Automated Web Software Testing With Selenium

Regina Ranstrom
Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556

Abstract:

A common question for software testers and developers is, when do we automate testing and when is manual testing sufficient? In the following report I will describe the benefits of automated testing when applied to a web application. I have also come to a conclusion about what stage in the development process it is best to begin testing and what tools are particularly helpful. This experience report will mostly describe the standard environment for testing with Selenium Integrated Development Environment (IDE). It includes lessons I learned about script writing, using the Selenium Integrated Development Environment, and building sustainable, efficient tests that will save time testing in the long run and ensure that the product is tested often. I will highlight the areas that a tester will most likely need to aid Selenium, as Selenium is not always able to sufficiently record and playback user interactions with the browser. I will also discuss what areas of a web application would benefit most from being tested with a tool like Selenium and how to be sure that Selenium has tested the functionality of the most crucial aspects of the program.

Introduction:

A high demand is emerging for methodologies and tools for the quality assurance of web-based systems. [5] There is a great responsibility for developers and testers to ensure that web software exhibits high reliability and speed. Somewhat recently, the software community has seen a rise in the usage of AJAX in web software development to achieve this goal. [9] The advantage of AJAX applications is that they are typically very responsive. The virtual Emergency Operations Center is an Emergency Management Training application that requires this level of interactivity. However, before Selenium IDE, there was no open source testing tool that could handle the amount of JavaScript present in AJAX applications, and even give the tester the freedom to add their own features. [3] Since web software is so frequently modified, the main goal for any test developer is to create sustainable tests. Unlike shrink-wrap software applications, web-based applications do not have to be sold or distributed when updates are made. Compatibility and interoperability are also urgent quality features and cause problems that are more serious than with traditional programs. So, the question is, can these types of applications be tested most efficiently via automation or simply by manual testing? Additionally, what is the best approach to creating maintainable test suites?

Background:

The goal of this research was to successfully test the functionality of a web application called the virtual Emergency Operations Center or vEOC. The vEOC is a virtual training environment for various emergency response managers and coordinators. It also serves as a research tool for cognitive scientists to study the decision-making process under emergency conditions. This focus of the project is to build a computational discovery infrastructure to examine the decision-making and organizational complexities that arise from events associated with the management of disasters, such as recovering from a hurricane. [14] The vEOC is comprised on the client side of HTML, JavaScript, AJAX, and Reverse AJAX and on the server side of PHP and MySQL. The application is designed to be very interactive, as it involves the communication between multiple users. (i.e. Assistant Mayor, Police Department, Fire Department, etc.) It also requires quick means of giving and receiving updates (reports, boards, logs, etc.) so that these users may relay information efficiently. Naturally, an AJAX application was the development approach as these applications typically data is exchanged asynchronously between server and browser. Therefore, when updates are made, to individual parts of the application, those changes will be displayed without requiring a full-page reload. This provides for quick, responsive environment as would be necessary in any emergency situation.

Automated vs. Manual Testing:

Automation is the best way to make sure that tests are run often. Automating tests also means that developers can develop a test script before the program is written. This is the ideal process because the developer can then confirm as soon as the program is written that it does what was expected of it with the click of a button. Each time you write an automated test, you might have missed the chance to perform 3 manual tests. [4] In general, creating the original automated test script with encoded verifications that test certain elements of the program can often be more intricate than simply clicking the links and verifying with one look that everything has loaded properly. That being said, with the knowledge that these scripted tests can be run over and over again, it is best to trust automation to do the job.

Consider the fact that the tester, who may or may not be involved in development, is not always able to pick up on the finer details of the program that might have changed. Instead of having to remember every aspect of the program and make sure everything is still in order, the developer can trust their previously constructed script to do so. In future cases, the need to manually test the program would no longer be necessary. Automated testing comes in especially handy with web software. If you were testing a shrink-wrap product whose product direction and code base has changed wildly in the last few months you may not even have time to try all the obvious tests once. In the time you would spend automating your tests, you could find at least one completely new bug. In this case, the cost of automation is high. There are a few questions one should ask oneself when determining whether or not to automate tests. Is the feature a core/critical feature? Is the test tedious and error prone? Will my test script verify results via a fragile method (screen capture) or a sturdy method? Is the feature I am trying to

automate undergoing a lot of churn? When this script fails, how easy will it be for me to investigate the failure? One thing that will cause test scripts to fail just about faster than anything else is the product changing. This is why refactoring tests is so important.

Tools suited for Web Testing:

Web testing software such as Canoo WebTest [17] and HttpUnit [18] cannot handle complex in-page JavaScript. These programs only simulate Firefox' or Internet Explorer's way to execute JavaScript. This means that it does not work as good as in the real browser. Selenium overcomes these problems because it is JavaScript based and runs directly in the browser as an add-on. Selenium runs inside of the browser in JavaScript and controls the browser by giving it commands. But when do we use Selenium? When is it most beneficial? A web application, similar to the vEOC is a prime example of when to use an automated testing tool like Selenium. Using a Graphical User Interface (GUI) capture/replay tool like Selenium that tracks your interactions with the product and builds a script from them makes automation relatively cheaper (or more efficient). Although, traditionally, capture/replay tools seem quite costly, when you consider having to recapture a test from the beginning after a mistake has been made, they save a lot of time. The time spent organizing and documenting all the files that make up the test suite, the aggravation of finding and working around bugs in the tool are all reasons that many have steered away from this method. However, Selenium IDE, overall, makes refactoring and fixing tests fairly easy. If the wrong command has been recorded, there is a feature in Selenium that gives a menu of other commands that might be suitable. This saves a lot of time overall.

Selenium IDE:

When it comes to automated testing tools, it seems that in fact Selenium will get the job done best for the lowest cost, (time-wise and financially). Previously, the common statement with regard to testing JavaScript applications was “write once, test everywhere.” Selenium makes this task less of a nuisance as it can be used across multiple platforms. Writing tests with Selenium makes it very simple to perform tests often and maintain them. It is clear that automated tests of these kind save a lot of time in the long run. Using browser recording playback and JavaScript conveniences such as loops, you will find that Selenium, and tools like it, will save plenty of time when it comes to testing web applications.

Selenium IDE in an integrated development environment which tests code while integrating browser activity. The main goal of writing these types of tests is to mimic user actions to determine if the database and web server are behaving as expected as they carry out the users’ commands. Verifications are the heart of selenium tests. These commands are a way of knowing what part of the application is being tested and what the user expects the browser to present. Writing tests is relatively easy with selenium. The end product of a selenium test can be converted to any language, however, by default, selenium scripts are simply a combination of “Selenese” commands in an HTML format. However, by downloading or creating a user extension, as I did, JavaScript commands may also be implemented. User extensions themselves are created

with JavaScript by adding methods to the Selenium object prototype.

Although the record and playback tool is what makes Selenium so easy to use, it only gives the test scripts a rough framework that you are usually required to alter. In reality tests do not necessarily have to be written this way. They can be written before or after the code has been completed, which is typically very convenient for developers. [6]

As mentioned before, the vEOC is an application that implements reverse AJAX, which is simply an AJAX design pattern. It uses HTTP connections to enable communication with very little delay between a web server and a browser. Basically it is a way of sending data from client to server and a mechanism for pushing server data back to the browser. AJAX, short for Asynchronous JavaScript and XML, is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. This is meant to increase the web page's interactivity, speed, and usability. Like any AJAX application, when you click a link in the vEOC web site, it will not cause the entire application to reload, only the relevant information. As it turns out, Selenium is the perfect tool for testing this type of application as it allows the script developer to demand it to wait for certain elements to load before verifying anything. When executing the "waitfor" command, Selenium will suspend the execution of current test case and wait for the expected value. These are just a few of my many examples that make Selenium very well suited for AJAX applications.

Testing efficiency and sustainability:

The goal of the software tester is similar to the goal of the developer. They want to transform their test code in order to bring it in the simplest possible state. This process is often referred to as refactoring. Selenium test suites make testing very practical. Test cases saved together as a test suite will be run sequentially. This makes it very easy to maintain test code as specific iterations may be added and removed as necessary. For example, if a new feature were added, it would not be necessary to write an entirely new test suite.

When writing these tests, elements may be identified using many different locators. By downloading an add-on that allowed me to look at the Document Object Model (DOM) of each webpage and the Xpaths of each element, I was able to write, short, efficient tests with a minimal amount of data entry. This came in very handy when I wrote a short script that tested every Login instance. The vEOC's login screen prompts the user to enter two values for a Role and an Individual. Creating over 100 test cases for each of these instances would not only be tedious, but it could require making 100 different modifications if the code changed. What I wanted was to create a sustainable test for this crucial part of the software. I realized this test could be done using one simple test case involving a loop function. Additionally, I found that writing the tests using X-Path and DOM locators was a much better approach to writing sustainable tests considering that titles often change and ID numbers are much more conducive to be used in loop functions. In other words, every dynamic page includes input forms. I found that, in many cases, when testing these types of pages it is best to define elements based on their location in the form, rather than by the name they are given. Since this type of test

is used often, (whenever there is a drop-down menu) it could also be reused easily by just changing the numbers correlating to the number of options in the menu (or menus). Again, this technique saved a huge amount of time because of the extensive number of options and is easily modifiable.

Because there are so many different scenarios, test engineers must focus their test activity on the portion of the application that exercises the majority of the system requirements and find the most practical ways that the user might use the software.

Limitations:

Something that might have made Selenium better would be if it was a bit easier to avoid test duplication. Ideally, each test case should only have to be written once, and should be able to be extracted from larger test schemes which it might be a part of. For example, the tester has to make the same verifications each time they visit the homepage or navigate through it. If these verifications were their own test case, they could be written and modified separately and embedded in several different test scenarios. However, in Selenium IDE, it is often necessary to rewrite these checks for every test, which imposes the risk of missing some verifications and makes modification much more tedious. Selenium allows the tester to create different test cases however it is not evidently possible, using the IDE, to embed certain test cases in others.

As mentioned, the DOM is an integral part of test script development. [12] Every Ajax web application is constructed around the structure of the DOM to be manipulated by message handlers. Because of the infinite options, it is often necessary for the tester to map out the program and determine the most crucial traversals for functionality. The Document Object Model is the way JavaScript sees its containing HTML page and browser state, and therefore, is very helpful to the tester who wishes to mimic this behavior. It serves the tester well as a clear map of the program and its elements. I found that access to the DOM very convenient for planning and writing test scripts. However, Selenium does not make this interface available to Selenium users. It is also quite difficult to identify XPath locators. To solve this problem I ended up having to install two more add-ons, X-Pather and DOM inspector. [4] [12] Ideally, these features should be included as a part of the Selenium Add-On.

Conclusion:

When developing web software, the ultimate goal of the tester or developer is to ensure that the application is tested often and thoroughly. More often than not, creating automated test scripts is the best way to be sure that this goal is accomplished. In particular, the developer wants to be sure to create maintainable test scripts that will last through the many changes that applications undergo. If modifying or refactoring the test script does become necessary, there are ways to make sure this job is done quickly and correctly. The main way is to avoid test duplication. By keeping specific tests self-contained, they can be reused in several places and only one modification would be necessary for all instances. An Open Source test tool, Selenium IDE has many advantages, including an easy to use record and playback tool, and the ability to test JavaScript inside of the browser. However, as test cases can only be run sequentially and

cannot be embedded in one another in the IDE, writing higher level test scripts can sometimes be difficult. In addition, the log, which reveals whether or not tests have run successfully, evidently cannot be exported. However, all in all, the user friendly nature and the ability to customize commands via user extensions make Selenium IDE an ideal test suite development environment in many ways.

References:

[1] http://seleniumhq.org/docs/03_selenium_ide.html#id4
[2] http://www.infoq.com/articles/testing-ajax-selenium
[3] C. Titus Brown, Grig Gheorghiu, and Jason R. Huggins. "An Introduction to Web Applications with twill and Selenium" 2007.
[4] Brian Marick. "When Should a Test be Automated?" 1998.
[5] Filippo Ricca and Paolo Tonella. "Analysis and Testing of Web Applications." 2001.
[6] http://xpath.alephzarro.com/
[7] http://www.hpl.hp.com/techreports/tandem/TR-87.3.pdf
[8] Antawan Holmes and Marc Kellogg. "Automating Functional Tests Using Selenium." 2006.
[9] Ali Mesbah and Arie van Deursen. "Invariant-Based Automatic Testing of AJAX User Interfaces." 2009.
[10] Grig Gheorghiu. "A Look at Selenium." 2005.
[11] Sebastian Elbaum, Srikanth Karre, and Gregg Rothermel. "Improving Web Application Testing with User Session Data." 2003.
[12] http://www.w3.org/DOM/#what
[13] http://homepages.cwi.nl/~leon/papers/xp2001/xp2001.pdf
[14] Arie van Deursen, Leon Moonen, Alex van den Bergh, Gerard Kok. "Refactoring Test Code." 2001.
[15] Irma Becerra-Fernandez, Michael Prietula, Greg Madey, Domingo Rodriguez. "Project ENSAYO: A Virtual Emergency Operations Center for Disaster Management Research, Training and Discovery." 2007.
[15] Johnson, T (2010). vEOC Usibility Test. July 1, 2010.
[16] http://webtest.canoo.com/webtest/manual/WebTestHome.html
[17] http://httpunit.sourceforge.net/

Appendix:

Here is a test that checks the "delete inject" feature, a part of the After Actions tab.

AAdelete		
open	/veoc/mainpanel.php	
click	link=After Actions	
waitForPopUp	aar	30000
selectWindow	name=aar	
click	link=Open Report	
clickAndWait	//input[@value='Open Report']	
click	link=Add Item	
click	link=Delete Item	
type	injectid	4
clickAndWait	submit	

This tests verifies all elements on the System Manager Main Panel.

Login		
open	/veoc/RegularLoginSM.php	
type	name	testuser
type	password	password
clickAndWait	//input[@value='Log In']	
selectWindow	title=vEOC 2.0	
verifyTextPresent	vEOC 2.0 Exercise Developer Console	
verifyElementPresent	link=Exercise Development	
verifyElementPresent	link=Logout	

verifyElementPresent	link=Exercise Evaluation	
verifyElementPresent	link=Exercise Control	
verifyElementPresent	link=Links	
verifyElementPresent	link=References	
verifyElementPresent	link=Training Reports	
verifyElementPresent	link=Handbook Developer	
verifyElementPresent	link=Script Developer	
verifyElementPresent	link=Evaluation Metrics	
verifyElementPresent	link=FEMA Training Center	
verifyElementPresent	link=Miami-Dade Map	
verifyElementPresent	link=Miami-Dade Resources	
verifyElementPresent	link=Player Reports	

This test iterates through all Login instances, represented first by their index in the "roles" menu, and then by the "individuals" menu.

LoginAll		
getEval	Roles = new Array(3, 4, 4, 4, 17, 17, 16, 20, 8, 4, 2);	
store	0	i
store	1	in2
store	1	in3
while	${i} < Roles.length	
storeEval	Roles[${i}]	individuals
while	${in3} < ${individuals}	
open	/veoc/RegularLogin2.php	
verifyElementPresent	//h3	
verifyElementPresent	//tr[2]/th	
type	name	testuser
type	password	password
clickAndWait	//input[@value='Log In']	
verifyElementPresent	//form/p[1]	
verifyElementPresent	//form/p[2]	
select	script	label=Hurricane
select	//select[2]	index=${in2}
select	//div/select	index=${in3}
clickAndWait	//input[@value='Enter']	
selectWindow	null	
storeElementPresent	//div[5]/div/div/div/p	playertaken
gotoIf	${playertaken}	target
waitForPopUp	MainPanel	30000
waitForPopUp	ExercisePanel	30000
waitForElementPresent	link=Logout	
selectWindow	title=vEOC 2.0	

clickAndWait	link=Logout	
label	target	
storeEval	"Player:" + ${in3} + " is taken"	message
echo	message	
storeEval	storedVars['in3'] = ${in3} + 1;	in3
endWhile		
storeEval	storedVars['i'] = ${i} + 1;	i
storeEval	storedVars['in2'] = ${in2} + 1;	in2
storeEval	storedVars['in3'] = 1;	in3
endWhile		