A Testing Framework for Web Applications

By Taggart Ashby

February 4, 2014

Abstract

Our framework is the best framework because we made it and we're awesome.

1 Introduction

Since the advent of the internet, the web has undergone an impressive evolution from plain-text webpages to the highly stylized and functional pages that we see today. In the last four to six years there has been an explosion of new web technologies that make applications more functional: HTML5, CSS3, WebGL, Touch APIs, Geo-location, and a host of others. [7] In that same time period, the number of global internet users has grown to somewhere around 2.5 billion people. [7] The combination of these new technologies and the ever-increasing usage of the internet has led to a significant growth in the number and complexity of web applications. Like any piece of software, these web applications ought to be tested in order to determine their correctness and give their users the best experience possible. Unfortunately, web applications are the new kids on the block and there is very little conventional wisdom on how to test them. Their dynamic nature and network communication require a different testing toolkit than their predecessors. In addition, web applications are being produced at an incredible rate and so there's often little time to test. We can see that this is not a solved problem by looking at some of the biggest web application around. A 2011 study showed that YouTube has at least eight errors, Apple has at least seven, Microsoft at least four, and the list goes on. [4] That may seem like a incredibly small amount for such monoliths, and, honestly, it is fairly small, but those are still errors in the software that ought to be fixed. What this all boils down to is that there needs to be a framework in place for testing these multi-layer, multi-faceted web applications that is easy to use, quick to develop on, and thorough in its coverage.

2 Background

In an effort to make the rest of the paper more understandable and to define the terms used in the remainder of the paper, it is useful to have a section pertaining to the different concepts and technologies surrounding the web. If one is familiar with HTML5, JavaScript, and Testing concepts and statistics, it will likely be safe to skip this background section.

2.1 Web Application Strengths

Web applications are great for both developers and consumers. They have a number of characteristics that make them more desirable than shrink-wrap software. These attributes are versioning, availability, platform independence, and the ability to rapidly prototype.

2.1.1 Versioning

Web applications can be updated at any time with instant roll-out and feedback. It requires absolutely no user interaction because the web page always serves the latest resources. In comparison to something like an operating system that requires users to manually update,

this feature leads to applications that can address problems quickly, roll-out security updates instantly, and incorporate consumer feedback much more quickly than any other type of software. Another perk is that users are not given a choice. This can lead to occasional customer outrage, but more often than not it's a powerful gain for both feature improvement and application security.

2.1.2 Availability

Web application are available anywhere there's internet. No need to install anything on a given device most of the time. Due to the ever-increasing popularity of mobile devices, most applications have both mobile and full versions of their software and so the application is "with you" all the time.

2.1.3 Multi-platform

Very similar to the availability strength, web applications are platform independent. Whether you're on your phone, tablet, laptop, or PC, the application only requires a web browser with certain functionality and in this day and age all devices come equipped with browsers that should have no problems.

2.1.4 Rapid Prototyping

Web applications are fantastic for rapid prototyping because of all of the strengths discussed above. An idea can be prototyped and posted on a website quickly and rapid iteration can occur without the user even necessarily noticing. There are a number of web tools that will outline where users are clicking, what features they're using, how long they're spending on a given page or step of a process, and a number of other metrics that can make iterations more directed and effective.

2.2 Web Features

Here I'll discuss some of the prevalent technologies that recently appeared in web applications and development. By no means is this an exhaustive list, just a sampling of the more prominent ones.

2.2.1 HTML5

The latest revision of HTML is HTML5 which first debuted on Firefox back in 2009. [7] HTML5 has not been officially recommended by the W3C (World Wide Web Consortium) who is the official keeper and recommender of web standards, however, there is a plan for final recommendation this year, 2014, called Plan 2014. [3] What this means is that in 2014 the W3C will release a stable HTML5 Recommendation which will make HTML5 an official standard and help universalize all the separate implementations.

HTML5 brings a number of new features including: audio and video tags, the canvas element, drag-and-drop functionality, web storage, offline page support, and a number of other modern web features. A lot of HTML5's features revolve around the inclusion of multimedia in web pages and stronger graphical processing and programming.

2.2.2 JavaScript

JavaScript, though not the only scripting language with web use, is the primary language used to compliment HTML when making web applications. It is an interpreted language that is most often associated with client-side functionality, but in recent times has extended to the server-side. JavaScript is a prototype based object oriented scripting language with dynamic typing and first-class functions. Much of JavaScript's popularity stems from this dynamic nature that allows for quick, and often dirty, implementation of features in an often iterative environment.

2.2.3 Node.JS

Node.JS, often called simply Node, is a JavaScript API for creating and running servers. "Node's goal is to provide an easy way to build scalable network programs." [9] Node was created in 2009 and is sponsored by Joyent. Node does not use thread-based networking, instead opting for a single-threaded event loop and non-blocking I/O. Node allows developers to create and control web servers without the need for external software, such as Apache, commonly found in other sites.

Node is found in a number of popular sites including: PayPal, The New York Times, Yahoo!, Uber, LinkedIn, Microsoft Azure, and a number of others. [9]

2.3 What is testing?

IEEE defines software testing as "the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior." [1] Put a bit more simply, software testing is the process of making sure the program does what it is supposed to.

There are a number of different types of tests and testing concept, we will just focus on Unit Testing and Continuous Integration.

2.3.1 Unit Testing

Unit testing is the testing of specific functions and functionality. Generally this means the testing of individual functions but can extend to a set of functions that produce a single module of functionality. What sets Unit Testing apart from other types of testing is that tests focus on one portion of the code or system rather than interactions between modules or the entire system.

Here's a brief example:

```
// Terribly useless code whose purpose is to eventually calculate a magic number
function addTwo (numToAddTo) {
        return numToAddTo + 2;
}

function twoTimes (numToMultiply) {
        var sum = 0;

        for (var i = 0; i < numToMultiply; i++) {
                sum = addTwo(sum);
        }

        return sum;
}

.... // More functions

function calculateMagicNumberPartOne (startingNum) {
        var foo = 3 + addTwo(startingNum);

        return twoTimes(foo);
}

... // More functions

function calculateMagicNumber (startingNum) {
        var numSoFar = calculateMagicNumberPartOne(startingNum);

        numSoFar = calculateMagicNumberPartTwo(numSoFar);
        numSoFar = calculateMagicNumberPartThree(numSoFar);
        ... // More calculateMagicNumberPart(s)

        return numSoFar;
}

function testAddTwo () ... // Unit Test
function testTwoTimes () ... // Unit Test
function testCalculateMagicNumberPartOne () ... // Still a Unit Test, just a
    whole module of functionality

function testCalculateMagicNumber () ... // NOT a Unit Test because it's testing
    the whole system
```

2.3.2 Continuous Integration

Continuous Integration (CI), in the context of testing, also known as Continuous Testing, is the term for a piece of software or service that continually runs tests on your code as you develop. The purpose of this is to make sure that new code is not breaking old tests and that new code is passing any tests you've written for it prior to writing it. This is invaluable feedback because you can see if you're making progress in the right direction as you write code, as opposed to a situation in which you finish coding and find out you made a bad assumption or decision early on.

One of the most important aspects of a good piece of CI software is that it is automatic. There should be no need for a developer to do anything besides "turn it on" and start coding. As soon as a tool, CI or otherwise, requires more than one or two interactions to run, it becomes more of a nuisance and less of an asset.

2.3.3 Acceptance Testing

PUT INFO HERE PLEASE!!!

2.3.4 Testing Statistics

It is widely accepted that the earlier a developer detects a defect, the easier it is to fix that defect. A famous book, "Code Complete" by Steve McConnell [11], includes the following figure, showing this statistic on a rough graph.

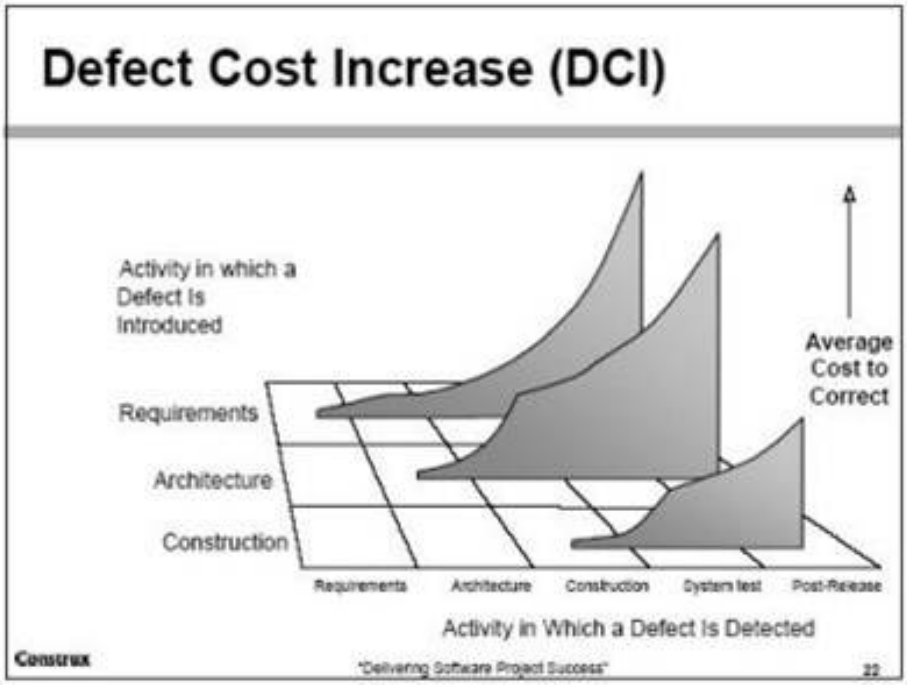

Figure 1: [11]

2.3.5 Why Testing is Important

I'm consistently surprised by how few people take testing seriously when it comes to non-trivial applications. Many developers will leave testing to their users, i.e., wait for users to have problems and report them before developers will fix them. This is an issue for a number of reasons, a few of which are: frustrated customers are unlikely to continue using a buggy product and are dissuaded from buying future products from the same developer, certain bugs can be security loopholes that may allow users access to information they shouldn't have, in the worst cases a bug may cause damage to a person or property and the developers would be on the hook.

2.4 Client/Server Architecture

Web applications have a user-facing client-side and a "rear-facing" (IS THIS THE RIGHT TERM...) server-side.

Here we'll discuss what the client-side is and does, what the server-side is and does, and how they communicate.

2.4.1 What is the Client-side Portion?

First, I'll address what a client is. The simple answer is: You. Your phone, computer, tablet, or other device communicating with a given website or application is the client. More specifically, the browser or service you're using to communicate with the web is the client.

So now that the client has been clarified, what is the client-side portion of an application? The client-side portion interacts with the browser to request assets from the server and then displaying them in a human-readable way. The client-side also deals with the user interface and interactivity of a web page, e.g., the client-side displays a web form, the client fills in the information and clicks "submit" and the client-side packages up the data and sends it to the server-side.

Client-side programming is done in a language that the browser understands, most frequently Javascript. Though not technically programming languages, client-side programming is also done through HTML and CSS which control the look and feel of web pages.

2.4.2 What is the Server-side Portion?

Once again, I'll outline a server and then delve into server-side programming. A server is the entity, be it machine, person, or cloud service, that stores the information about a given web page or application. The server stores all assets, HTML pages, images, videos, databases, etc, and provides them upon request to the client. The server also handles all of the processing for interactivity, accounts, and a general number-crunching.

The server-side portion of an application, then, stores all of the assets, accounts, and structure of the entire application. The server-side authenticates users, retrieves data from databases, processes large data, and send information, be it assets or results, back to the client to display to the user.

2.4.3 Client and Server Interactions

The client has limited access to resources on any given web application. This access is restricted by the server. Without such restrictions in place, someone could access user databases, credit card information, order reports, etc. In addition, there would be an overwhelming amount of content for a client to parse through, even if they have no malicious intent.

Clients communicate with the Server through a request-response architecture. Clients send a request to the server, sometimes with information attached, such as a form submittal or login, and the server returns a response with either a success/failure message, or additional data.

3 Product

4 Implementation

5 Evaluation

6 Related Work

I was unable to find any works that attempted to bring existing technologies together to create a web testing framework. Instead, I came across a number of proprietary tools and research. For the proprietary tool an advantage of the framework I already laid out is that it is freely available and has support behind it for each individual component. There is no need to contact any paper authors to get software or support. I will, however, still outline them as they provided insight into what the problems in existing frameworks were, gaps in coverage, what was important in a framework, and inspiration and acknowledgment that this is a real problem that many people are trying to solve.

I also found research into common bugs, general testing techniques, and finding bugs that greatly improved my test cases and understanding.

6.1 Testing Frameworks

6.1.1 "A Framework for Automated Testing of JavaScript Web Applications" [21]

A collection of IBM Researchers and two university students came up with a framework called "Artemis" for testing web applications. What is novel about their tool is that it generates test cases automatically based on execution patterns and feedback. They have attempted to solve the problem that writing test cases by hand is both time-consuming and often difficult. Their automatic test generation lead to an average of 69% test coverage with enough tests generated.

Though it was an interesting and somewhat effective method, 69% coverage is not great and this particular framework only covers the client-side, so it was incomplete for our purposes.

6.1.2 "A Multi-Agent Software Environment for Testing Web-based Applications" [17]

Two students and a person from Lanware Limited brought AI into the web testing world with an agent-based environment for testing web applications. In order to split testing into manageable tasks for agents to carry out, they created an ontology for web testing using XML. Each agent is set up to handle one particular kind of testing with certain test data, i.e., a unit tester with data for one or two functions. This agent may then communicate

with another agent who needs the results from that test, such as a test coverage agent or agent that is keeping track of test success and failure. This communication is done through a message-passing intermediary layer and a set of brokers who shuffle messages between agents.

This system is exciting and intricate but far too complex for a general web testing toolkit that just about any developer could pick up. It was important to weed techniques like this out as we wanted a fairly easy to use and understand toolkit.

6.1.3 "A 2-Layer Model for the White-Box Testing of Web Applications" [22]

A couple of gentlemen at the Center for Scientific Research and Technology in Povo, Italy came with a model for white-box testing of web applications. (White box testing is when you access to the underlying code.) This paper breaks up the testing process into two abstraction levels, the navigation model, the way in which a webpage goes from page to page, and the control flow model, the way in which information is passed and stored on a given page or between pages. These constitute the 2-layers in the title. The navigation model represents high-level test cases, like asserting that a given link redirects to the correct location, while control flow represents low-level test cases, like making sure data is persisted between pages.

This 2-layer approach is interesting and provided me with some insight, but the system was made for PHP applications and I wasn't looking to implement a brand-new system and test PolyXpress with it all in one year.

6.1.4 "Invariant-Based Automatic Testing of AJAX User Interfaces" [12]

Two individuals from the SOftware Engineering Research Group at Delft University in the Netherlands came up with a way to automatically test AJAX UI. The core of this idea is using a crawler to infer a flow graph. This paper outlines an plugin for an automated tool, ATUSA, for creating state validators and test suites for covering paths discovered during crawling (with a separate tool, CRAWLJAX). Their tests show that with minimal manual effort, the use of ATUSA can lead to high code coverage and error discovery.

This tool showed promise as an AJAX testing tool, but we wanted something more generic. Their crawler, CRAWLJAX, however, was very interesting and while we didn't use it, the techniques it used are used in other tools we have.

References

[1] "Systems and software engineering – vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, Dec 2010.

[2] "Software and systems engineering software testing part 1:concepts and definitions," *ISO/IEC/IEEE 29119-1:2013(E)*, pp. 1–64, Sept 2013.

[3] W. W. W. Consortium, "Plan 2014," 2012. [Online]. Available: http://dev.w3.org/html5/decision-policy/html5-2014-plan.html

[4] K. P. Frolin S. Ocasriza Jr. and B. Zorn, "Javascript errors in the wild: An empirical study," 2011. [Online]. Available: http://ece.ubc.ca/~frolino/projects/jser/tech_report.pdf

[5] P. Heidegger and P. Thiemann, "Contract-driven testing of javascript code," 2010. [Online]. Available: https://proglang.informatik.uni-freiburg.de/jscontest/jscontest/

[6] E. Hieatt and R. Mee, "Going faster: Testing the web application," *IEEE Software*, 2002. [Online]. Available: http://www.ecs.csun.edu/~rlingard/COMP595VAV/GoingFaster.pdf

[7] Hyperakt and Vizzuality, "The evolution of the web." [Online]. Available: http://www.evolutionoftheweb.com/

[8] M. M. J. T. Javier Jess Gutirrez, Maria Jos Escalona, "Testing web application in practice," in *First International Workshopl, WWV*, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.3910&rep=rep1&type=pdf#page=75

[9] I. Joyent, "Node.js." [Online]. Available: http://nodejs.org/

[10] V. Kongsli, "Security testing with selenium," 2007. [Online]. Available: http://www.kongsli.net/oopsla/OOPSLA'07%20Security%20Testing%20with%20Selenium%20v1.0.pdf

[11] S. McConnell, "Code complete, 2nd edition," 2004.

[12] A. Mesbah and A. van Deursen, "Invariant-based automatic testing of ajax user interfaces," 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.9899&rep=rep1&type=pdf

[13] S. Mirshokraie and A. Mesbah, "Jsart: Javascript assertion-based regression testing," 2012. [Online]. Available: http://www.ece.ubc.ca/~amesbah/docs/icwe12.pdf

[14] M. H. Mustafa Bozkurt and Y. Hassoun, "Testing web services: A survey," King's College London, Tech. Rep., 2010. [Online]. Available: http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-10-01.pdf

[15] D. A. Ostrowski, "Jaws: A javascript api for the efficient testing and integration of semantic web services," 2007. [Online]. Available: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-248/paper9.pdf

[16] A. B. Phillip Heidegger and P. Thiemann, "Dom transactions for testing javascript," 2010. [Online]. Available: https://proglang.informatik.uni-freiburg.de/jscontest/trans.pdf

[17] H. Z. Qingning Huo and S. Greenwood, "A multi-agent software environment for testing web-based applications," in *International Computer Software and Applications Conference*, 2003. [Online]. Available: http://student.cse.fau.edu/~jsloan11/CEN6076/002_MasArch4Test.pdf

[18] B. Rady and R. Coffin, *Continuous Testing with Ruby, Rails, and JavaScript.* The Pragmatic Bookshelf, 2011, ch. Creating a JavaScript CT Environment.

[19] H. Reza and D. V. Gilst, "A framework for testing restful web services," in *Seventh International Conference on Information Technology*, 2010. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5501468

[20] J. D. F. T. D. D. A. P. Shay Artzi, Adam Kiezun and M. D. Ernst, "Finding bugs in dynamic web applications," 2008. [Online]. Available: http://dspace.mit.edu/bitstream/handle/1721.1/40249/MIT-CSAIL-TR-2008-006.pdf

[21] S. H. J. A. M. Shay Artzi, Julian Dolby and F. Tip, "A framework for automated testing of javascript web applications," 2011. [Online]. Available: https://cs.uwaterloo.ca/~ftip/pubs/icse2011artemis.pdf

[22] P. Tonella and F. Ricca, "A 2-layer model for the white-box testing of web applications," in *Telecommunications Energy Conference, 2004. INTELEC 2004. 26th Annual International*, 2004.

[23] T. O. Valentin Dallmeier, Martin Burger and A. Zeller, "Webmate: A tool for testing web 2.0 applications," 2012. [Online]. Available: http://www.testfabrik.com/assets/pdf/webmate-jstools12.pdf