A Testing Framework for Web Applications

By Taggart Ashby

February 28, 2014

Abstract

We have brought together a number of testing tools from the web and created what we feel is a fully-featured, easy to use, testing framework for web applications.

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. [11] In that same time period, the number of global internet users has grown to somewhere around 2.5 billion people. [11] 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 require testing in order to determine their correctness and give their users the best experience possible. Unfortunately, web applications are new and there are no standard practices regarding 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 a need for rapid testing.

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. [7] 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.

1.1 What Has Been Done Already?

- Info info info

1.1.1 Why It Is Insufficient

- No standard - Info info info

1.2 Problems Still Faced/Why It's So Hard

- A lot of products to test against (browsers) - Asynchronous shennanigans

1.3 Contributions of This Thesis

- Framework - Wade through all the tools currently out there – Focus in, no need to have every developer check out all 10000 tools – Weigh options

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. [11] 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. [6] 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.

- Talk about GeoLocation
- TIE JS PORTION INTO THIS

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.

- THIS NEEDS TO MOVE OR CHANGE SUBSECTION TO 'WEB APPLICATIONS'

2.2.2 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." [14] 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.

3

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. [14]

- TIE THIS CLIENT/SERVER STUFF into Web Applications, be specific, not generic

2.3 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.

2.3.1 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.

2.3.2 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.

4

2.3.3   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.

2.4   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, Continuous Integration, and Acceptance Testing.

2.4.1   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
```

2.4.2   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.

2.4.3 Acceptance Testing

Acceptance testing is testing to confirm that your product meets a certain specification. Often in the terms of software this is making sure that your software meets an SRS (Software Requirements Specification) or a customer's outlined expectations. Often this testing is performed by the user of the software or that customer that ordered the software, rather than the developers.

2.4.4 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 [20], includes the following figure, showing this statistic on a rough graph.
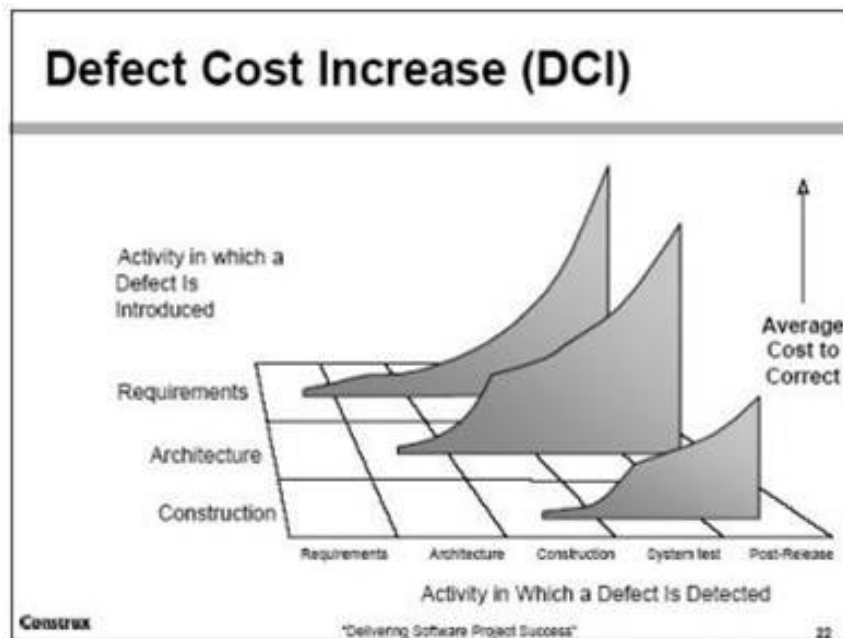

Figure 1: [20]

3 Product

4 Implementation

5 Sample App and Testing Frameworks

5.1 Assumptions and Requirements

In order to more thoroughly vet the many testing frameworks we came across, we had a number of assumptions about the projects we'd be looking at and requirements for those framework. - Node.js server because it's JS, so it's testable. - Something to do unit testing - Something to do continuous integration

5.2 Sample ToDo App

In order to test out the various frameworks that we found online, we needed a small application with some functionality, but without the inherent complexity that was going to come with our evaluation application. After some searching, I decided upon a Todo application made with Express.js, Node.js, and MongoDB. [19] This application was selected because it included Node.js and had enough functionality that tests wouldn't be completely trivial while also not being overwhelming. One of the challenges in testing that I quickly came across was the asynchronous nature of the MongoDB calls that were made in the sample app. Because of this headache, I will be discussing how the following frameworks deal with asynchronous test support as JavaScript is inherently asynchronous. - Screenshots - Example Navigation

5.3 Unit Testing Frameworks

The following frameworks are reviewed in chronological order of my evaluation. As such, comparisons will generally only be drawn between a framework and its predecessors in paper order. After all of the frameworks are outlined and evaluated, a longer section regarding the final decision will be made where comparisons will be drawn between all of them.

5.3.1 Jasmine [26]

The first of the unit testing frameworks that I looked at was Jasmine, a Behavior Driven Development (BDD) testing framework. BDD is based on Test Driven Development and adds some simplifications and patterns that attempt to bring both developers and businessmen into the software testing process. The idea behind Behavior Driven Development is that input from non-technical stakeholders as well as technical stakeholders can come together so that everyone understands what the project should do.

This relates to Jasmine is the way that you write Jasmine tests. The idea behind jasmine tests is that they read, as much as possible, like english sentences. As an example, the following is a valid Jasmine test:

```javascript
describe("Arithmetic Test", function () {
  it("should be able to compute the addition of two numbers", function () {
    expect(1 + 2).toEqual(3);
  });

  it("should be able to subtract numbers", function () {
    var theAnswer = 10;

    expect(12 - 2).toEqual(theAnswer);
  });
});
```

The "describe" statement creates a test suite to collect related tests under. Each "it" statement is generally one test with some number of "expect" statements. It may not be particularly pretty, but the above code is quite readable even to someone with no coding experience.

The default Jasmine package had some ineptitudes in dealing with Node.js but there was a package available through npm called jasmine-node [9] that gave jasmine access to the node package and functions.

Asynchronous testing using Jasmine was a bit of a chore, but there is support built in. This support comes in the form of a three-set function progression. The first function, "runs(function () { .... })", runs the asynchronous code. The second function, "waitsFor(function () { ... })", polls a flag that is set by the first function when it completes. Finally, the third function, another "runs(...)", houses the assertions to determine if the code ran properly or not. This method requires a fairly substantial amount of typing as well as complicated function chains when more than one asynchronous call is required.

NO WEB PAGE TESTING SO FAR

5.3.2 Mocha [37]

Mocha is another BDD testing framework that is almost identical to Jasmine. Unlike Jasmine, however, Mocha is not completely self-contained and requires an additional assertion library of one's choosing. I will outline my choice of Chai.js below.

Mocha also uses the "describe -¿ it(should...")" pattern to create tests. There is no standard for writing expectation though, as each assertion library is likely to be slightly different.

Mocha's approach to asynchronous testing is quite a bit more straightforward than Jasmine's. In your "it(...)" anonymous function you pass an additional parameter called "done" which you then call when you know the asynchronous call has finished. This re-

quired me to edit the code I was testing to make all the async calls accept an additional callback, but it was a very small price to pay for extended testability. An example asynchronous test would look like this:

```javascript
describe("Asynch Mocha Test", function () {
  it("look like this", function (done) {
    myAsyncCall(param1, param2, function () {
      // This callback expected to run when async returns

      // Test ...
      // Test ...
      // ...
      done();
    });
  });
});
```

More setup, more flexibility. Better Async support.

5.3.2.1 Assertion Library of Choice: Chai.js [3]   For the assertion library I chose Chai.js because it had the most flexibility for writing assertions that I'd found. It provides access to an Expect API, a Should API, and more general Assert API. The various APIs combined with Mocha's flexibilty led to a nice developer experience that can appeal to a variety of different developers.

5.3.3 QUnit [15]

QUnit is an Assertion style framework developed by the jQuery Foundation, the makers of jQuery, an immensely popular framework for DOM manipulation and access. QUnit is a bit more developer-centered and not as "feel-good" as the other frameworks but has, so far, been the most easy to use and has the most thorough test reporting of any framework out of the box. Mocha and Jasmine show how many "specs", the "it" functions, have passed, which is great information, but QUnit takes it a step further and breaks down each spec into the asserts contained within:

```
Assertions:
| Module        | Test                            | Assertion                          | Result |
| Task Spec     | All tasks can be shown          | Correct page is being rendered.    | ok     |
|               |                                 | Both tasks are being shown.        | ok     |
|               | A task can be added             | The additional task is present.    | ok     |
|               |                                 | The exact task we added is there.  | ok     |
|               | All tasks can be marked completed | Both tasks are marked complete.  | ok     |
|               | All completed tasks can be shown. | marked one as completed          | ok     |
|               |                                 | The one we marked completed is shown. | ok  |
|               |                                 | The correct page is shown.         | ok     |
|               | A single task can be marked as comple… | Our one task is marked completed. | ok |
|               | A task can be deleted.          | Only one task remains.             | ok     |
|               |                                 | Page was sent the delete code, 200 | ok     |
```

Figure 2: Each individual spec is further broken down into the assertions within it.

An interesting feature that QUnit has that I've not seen elsewhere is the ability to specify how many assertions you expect to run in the test. This is incredibly useful for callbacks. By setting an expected number of assertions, tests can no longer fail silently just because an assertion was never run. If the QUnit test finishes and the assertion count is not what was specified, it will fail the test and let you know fewer assertions were called than expected.

QUnit isn't perfect however, QUnit tests are not grouped into suites by default and the syntax for grouping them leaves much to be desired. In order to group tests into "modules", as QUnit calls them, the developer writes code like the following:

```javascript
Qunit.module("Module A"); // All code following is grouped into Module A
test("Addition", function () {
  equal(2 + 2, 4, "Two + Two = Four");
  equal(3 + 5, 8, "Three + Five = Eight");
});
test("Subtraction", function () {
  equal(4 - 2, 2, "Four - Two = Two");
  equal(2 - 2, 0, "Two - Two = Zero");
});

// More tests....

QUnit.module("Module B");
// Tests under Module B

// ... Etc
```

It is difficult to visually see where one module starts and ends and so grouping and finding the right place to add a test can be a chore.

QUnit's handling of asynchronous tests is similar to Mocha's except instead of adding a parameter to the "it" function, you call the "asyncTest" function instead of the "test" function and then call "start()" when you finish your asynchronous tests. Example:

```
asyncTest("Async Test", function () {
  myAsyncCall(param1, param2, function () {
    // This callback expected to run when async returns

    // Test ...
    // Test ...
    // ...
    start();
  });
});
```

5.3.4   Intern

5.3.5   Selenium

5.3.6   PhantomJS

5.3.7   ZombieJS

5.4   Continuous Integration Frameworks

Continuous Integration frameworks attempt to make development easier by running specified actions, usually tests, after a change to the project. This change can be simply saving a modified file, or, more usually, an action like pushing a change to a repository.

Continuous Integration frameworks, therefore, need to be flexible in the type of project they support, customizable in their chosen environments to support differing projects, and have a simple way to connect to repositories or file systems.

5.4.1   Jenkins [16]

Jenkins is the first of the continuous integration frameworks that I tried out. It was created by developers of a previous tool called Hudson when a dispute with Oracle led them to fork the Hudson project and rename it Jenkins. As of February 2014, the Jenkins organization has over 1,100 public repositories and 570 members. [13] Jenkins has, according to their website, two main focuses: (1) building and testing software continuously and (2) monitoring executions of externally-run jobs. The first focus is the general goal for any CI framework. The second focus is of note and will be discussed after the initial evaluation.

Jenkins is a series of web pages that you set up to run on a given server. This is what it looks like:

Most companies give Jenkins a sub-domain on their web pages.

Jenkins comes bare-boned but is highly customizable and has a wide array of plugins. The plugins available integrate Jenkins with various services, integrate Jenkins with source control systems, and even just change the look of the Jenkins page or how it behaves. As an example, Jenkins can integrate with GitHub to run builds and tests when changes are pushed to a branch.

Because Jenkins comes bare and runs on the system of your choosing, it can run any project that you set up. The downside is that there is no support out-of-the-box unless you already have the necessary compilers, interpreters, applications, and/or libraries installed on the server running Jenkins.

One of the exceptional strengths of Jenkins is that the developer is only limited by the hardware of the machine running Jenkins. Any increase in computing power will be directly usable by Jenkins. Jenkins can run as many jobs as your server can handle at one time.

The other use for a tool like Jenkins is the ability to start services like cron jobs or similarly long-running tools and status monitors and then collect the output for them and allow the developer to check on them at any time.

Jenkins is an exceptionally strong tool for CI, but requires immense overhead in setting up the environment to run your project in. However, once the server has the necessary applications and libraries, your builds and tests will run without issue. Jenkins can run an "unlimited" number of builds, tests, and processes with the only limiting factor the hardware that it is running on. It is also 100% free and can connect with public and private repositories alike.

5.4.2 Travis [4]

Travis is an online continuous integration framework with 29 members and 64 repositories at the time of this writing. Unlike Jenkins, Travis is all online and includes a dashboard from which you can monitor your builds and processes. Travis has a free version which is fully-featured but requires your project to be open-source and a paid version which gives you more processing power and access to private repositories.

Travis comes with internal support for eleven different languages including Node.JS, Ruby, Java, Python, and more. Travis has incredibly thorough documentation and a wealth of built-in services to work with your project if needed. Services like databases, notifications, deployment options, and GUI/Headless browsers. Travis supports all of the most popular databases: MySql, PostgreSql, MongoDB, CouchDB, and a myriad of others. It offers a number of notification options ranging from Email to IRC to HipChat (a chat service from Atlassian). Travis also offers deployment to just about every sort of provider there is, from Heroku to AWS S3 to RubyGems and also provides the ability to create custom web hooks to deploy to a provider not supported natively.

If at any point your project requires something that Travis does not support natively, you are given the permissions necessary to install any sort of libraries or services that can be installed with apt-get on an Ubuntu machine.

Travis requires that your projects be hosted on GitHub, but by doing so the setup is incredibly easy. You sign into Travis using your GitHub credentials and Travis pulls up all of your repositories automatically.

Getting a project running with Travis is fairly easy; you just create a yml file in your repository with certain lines pertaining to the language, needed services, commands to run, etc. I was able to set up everything I needed in six simple lines.

In order to get access to your private repositories, Travis offers several pricing plans, the cheapest of which is $129 [5] for their "Startup" plan. That's a bit steep, but it does offer unlimited private repositories, unlimited collaborators, and two concurrent jobs.

From what I gathered, Travis appears to be targeted at companies, not individual developers. Travis requires you to host your projects on GitHub and only supports one process running at a time. However, Travis is incredibly easy to set up, does not require you to have your own server, comes with impressive built-in features, and works wonderfully if you are willing to open-source your repositories and put them on GitHub.

5.4.3 Semaphore [29]

Semaphore is another online CI framework that connects to GitHub to bring in your projects. Semaphore is made by RenderedText, a software consulting team. Semaphore connects with your public and private repositories but requires a subscription after 30 days no matter which repositories you use.

Semaphore was originally designed for only Ruby applications but in January of 2014 they announced official support for Node.js and Javascript and as of February 26th they

now offer official support for Clojure projects.[30] These types of projects require no setup or files in your repository, Semaphore automatically detects project type and auto-fills build steps. Their "supported stack" page also lists C/C++, Java, Perl, and Python, but those must be configured manually.

Semaphore supports most databases and has a number of testing frameworks built-in. Once again you can use apt-get to install any missing libraries or dependencies. It would appear that deployment support is only available for Ruby projects.

Semaphore's pricing scheme is different than Travis' as it focuses on an individual, not a company. Their lowest cost subscription is one project (public or private) for $14 per month. They offer more subscriptions with the ability to have more projects and concurrent processes in higher tiers.

Semaphore is still ruby-centric and requires a subscription. As our goal is to find a framework that is free for open-source projects, Semaphore was out automatically. It is included here because it's well-known and fairly established, especially in the ruby community.

5.4.4   MagnumCI [18]

MagnumCI is another online CI, it appears that Jenkins is one of the only "off-line" CI systems. MagnumCI has every feature available for free, however, it's still in public beta and although prices haven't been released, the team making MagnumCI has already stated that there will be price tiers. It is not unreasonable, though, to believe that MagnumCI will be free for open-source projects, and that's what we care about.

MagnumCI offers support for Ruby, PHP, Node.JS, and Go. MagnumCI, much like Semaphore, will attempt to auto-detect commands that it should run based on your project structure and language. All settings are overridable in a yml file if you choose.

MagnumCI has all the popular databases built-in, many of which run automatically on startup. MagnumCI also has two headless browser frameworks built in for running any browser testing that needs to be done. The developer can, again, use apt-get to install missing packages. MagnumCI does not, at current, provide automatic deployment options given a successful build.

Where MagnumCI shines is in support of different repositories. MagnumCI offers support for pulling from GitHub, BitBucket, GitLab, and a few others. It does require a bit more setup to get your repositories connected, but what it comes down to is a couple lines of copied and pasted text within your repositories settings.

Right now MagnumCI offers impressive features, private repositories, and extensive repository sources, all for free. It is hard to say that MagnumCI is the best because those features may end up costing money soon after this paper is completed. It is included in this review because it seems to be an up and comer that could prove quite strong.

5.4.5 Drone.io [32]

Drone.io is an impressive online CI framework that offers substantial language support, is free for open-source projects, and offers the most reasonable pricing scheme for private repositories.

Drone.io supports eleven languages including: C++, Dart, Go, Haskell, Java, Node.JS, Ruby, and others. Setup requires you to specify your language, as opposed to the automatic tools provided by the previous two CI frameworks, but will auto-generate built commands for you.

Drone.io offers support for all of the databases you'd want, has a variety of deployment options, and is entirely configurable from the online dashboard. It also offers the most browsers and headless browsers for use with web page testing tools such as Selenium and PhantomJS.

Drone.io is able to connect to GitHub, BitBucket, and Google Code respositories to get your projects automatically. Setup is simple, select the source of the repository (GitHub, BitBucket, or Google Code), select the repository from the list it gives you once it connects, select the language of the repository, edit the default build script, and you're ready to go.

Drone.io offers unlimited free open-source repositories. The next step up offers 5 private repositories for $25 per month and for $50 per month you get unlimited private repositories. They also offer higher tier plans that allow for concurrently running builds.

Drone.io is a strong contender because of its language support, deployment options, and ease of setup. Where it lacks is in its notification options. The only notification option at present is email while other CI frameworks offer a number of other notification options for various messaging services.

5.4.6 Nightwatchjs

5.5 Misc Frameworks

5.5.1 Sauce Labs

5.5.2 Istanbul

5.5.3 Karma

5.6 Misc Tools

5.7 Node Inspector

Fantastic for debugging node.js server-side code!

6 Evaluation

7 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.

7.1 Testing Frameworks

7.1.1 "A Framework for Automated Testing of JavaScript Web Applications" [34]

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.

7.1.2 "A Multi-Agent Software Environment for Testing Web-based Applications" [27]

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.

7.1.3 "A 2-Layer Model for the White-Box Testing of Web Applications" [35]

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.

7.1.4 "Invariant-Based Automatic Testing of AJAX User Interfaces" [21]

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.

7.1.5 "JSART: JavaScript Assertion-based Regression Testing" [22]

Two students from the University of British Columbia created a tool called JSART that uses run-time code instrumentation and analysis to infer invariants that can be used for regression testing. When the code runs, JSART creates a number of test cases that will be run the next time the code executes. The point being to catch any discrepancies between old code and new code. They ran their tool on nine web systems with fairly strong results. The tool was not perfect and occasionally created inappropriate tests, but generally the tests were appropriate and useful.

This is a useful concept that could be integrated into the toolkit at a later time.

7.1.6 "DOM Transactions for Testing JavaScript" [25]

Three people from Albert Ludwigs University in Germany came up with a way to deal with test fixtures that are built and torn down frequently. When testing JavaScript, many of one's tests will change the underlying web page in some way. This is often different from testing traditional software where components are separated. In order to test JavaScript, test fixtures for setting up the page and tearing down the page are exceptionally important. Because these fixtures are being made and torn down frequently, this paper discusses a way to utilize transactional memory to quickly get the web page into the right state without constant setup and tear down.

This could be integrated with one of the existing tools we will use, but our goal was to collect existing tools that did not need alteration.

7.1.7 "Testing Web Applications in Practice" [12]

Four students from the University of Seville, Spain published a paper with an overview of web testing. This paper was more of an introduction to testing, outlining the concepts of Unit Testing, Integration Testing, Regression Testing, testing the client versus testing the server, and a few others. It also describes how to test an actual web application using PHPUnit, a library similar to JUnit, for those of you familiar with Java. When writing their tests, they suggest taking more abstract actions and breaking them up into their functional parts and testing each of those. As an example, inserting a customer into the database (albeit not that abstract) is broken into tests for the function that inserts the customer and tests for making sure the customer makes it into the database.

The paper was good refresher on testing concepts and a hands-on application of those concepts. There wasn't any new information or tools to put toward our toolkit, but it was a good reminder and outlined some testing procedures.

7.1.8 "Contract-Driven Testing of JavaScript Code" [8]

Two individuals from the Albert Ludwigs University in Germany created a tool, JSConTest, that provides a framework for making contracts for a JavaScript program, allowing for guided random testing on inputs and outputs. Contracts in JSConTest are comments above each function that are of the form "type -¿ type". More complicated forms can introduce parenthesis for function types. In general, JSConTest is useful for detecting type errors on input or output based on the operations performed within a function on the input and the result being output.

Although JSConTest could be useful in a toolkit, its source is no-where to be found and there were other, more robust, tools out there for our purposes.

7.1.9 "JAWS: A Javascript API for the Efficient Testing and Integration of Semantic Web Services" [24]

A researcher from Ford Motor Company published a paper regarding an API, called JAWS (Javascript, AJAX, Web Service), to facilitate testing and integration of Semantic Web Services. Semantic Web Services (SWS) are the server side of machine-to-machine interaction via the internet. This is opposed to machine-to-human interaction such as web pages that you visit personally. SWS use markup that is detailed and sophisticated which conforms to certain standards but is not human-readable.

Another interesting API and project, but we are not focusing on SWS and so including a tool or API like JAWS would be overkill and beyond trying to keep the toolkit as simple and robust as possible.

7.1.10 "WebMate: A Tool for Testing Web 2.0 Applications" [36]

Four people from Saarland University in Germany created a tool, WebMate, for automatically navigating web-pages with dynamic content and data. WebMate's goal is to output a usage model for testing purposes. This usage model allows tests to be focused on navigation as well as functionality. This dual usage leads to the ability to test the experience as well as the functions. In addition, WebMate can be used for Cross-Browser testing by running it on multiple browsers and comparing the results. In essence, WebMate is a web application crawler seeking to automatically suss out the layout and interaction between web pages to provide a model to the tester, be it a developer or the automatic testing capabilities of WebMate.

WebMate was another interesting project that could have seen use in our toolkit if crawlers weren't already available in other, more mainstream, tools.

7.1.11 "Continuous Testing with Ruby, Rails, and JavaScript" [28]

The well known book by Rady and Coffin talks about Continuous Testing and has lots of examples. The last two chapters of the book, six and seven, talk about setting up a Continuous Testing environment for JavaScript using Node.js and a number of other tools. It fit almost perfectly with our project and was used as the basis of my testing setup. They use JSLint for linting, Jasmine for test cases, and Watchr for running the tests continuously. Lastly it discussing writing effective tests, which is always important.

Not all of those tools made it into the eventual final toolkit, but they were all important to look at in the context of making the overarching toolkit.

7.2 Finding Bugs

One of the most important aspects of testing is actually finding the bugs in the program. A number of sources provided guidelines and suggestions for finding bugs in web applications.

7.2.1 "Finding Bugs In Dynamic Web Applications" [33]

Seven researchers from the MIT Computer Science and Artificial Intelligence Lab wrote a paper addressing testing in dynamic web applications. Common testing techniques were not made to work with dynamic content that can be generated at the drop of a hat or click of a button. Therefore, common testing techniques are inadequate for testing dynamic web applications. This paper discusses ways to find bugs in these new-fangled dynamic web applications. They use a new technique and extended a tool called Apollo. This technique generates dynamic test cases based on concrete and symbolic execution. "The basic idea is to execute an application on an initial input (e.g., an unconstrained or randomly chosen input), and then on additional inputs obtained by solving constraints derived from exercised control flow paths."

Although they test PHP applications, the extension to JavaScript is fairly straightforward. We didn't use Apollo, but the techniques described within allowed for more focused and productive testing.

7.2.2 "Testing Web Services: A Survey" [23]

Three researchers at the Centre for Research on Evolution, Search & Testing at King's College in London wrote a survey of web frameworks for testing web services. Their focus is on Service-Oriented Computing, a paradigm in which web sites or applications provide small pieces of functionality usable by "anyone". These services are then combined to create more fully featured applications. What they note in particular is that when using web services, you rarely have access to the source code and must therefore black-box test them. Corollary to that is that you have to implicitly trust whoever made the service. They go through and discuss a number of frameworks and come to the conclusion that it's most certainly not a solved problem for the following reasons: The frequency of testing required, Testing without disrupting the operation of the service, Determining when testing is required and which operations need to be tested. Regarding the frequency of testing, web services often change in effort to make improvements, but those improvements can change the services and break tests. Testing without disrupting service is difficult because these services often limit accesses or simultaneous access of both a test and a running web application. Finally, determining when and what to test comes back to the trust issue. Some services provide their own testing metrics that include code coverage, number of tests, etc. But you have to trust that they are thoroughly testing whatever it may be.

7.2.3 "A Framework for Testing RESTful Web Services" [31]

Two people from the University of Dakota School of Aerospace Studies outline a framework for testing RESTful web services. RESTful web services, short for Representation State Transfer, are, in the context of web applications, a name for web APIs that conform to a set of constraints. In particular they must offer four request methods: GET, POST, PUT,

and DELETE. These methods generally connect to a database or other storage system in order to store, retrieve, or delete entries therein.

Finish reading this paper... My bad Dr. Haungs.

7.3 Writing Effective Test Cases

Although not too distinct from *finding bugs*, writing effective test cases is its own art form. The important of writing concise, correct, productive test cases cannot be overstated. The difference between writing one test case that effective tests multiple things and writing a single test case for every little detail is the difference between developers wanting to write tests at all and skipping it due to time constraints or frustration.

7.3.1 "Going Faster: Testing The Web Application" [10]

Two employees of Evant Software wrote an article for IEEE Software regarding the testing of web applications. Mostly this paper was an overview and implementation of Test Driven Development (TDD) and eXtreme Programming, but in the middle of the article was a good section about the difficulties of testing the web and how to deal them. A few of the remarks regarding the difficulty of web testing is that often the client-side and server-side portions of web applications are written in different languages. This wasn't an issue for our application, but is still an issue for many applications. Some of the most useful advice was: "First, test those parts of the server-side code that are not directly concerned with being part of the Web application, without involving Web peculiarities. ... Second, test those parts of the client-side code that have no server interaction. This is typically code that contains little or no business logic. ... Third, write functional tests of a low grain in the server-side language (for example, Java or C++) that simulate the request/response Web environment. ... Using this framework you can write walk-through tests that simulate the user moving through the UI".

This step-by-step walkthrough of how to test was an excellent starting point to look at different tools and how they handled this issue.

8 Conclusion

9 Future Work

10 Appendices

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] "Chaijs," chaijs. [Online]. Available: http://chaijs.com/

[4] T. CI, "Travis." [Online]. Available: https://travis-ci.org/

[5] ——, "Travis plans." [Online]. Available: https://travis-ci.com/plans

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

[7] 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

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

[9] M. Hevery, "jasmine-node." [Online]. Available: https://github.com/mhevery/jasmine-node

[10] 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

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

[12] 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

[13] "Jenkins organization," Jenkins CI. [Online]. Available: https://github.com/jenkinsci

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

[15] "Qunit," The jQuery Foundation. [Online]. Available: http://qunitjs.com/

[16] K. Kawaguchi, "Jenkins." [Online]. Available: http://jenkins-ci.org/

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

[18] "Magnumci," Magnum CI. [Online]. Available: https://magnum-ci.com

[19] A. Mardanov, "Todo app with express.js/node.js and mongodb." [Online]. Available: http://webapplog.com/todo-app-with-express-jsnode-js-and-mongodb/

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

[21] 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

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

[23] 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

[24] 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

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

[26] "Jasmine," Pivotal Labs. [Online]. Available: http://jasmine.github.io/

[27] 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

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

[29] "Semaphore," RenderedText. [Online]. Available: https://semaphoreapp.com/

[30] "Semaphore blog," RenderedText. [Online]. Available: https://semaphoreapp.com/blog/

[31] 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

[32] B. Rydzewski and T. Burke, "Drone.io." [Online]. Available: https://drone.io/

[33] 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

[34] 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

[35] 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.

[36] 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

[37] "Mocha," VisionMedia. [Online]. Available: http://visionmedia.github.io/mocha/