Instructions for ACL-2013 Proceedings

Cedric Takongmo
cedric.takongmo@mni.thm.de

Abstract

Get a concise introduction to Jasmine, the popular testing framework for JavaScript. This document shows you how to write unit tests with Jasmine that automatically check for bugs in your application using the Test-Driven development pattern and the Karma test runner. After you get an overview of test-driven development, learn how to write specifications for individual components, and then use those specs to test the code you write. Write useful specs by determining what you need to test and what you dont. Test the behavior of new and existing code against the specs you create. Apply Jasmine matchers and discover how to build your own. Organize code suites into groups and subgroups as your code becomes more complex. Use a Jasmine spy in place of a function or an object and learn why its valuable.

1 Credits

This document has been wrote on the basis of some external Internet links and a presentation of Christopher Bartling about JavaScript tests with Jasmine and Karma. The statistics to illustrate the importance of JavaScript today come from studies of diverse companies like TIOBE software. A detailed presentation of the Jasmine library was inspired by the online documentation on the official Jasmine website.

2 Motivation for Javascript TDD

Today, Javascript is growing in popularity and will continue to grow. As show in the Abbildung 1, from 1996 to 2016, JavaScript is from the 23th to the 7th position of the most used programming languages. JavaScript is also used all over the place, and is used even with every backend language. This programming language is also the most commonly used in frontend development to dynamically give more user experience to web applications. JS-libraries are used for the development of web and mobile applications with JQuery Mobile/PhoneGap. Famous libraries like Angular.js, jQuery, Node.js, React and various others tools inherit JavaScript. Some hardware engineer are using JavaScript to do embedded programming so that JavaScript can be used for the direct communication with the outside world by using MicroControllers and other electronics related stuff, although this is definitely less common at the moment. This is why JavaScript is such a good language to learn. (https://www.quora.com/When-do-developers-use-JavaScript-and-why).

A software product has to be considered like every other commercial product and it quality must be tested. This is also valid for javaScript Software. In the software engineering, we are talking about Code Quality. Code quality is a loose approximation of how long-term useful and long-term maintainable the code is. If the Code is thrown away tomorrow is mean that this Code has a Low quality. High quality Code is being carried over from product to product, developed further, maybe even open sourced after establishing is value. The software quality is defined with a clear and understandable design and implementation and also well defined interfaces. Good Code is ease to build and use and have to be extensible. The minimum extra dependencies and the good documentation are very important criteria for Software Quality. One off he most important criteria are the tests: Unit, integration, and functional/acceptance testing. (https://www.quora.com/How-do-you-define-code-quality)

In this context of Code Quality and Unit Testing in JavaScript, Test-driven development (TDD)

is the new trend. TDD is a new way to software development which combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring. What is the primary goal of TDD? One view is the goal of TDD is specification and not validation (Martin, Newkirk, and Kess 2003). In other words, its one way to think through your requirements or design before your write your functional code (implying that TDD is both an important agile requirements and agile design technique). Another view is that TDD is a programming technique. As Ron Jeffries likes to say, the goal of TDD is to write clean code that works. I think that there is merit in both arguments, although I lean towards the specification view, but I leave it for you to decide.

3 Test-driven development cycle

TDD can be described with this simple formula: TDD = The steps of test first development (TFD) + a Refactoring. TFD are overviewed in the UML activity diagram of Abbildung 2. With this approach a test should be quickly added. Normally just enough code to fail. After that the tests should be run, often the complete test suite although for sake of speed you may decide to run only a subset, to ensure that the new test does in fact fail. Then, You update your functional code to make it should pass the new tests. The fourth step is to run your tests again. If they fail you need to update your functional code and retest. Once the tests pass the next step is to start over (you may first need to refactor any duplication out of your design as needed, turning TFD into TDD) (http://agiledata.org/essays/tdd.html)

4 Benefits of TDD

TDD enables you to take small steps when writing software. This is the most important thing of this concept. This practice is far more productive than attempting to code in large steps. For example, assume some new functional code have been added to code, compiled, and tested. Chances are important that your tests will be broken by defects that exist in the new code. It is much easier to find, and then fix, those defects if you've written two new lines of code than two thousand. The implication is that the faster your compiler and regression test suite, the more attractive it is to proceed in smaller and smaller steps. I generally prefer to add a few new lines of functional code, typically less than ten, before I recompile and rerun my tests.

5 Karma

Karma is a test runner for JavaScript that runs on Node.js. It is very well suited to testing any JavaScript projects. Using Karma to run tests using one of many popular JavaScript testing suites (Jasmine, QUnit, Mocha, etc.) and have those tests executed not only in the browsers of your choice, but also on any platform (desktop, phone, tablet.) Karma is highly configurable, integrates with popular continuous integration packages (Jenkins, Travis, and Semaphore) and has excellent plugin support. (http://www.methodsandtools.com/tools/karma.php) Karma is an Open Source distribution and actually in the version 1.0.

- Installation - Karma requires Node.js and the Node Package Manager (NPM). So we can install Karma with the simple command:

  $ npm install -g karma

- Configuration - A configuration file have first to be created. Then Karma can do what you want. This configuration file can be a JavaScript or a CoffeeScript file. The configuration file can be created manually or generated step by step via the command line:

  $ karma init

  in this way, the created configuration file is then as follows:

- Application - Karma is started on the console with the following command:

  $ karma start [path/to/config/file.js]

  All tests are now performed and Karma wait for code changes. while tests succeed, Karma will automatically start another test run. In the case of faulty test the result will be show in the console. In this case karma waits for an update or correction of JavaScript code. (https://blog.mayflower.de/4333-Karma-Testrunner-Einfuehrung.html)

5.1 Layout

Format manuscripts two columns to a page, in the manner these instructions are formatted. The exact dimensions for a page on A4 paper are:

- Left and right margins: 2.5 cm
- Top margin: 2.5 cm
- Bottom margin: 2.5 cm
- Column width: 7.7 cm
- Column height: 24.7 cm
- Gap between columns: 0.6 cm

Papers should not be submitted on any other paper size. If you cannot meet the above requirements about the production of your electronic submission, please contact the publication chairs above as soon as possible.

5.2 Fonts

For reasons of uniformity, Adobe's **Times Roman** font should be used. In LaTeX2e this is accomplished by putting

```
\usepackage{times}
\usepackage{latexsym}
```

in the preamble. If Times Roman is unavailable, use **Computer Modern Roman** (LaTeX2e's default). Note that the latter is about 10% less dense than Adobe's Times Roman font.

Type of Text	Font Size	Style
paper title	15 pt	bold
author names	12 pt	bold
author affiliation	12 pt	
the word "Abstract"	12 pt	bold
section titles	12 pt	bold
document text	11 pt	
captions	11 pt	
abstract text	10 pt	
bibliography	10 pt	
footnotes	9 pt	

Table 1: Font guide.

5.3 The First Page

Center the title, author's name(s) and affiliation(s) across both columns. Do not use footnotes for affiliations. Do not include the paper ID number assigned during the submission process. Use the two-column format only when you begin the abstract.

Title: Place the title centered at the top of the first page, in a 15-point bold font. (For a complete guide to font sizes and styles, see Table 1) Long titles should be typed on two lines without a blank line intervening. Approximately, put the title at 2.5 cm from the top of the page, followed by a blank line, then the author's names(s), and the affiliation on the following line. Do not use only initials for given names (middle initials are allowed). Do not format surnames in all capitals (e.g., use "Schlangen" not "SCHLANGEN"). Do not format title and section headings in all capitals as well except for proper names (such as "BLEU") that are conventionally in all capitals. The affiliation should contain the author's complete address, and if possible, an electronic mail address. Leave about 2 cm between the affiliation and the body of the first page. The title, author names and addresses should be completely identical to those entered to the electronical paper submission website in order to maintain the consistency of author information among all publications of the conference.

Abstract: Type the abstract at the beginning of the first column. The width of the abstract text should be smaller than the width of the columns for the text in the body of the paper by about 0.6 cm on each side. Center the word **Abstract** in a 12 point bold font above the body of the abstract. The abstract should be a concise summary of the general thesis and conclusions of the paper. It should be no longer than 200 words. The abstract text should be in 10 point font.

Text: Begin typing the main body of the text immediately after the abstract, observing the two-column format as shown in the present document. Do not include page numbers.

Indent when starting a new paragraph. Use 11 points for text and subsection headings, 12 points for section headings and 15 points for the title.

5.4 Sections

Headings: Type and label section and subsection headings in the style shown on the present document. Use numbered sections (Arabic numerals) in order to facilitate cross references. Number subsections with the section number and the subsection number separated by a dot, in Arabic numerals. Do not number subsubsections.

Citations: Citations within the text appear in parentheses as (Gusfield, 1997) or, if the author's

name appears in the text itself, as Gusfield (1997). Append lowercase letters to the year in cases of ambiguity. Treat double authors as in (Aho and Ullman, 1972), but write as in (Chandra et al., 1981) when more than two authors are involved. Collapse multiple citations as in (Gusfield, 1997; Aho and Ullman, 1972). Also refrain from using full citations as sentence constituents. We suggest that instead of

> "(Gusfield, 1997) showed that ..."

you use

> "Gusfield (1997) showed that ..."

If you are using the provided LaTeX and BibTeX style files, you can use the command `\newcite` to get "author (year)" citations.

As reviewing will be double-blind, the submitted version of the papers should not include the authors' names and affiliations. Furthermore, self-references that reveal the author's identity, e.g.,

> "We previously showed (Gusfield, 1997) ..."

should be avoided. Instead, use citations such as

> "Gusfield (1997) previously showed ... "

Please do not use anonymous citations and do not include acknowledgements when submitting your papers. Papers that do not conform to these requirements may be rejected without review.

References: Gather the full set of references together under the heading **References**; place the section before any Appendices, unless they contain references. Arrange the references alphabetically by first author, rather than by order of occurrence in the text. Provide as complete a citation as possible, using a consistent format, such as the one for *Computational Linguistics* or the one in the *Publication Manual of the American Psychological Association* (American Psychological Association, 1983). Use of full names for authors rather than initials is preferred. A list of abbreviations for common computer science journals can be found in the ACM *Computing Reviews* (Association for Computing Machinery, 1983).

The LaTeX and BibTeX style files provided roughly fit the American Psychological Association format, allowing regular citations, short citations and multiple citations as described above.

Appendices: Appendices, if any, directly follow the text and the references (but see above). Letter them in sequence and provide an informative title: **Appendix A. Title of Appendix**.

Acknowledgement section should go as a last section immediately before the references. Do not number the acknowledgement section.

5.5 Footnotes

Footnotes: Put footnotes at the bottom of the page and use 9 points text. They may be numbered or referred to by asterisks or other symbols.[1] Footnotes should be separated from the text by a line.[2]

5.6 Graphics

Illustrations: Place figures, tables, and photographs in the paper near where they are first discussed, rather than at the end, if possible. Wide illustrations may run across both columns. Color illustrations are discouraged, unless you have verified that they will be understandable when printed in black ink.

Captions: Provide a caption for every illustration; number each one sequentially in the form: "Figure 1. Caption of the Figure." "Table 1. Caption of the Table." Type the captions of the figures and tables below the body, using 11 point text.

6 Translation of non-English Terms

It is also advised to supplement non-English characters and terms with appropriate transliterations and/or translations since not all readers understand all such characters and terms. Inline transliteration or translation can be represented in the order of: original-form transliteration "translation".

7 Length of Submission

Long papers may consist of up to 8 pages of content, plus two extra pages for references and short papers may consist of up to 4 pages of content, plus two extra pages for references, in the proceedings. Papers that do not conform to the specified length and formatting requirements are subject to be rejected without review.

8 Other Issues

Those papers that had software and/or dataset submitted for the review process should also submit it

[1] This is how a footnote should appear.

[2] Note the line separating the footnotes from the text.

with the camera-ready paper. Besides, the software and/or dataset should not be anonymous.

Please note that the publications of ACL 2013 will be publicly available at ACL Anthology (http://aclweb.org/anthology-new/) on July 28th, 2013, one week before the start of the conference. Since some of the authors may have plans to file patents related to their papers in the conference, we are sending this reminder that July 28th, 2013 may be considered to be the official publication date, instead of the opening day of the conference.

Acknowledgments

Do not number the acknowledgment section. Do not include this section when submitting your paper for review.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Association for Computing Machinery. 1983. *Computing Reviews*, 24(11):503–512.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.