Introducing Selenium

Jason Ebueng

K. Ngo

K. Nguyen

T. Yau

I. Brief History and Motivation

Selenium is a collection of tools that was created for automation testing for web applications. The original creator, Jason Huggins, developed a program in 2004 that automatically control the browser's actions on web applications. He called this the JavaScriptRunner, which later became the open source program called Selenium Core. The motivation behind Selenium was to have an efficient automated testing over time consuming repetitive work of manual testing. The four components of Selenium are Selenium Integrated Development Environment (IDE), Selenium Grid, Selenium Remote Control (RC), and WebDriver. Selenium is currently updated with a newer edition known as Selenium 2.

Selenium Integrated Development Environment

_____The creator of Selenium IDE is Shinya Kasatani, who wanted to help increase the speed of writing simple test cases. Selenium IDE, available through Firefox extension since 2006, is able to automate browsers and allows you to create test cases fast because it has a built-in help and test results reporting module. This IDE has a recording and playback feature for users to record their scripts and reuse them.

One of the advantages of Selenium IDE is that it is the easiest usable tool out of the four Selenium components. It has many capabilities such as being able to convert test to different programming languages or different formats to use on Selenium RC and

WebDriver. It allows users to be able to debug and set breakpoints. Users are able to utilize Selenium IDE without having experience in programming language.

Selenium Grid

_____Patrick Lightbody wanted to create a server that would speed up the time to execute test, thus Selenium Grid was created. This framework runs on a central hub and can simultaneously send out commands to different machines, known as nodes. It also has the capability of taking browser screenshots during testing. It speeds up the time to execute and complete test suite because Selenium Grid can run in conjunction with Selenium RC to perform parallel tests simultaneously on different browsers and operating systems.

Selenium Remote Control

_____Paul Hammant was motivated to have a tool that runs against the “same origin policy”. This policy restricts JavaScript code from accessing other domains from its original domain. He created Selenium RC around 2004 so that it can be used with web application that is not from the same domain. How it works is that Selenium RC gives instructions to Selenium Core to controls the browser. Selenium RC, also known as Selenium 1, is a versatile framework because it supports many languages such as Java, C#, PHP, Python, Perl, and Ruby. It has matured API and can execute test much faster than Selenium IDE. It is efficient in that it can perform looping and conditional operations. Data-driven testing is generally used in testing computer software and Selenium RC is able to support this testing methodology.

WebDriver

_____WebDriver is a cross-platform testing framework created in 2006 by Simon Stewart in response to powerful web applications that were becoming more restrictive with JavaScript programs. It supports the same languages as Selenium RC, but it is better in that it does not depend on JavaScript for automation and can execute test much faster than both Selenium IDE and Selenium RC. The functionality of WebDriver is efficient because it communicates directly to the browser. The architecture is quite simple in that you only need an IDE that contains the programming language and Selenium commands, and a browser.

Selenium 2

______In 2008, the Selenium development team decided to make an improved and much faster version of Selenium by merging Selenium RC and WebDriver together. Their motivation is to provide the best possible framework for automated testing. The disadvantages and limitations of Selenium RC are not a problem because WebDriver's capabilities make up for it; and vice versa. Selenium 2 can control the browser at the OS level, automatically generate an HTML file of test results, has simple user-friendly commands, and can support a broader range of browsers. It can also support HtmlUnit much like the WebDriver.

II. How Selenium Works with Agile Methodologies

The introduction of Agile software development has transformed the test paradigm considerably for many organizations. Agile methodologies were initially developed to solve the challenges present in the Waterfall development model. Under the Waterfall model, teams were failing to satisfy customers and meet schedule demands. Due to the enormous time lag between requirements acceptance and project completion, the final product often did not meet current needs, leading to cancellation of entire projects. To overcome this issue, Agile teams deliver software frequently and continuously at regular intervals. To better meet customer needs, teams under Agile development allow requirements to evolve throughout the software development cycle. While this approach was successful, it also changed the role of software testing considerably.

Under the Agile approach, software testers coming from manual and exploratory testing backgrounds will find it difficult to to keep up with the pace of delivery. Changing requirements is a regular concern even as systems reach critical sizes. Moreover, modern web development implies application compatibility with multiple browsers and devices. In order to address these issues and to keep up with delivery, organizations must use tools to automate testing. Selenium is a tool that automates testing on the client side, suiting well in Agile environments and providing many benefits. Selenium’s effortless integration with various development platforms and support for a wide range of languages allow minimal overhead work. Moreover, Selenium reduces manual work and accommodates well in continuous integration.

Continuous integration (CI) is a practice used in Agile development intended to solve the challenges of late software integration in the Waterfall model. Automated testing is core to Agile and continuous integration processes. In a standard CI process, each commit automatically triggers a suite of tests, preventing new code from containing defects and breaking existing functionality. Commits containing defects is automatically blocked from merging onto the main branch. Selenium’s role in CI is automating tests on the client side, reducing hundreds of hours of manual work. The reduction of manual labor allows developers to acquire feedback quicker and deploy high-quality code faster.

Selenium integrates seamlessly with a wide variety of development platforms to support continuous integration. One popular option is to integrate Selenium with Git and Jenkins. Jenkins is an open source automation server that automates continuous integration tasks related to building and deploying a project. In a typical continuous integration process, each commit will trigger a suite of regression tests. First, Jenkins will pull the source code and tests from Git to compile and run the web application. It then compiles tests, including client side tests on Selenium, to run on the application. Another popular tool to use with Selenium is Maven, a powerful build management tool. Maven is used with Selenium to help manage dependencies and tests and define project structures. Using “pom.xml”, a file in Maven, dependencies can be configured for tests and code. Maven will then automatically download necessary files from the repository when building the project. Another common tool that complements Selenium in projects is TestNG. TestNG is used to generate test results in report format since Selenium does not offer that function out of the box.

As previously mentioned, Selenium is a useful tool for reducing overhead work and supporting Agile processes. In addition to supporting continuous integration, Selenium also speeds up various other processes. For instance, developers may use Selenium for automation-aided exploratory testing. Scripts may be written in Selenium to call various functions in an exploratory sequence to uncover defects. Developers may also write bug reproduction scripts to speed up defect investigations.   Another benefit of Selenium is its support for different web browsers. Selenium makes compatibility testing simple across different devices and browsers.

In conclusion, findings from our research indicates that Selenium is a great framework for Agile environments. Agile methodologies aim to reduce busy work while also increasing quality and value. Selenium provides a great framework to utilize as part of that automated test strategy, and it reduces overhead work by making it simpler for developers to automate, record, and document user actions. In order for a team to be truly Agile, repeated processes, such as regression tests, should ideally be automated.

III. Selenium Code Demonstration

Selenium, at its heart, automates web browsing. It can host servers for massive machine testing for scale, however only one machine is needed. The automated framework follows a number of design goals that encourages many different types of programming and fits into different infrastructures. These goals include compatibility, conformance, consistent terminology and interfaces.

Since Selenium interacts with web browsing, it supports popular back-end languages in C#, Java, Python, and Ruby. Although there's a heavy presence of mobile technologies with iOS and Android platforms, the only front-end language supported is JavaScript. Given that both major mobile platforms support JavaScript, Objective-C and XCode are the main tools not represented by Selenium.

For the purpose of the demonstration, JavaScript code will be examined within OSX. Linux and Windows also can utilize Selenium within the programming language. Before going into Selenium's main API, it's important to establish environmental fundamentals. The developer should be familiar with JavaScript principles and tools, such as the Document Object Model (DOM) and general syntax for targeting HTML elements. The developer should have NPM (or Yarn) installed, then they can begin with the following command

```
$ npm install selenium-webdriver
```

into the terminal. Yarn has a similar command, and is a more performant package manager, but NPM will be emphasized here instead. This will install the JavaScript version of Selenium's web-driver, then the developer can utilize the API into their scripts and code. A common API for automated action is to visit a web page or URL:
`driver.get`

```
driver.get(website_url)
```

The driver is initialized from selenium-webdriver's Builder. To access that API, it must be imported from its dependencies:

```
const {Builder} = require('selenium-webdriver')
```

Then the driver can be created:

```
let driver = await new Builder().forBrowser('chrome').build()
```

Here the driver variable is a host for the web browser automated driver. In this case, a Google Chrome Web Browser is configured. There are options for other popular browsers, such as Safari, Mozilla Firefox, and Microsoft's Internet Edge as well. With this setup, Selenium will automatically open a Chrome Browser when called to action.

```
driver.get('https://google.com')
```

This will open the Google webpage. Once loaded, Selenium can target elements. Within the Google webpage, the iconic search box can be targeted.

```
driver.findElement(By.name('q'))
```

Then actions can be sent, such as querying or entering input and submitting.

```
driver.sendKeys('Christmas Sales', Key.RETURN)
```

The developer can visually see a window being opened, text being inputted into Google's search form, and see the actions. That represents the essential interaction for Selenium's automated web browsing. Note the extra commands By and Key. These would be imported in the beginning as well, and are extra packages from Selenium web-driver. Their functionality help abstract more granular commands for interaction.

Selenium by itself automates web-browsing. However, it becomes even more powerful when combined with testing frameworks such as Ruby's RSpec and JavaScript's Mocha. Since these web-browsing can be automated, it can be easily recorded and examined for expected output and behavior. Common cases include a user adding a post, logging in, or deleting a picture. For the report's code demonstration, we can perform a very popular test - checking if Google's Search is working. Javascript's mocha testing framework follows modern agile principles, but it needs to be initialized alongside Selenium. First to install:

```
$ npm install mocha
```

Then initialized:

```
const assert = require('assert');
```

We can then use mocha to leverage selenium-webdriver. Note the following code includes Domain Specific Language from mocha that follows standard testing syntax.

```
describe(‘Search Google on Firefox’, () => {
    let driver;

    before(async function() {
        driver = await new Builder().forBrowser('firefox').build();
    });

    it('Searches on Google', async function() {
        // Load the page
        await driver.get('https://google.com');

        // Target Search box, then send input
        await driver.findElement(By.name('q'))
                    .sendKeys('test', Key.RETURN);

        // Wait for the results
        await driver.wait(until.titleContains('Google Search'), 2000);

        // Extract title
        let title = await driver.getTitle();

       //Test
        assert.equal(title, 'test - Google Search');
    });
    // close selenium browser
    after(() => driver && driver.quit());
})
```

There are several options within Selenium’s tools. Waiting for results is not as straightforward as checking for proper text - there are also options for regex matching and presence of other elements. Mocha also allows integration for data comparisons, which is popular if the developer is examining APIs. User interaction is also much more than clicking an element and sending text.

Nonetheless this covers a typical feature with automated web-browsing and testing with JavaScript. It’s very powerful for User Interface (UI) testing, since visual elements is more difficult to quantify which makes it more difficult to test strictly. For UI developers, this enables them to test much easier across different environments while establishing a baseline.

Although this is the base case, Selenium also allows more robust features for larger projects. Other browser drivers can be summoned instead of Firefox, and it can be easily added to the test suite. Selenium also has its own local server for developers, but it can also leverage several machines and systems in networks with Selenium Grid. Since Selenium can be leveraged within these testing frameworks, other testing frameworks such as React’s Jest can be incorporated.

When combined with more mature and profound tools such as Continuous Integration, developers are empowered to build faster and test better. Selenium is wrapped into Mocha, and that can be wrapped into tools such as TravisCI for continuous deployment and that can be combined into virtual environments like Docker. All of these tools capitulate the Agile methodology promote productive software development at scale.

IV. Alternatives and Competitors to Selenium

There are a variety of alternatives and competitors to Selenium / Selenium WebDriver used in today's software industry. These alternatives boast specific advantages to Selenium which, although is the predominantly used UI automation framework in the industry, is facing competition due to the emergence of these alternatives. Popular testing tools that are favored commonly exhibit the following characteristics: easy learning curve, greater test stability, better reusability, and fewer number of dependencies.

One example of a testing tool that is gaining a lot of traction in the software testing industry is Cypress, which is a Javascript-based web automation framework. Cypress is the ideal testing tool to utilize in an Agile test-driven development environment since UI tests can be created at the same time features are developed, within the browser. Therefore, developers can test their work almost immediately after development of the feature is complete.

Also, unlike Selenium, Cypress does not require that web elements need to be in an interactable state (i.e. visible, element fully loaded) before performing a test on the element. The bulk of test failures in Selenium tend to be due to elements not being correctly found because Selenium requires elements to be in a ready state, commonly handled using element waits. However, if the element somehow fails to load, the test code will never make it to the point of testing the element, which defeats the purpose of the test. As a deterministic testing tool, Cypress handles this problem by automatically waiting for the web application to reach a ready state before performing any validations, thereby eliminating test flakiness.

Splinter is another popular web UI test automation tool that shows plenty of potential to become a top testing tool in the software industry. Splinter is actually an abstraction layer built on top of Selenium, allowing one to reuse Selenium code in a Splinter test framework. One of the major advantages of Splinter is the ability to write tests with considerably less code than the same test in Selenium would require. Splinter is developed with simplicity in mind by bundling up common automated test commands into a convenience method. A good example to demonstrate this point is filling forms. In Selenium, we would need to first find the element then send keys to it, like so:

```
element = browser.find_element.by_name(‘email’)
element.send_keys(“test_email@gmail.com”)
```

While in Splinter, the equivalent action can be performed using a one-liner:

```
browser.fill(‘email’, “test_email@gmail.com”)
```

Furthermore, Splinter uses the same web drivers as Selenium, allowing software teams to run their tests on common browsers like Chrome and Firefox especially for compatibility testing. The only drawback is that Splinter tests are currently only implementable in Python (just like how Cypress tests can only be written in Javascript), which may pose as an inconvenience for those who are less experienced in this programming language.

References

Selenium Main Repository
https://github.com/SeleniumHQ

Introduction to Selenium

https://www.guru99.com/introduction-to-selenium.html

Selenium rc and web-driver

https://www.tutorialspoint.com/difference-between-selenium-rc-and-web-driver

Webdriver Comparison

https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html

Selenium Sample Project with Node.js

https://github.com/bmshamsnahid/Automation-With-Selenium-And-Node.js

Automated Google Chrome Test

https://itnext.io/automated-ui-testing-with-selenium-and-javascript-90bbe7ca13a3