (i.e. the typology of selectors). The type for a selector is resulting by neutralizing the selector, e.g.

Selector	Type
div span	tag tag
#header a	id tag
div.menu	tag.class
input[type=text]	tag[name=value]
p:first-child	tag:pseudo-class

- **Metric M4: #IdAndClasses** = Total number of id and classes that are used in jQuery selectors

- **Metric M5: #DistinctIdAndClasses** = Total number of distinct id and classes that are used in jQuery selectors

- **Metric M6: #MeaningfulDistinctIdAndClasses** = Total number of distinct id and classes that are used in jQuery selectors and their names are meaningful. We believe that this metric could give a strong indication of a code that can be or cannot be easily maintained. The problem with this metric is that we need human interference.

In order to estimate the quality of jQuery selectors, we intend to calculate the cognitive complexity of the selectors, in terms of cognitive weights, much like Wang and Shao (Wang and Sao, 2003) proposed for classic software. In order to accomplish that, we have to estimate the cognitive weight for each type of selectors, by asking jQuery developers to rank them.

Conclusion

In this work, we outline a methodology that we used to measure the quality and the performance of JS Frameworks. This methodology could be used by web developers today and in the future in order to select the most appropriate framework. In this work we followed this methodology and tried to accomplish an overall quality, validity and performance assessment of the most commonly used JFs today. The outcomes of this assessment are outlined below for each stakeholder:

Developers:

- Before choosing among JFs, take into consideration their quality and maintainability metrics. These metrics are almost always ignored, but if you are about to invest in a technology you should not. A software that is hard to maintain, it is hard to evolve and the possibility to be abandoned in the near future is not negligible.

- Specific types of selectors were measured to be very slow, especially in mobile devices and should be avoided. Use it for best performance.

Jf Communities:

- Our intention was to reveal to their supporting community the drawbacks and help them in producing high quality, full-featured JavaScript frameworks for the web developers. We have already contacted the core development team of the JQuery Frameworks, and discussed with them the results of this study.

- The results of the quality tests reveal some functions and points in code that probably need to be improved.

- The validation tests also reveal that some parts of code must be modified in order to harmonize with browsers continuous evolvement.

End-Users:

- Upgrade to the last version of your favorite browser, as there are numerous compatibility and performance issues with old versions of browsers, which may cause several high-end applications built on JavaScript Frameworks not to work properly.

Some of these metrics will probably prove to be important. Others not. Regarding future work, we plan to specify and conduct similar quality and performance tests on every JavaScript Framework and especially on real web applications with the various JFs (including Ajax implementations and rich interface applications). We also outline that mobile applications and mobile versions of some JFs will be examined more thoroughly. Moreover, we are already working on building a web-based test-suite that developers can use to do all tests at one place and also have the capability to test older versions of the JFs in order to observe the evolution of the frameworks (this is good indication for the future). More metrics should be proposed for the actual evaluation not only of jQuery programs but for all JF frameworks.

ACKNOWLEDGMENTS

Our thanks to the communities of the frameworks involved for their suggestions and detailed information in order to reveal to us their methodology and tools that they use during testing and deployment of their JFs. We thank Dave Methvin, President of the jQuery Foundation, for his quick answers and his proposals and suggestions about what and how should we measure in the jQuery code.

REFERENCES

"Dojo vs JQuery vs MooTools vs PrototypevsYUI vs Dojo". http://blog.creonfx.com/javascript/mootools-vs-jquery-vs-prototype-vs-yui-vs-dojo-comparison-revised.

BarkmannH, LinckeR, LoweW, 2009, Quantitative evaluation of software quality metrics in open-source projects. *In Proceedings of the 2009 IEEE International Workshop on Quantitative Evaluation of largescale Systems and Technologies (QuEST09)*, UK, Bradford.

Calero, C., Ruiz, J., Piattini, M., 2004, A Web Metrics Survey Using WQM. *Proceedings of The Fourth International Conference on Web Engineering*, LNCS 3140, Springer-Verlag, Berlin, pp. 147-160.

Chuan Yue, Haining Wang, 2009, Characterizing Insecure JavaScript Practices on the Web. *In Proceeding WWW '09 Proceedings of the 18th international conference on World Wide Web.* Madrid, Spain, pp 964-965.

Chuck Crandall, Sep 08, 2006. Javascript Toolkit Comparison https://wiki.jasig.org/display/UP3/Javascript+Toolkit+Comparison.

Compare JavaScript frameworks: *An overview of the frameworks that greatly enhance JavaScript development.IBM 2010.* http://www.ibm.com/developerworks/web/library/wa-jsframeworks/.

Evaluation of Javascript Libraries. http://wiki.freaks-unidos.net/javascript-libraries.

G. Richards, S. Lebresne, B. Burg and J. Vitek, June 2010, An analysis of the dynamic behavior of JavaScript programs. *In Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation (PLDI).* Canada, Ontario, Toronto, pp 1.

Halstead, Maurice H. 1977, *Elements of Software Science, Operating, and Programming Systems Series*, Vol 7. New York, NY: Elsevier Science Inc.

http://gs.statcounter.com/#os-ww-monthly-201004-201104.

IEEE Std. 1061-1998 IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998.

Kafura, D. and J. Canning., 1985. A Validation of Soft-Metrics Using Many Metrics and Two Resources. *Proc. 8th Intl. Conf. on Software Engineering.* IEEE Computer Society Press, Washington, D. C.

Kamaljit Kaur, Kirti Minhas, Neha Mehan, Namita Kakkar, 2009, Static and Dynamic Complexity Analysis of Software Metrics. *Empirical Software Engineering*, Vol: 56, Issue: V, Pages: 159-161.

LiLi, H.F., Cheung, W.K., 1987. An empirical study of software metrics. IEEE Trans. Software Engrg. SE-13, pp. 697-708.

Mata-Toledo, R.A., Gustafson, D.A., 1992. A factor analysis of software complexity measures. J. Syst. Software 17, pp 267-273.

N Gayatri, S Nickolas, A.V.Reddy, November 2009,

Performance Analysis and Enhancement of Software Quality Metrics using Decision Tree based Feature Extraction. *International Journal of Recent Trends in Engineering*, Vol 2, No. 4.

Olsina, L. and Rossi, G. 2002. Measuring Web Application Quality with WebQEM. *IEEE Multimedia*,Vol. 9(4), pp 20-29.

Oman, P. & Hagemeister, 1992 J. "Metrics for Assessing a Software System's Maintainability," *Conference on Software Maintenance 1992*. CA: IEEE Computer Society Press, Orlando, FL, Los Alamitos, pp 337-344.

P. Ratanaworabhan et al., 2010, JSMeter: Comparing the behavior of javascript benchmarks with realweb applications. *In Webapps'10*, pp 27–38.

Rosales-Morales, V.Y. Alor-Hernández, G. Juárez-Martínez, 2011, An overview of multimedia support into JavaScript-based Frameworks for developing RIAs. *In U. Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference,* IEEE.

Scott Trent , Michiaki Tatsubori, Toyotaro Suzumura, Akihiko Tozawa , Tamiya Onodera, 2008 Performance comparison of PHP and JSP as server-side scripting languages . *Proceeding Middleware '08 Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware,* Springer-Verlag New York, Inc., USA, New York.

Server-Side JavaScript Reference v1.2. http://research.nihonsoft.org/javascript/ServerReferenceJS12

The State of Web Development 2010- Web Directions Survey, http://www.webdirections.org/sotw10/.

Wang, Y., and Sao, J. 2003. " A new Measure of Software Complexity based on Cognitive Weights", *Canadian Journal of Electrical and Computer Engineering*, 28 , (2) , 69-74.

Yue Jiang, Bojan Cuki, Tim Menzies, Nick Bartlow, 2008. Comparing design and code metrics for software quality prediction**.** *Proceedings of the 4th international workshop on Predictor models in software engineering*, PROMISE '08.