Transition from a rapid prototyping to programmatic test framework

Robert A Zakes and
Brenden Beaman
robzak@state.or.us and Brenden.beaman@state.or.us

Abstract

The office of the Oregon Secretary of State was getting an increasing number of help desk calls related to running our applications in different browsers and mobile platforms. Automated testing had been implemented in 2007 using Selenium IDE (a rapid-prototyping test framework) with great success as reported in a 2009 PNSQC paper.

Unfortunately, the IDE version of Selenium is a Firefox plug in and consequently only tested our Web applications using Firefox.

We needed a way to run our automated test scripts in multiple browsers and on the most popular mobile platforms. Selenium WebDriver, the follow on product to Selenium RC has this capability, but requires more technical programming skills to build and maintain scripts.

This paper describes our transition from a rapid-prototyping test framework, Selenium IDE, to a programmatic test framework, Selenium WebDriver. Specifically, it will compare the strengths and weaknesses of both tools in terms of:

- Capabilities and adaptability
- Time investment, metrics for script development
- Script maintenance
- Technical abilities required to become proficient
- Scalability
- Load testing

The paper will discuss our evolved strategy for conversion of a large inventory of IDE regression test scripts and when and where we have decided not to convert.

Biography

Bob Zakes is a Requirements and Testing Manager for the Oregon Secretary of State.
Bob has over forty years of experience in project management and software requirements and testing. Bob has worked for IBM as an instructor and marketing and systems engineering manager, National Retail Systems West as a product manager, Hanna International as Engineering and Quality Assurance Manager and the State of Oregon as a project manager and his current position. He has been responsible for several successful system implementations at the State. His current primary project is Oregon's Central Business Registry.

Brenden Beaman is a Testing Analyst for the Oregon Secretary of State.
Brenden has ten years of experience in software development and testing. Brenden has worked at Hewlett Packard as a quality assurance lead, and the State of Oregon as a testing analyst as his current position. His current primary project is Oregon's Central Business Registry.

1 Introduction

1.1 Our Agency Mission

The Office of the Oregon Secretary of State runs several of public-facing software applications. Most of these are complex, high exposure web apps, with front-end interfaces as well as searchable databases. To put this in context, the applications are integral to the integrity of Oregon's politics, business, financial condition and history. The applications include Oregon's Central Voter registry and election system, ORESTAR for filing all state candidates for elections, recording all political action committees, campaign contributions and expenditures and the Voter Pamphlet. The Central Business Registry serves as the one stop site for registering businesses and getting registered with state regulatory and licensing agencies. Other applications record notices of security interests in moveable and personal property, statutory liens and warrants. Also included are the registration of the States Notaries Public and indexes to the Oregon's Archive. The Office is also the auditor of all Public accounts. More information about the Office, and the current Secretary of State, Kate Brown, can be found at sos.oregon.gov.

1.2 Our Testing Automation Progress

To meet the challenge of testing these applications we started automated testing seven years ago. This was primarily driven by the need to maintain quality testing in an environment of accelerating application growth and limited testing resources. Selenium IDE was selected as the test automation tool due to its ease of use, wide acceptance, and open source being all we could afford. The initial success was reported in a 2009 PNSQC paper, Some Observations on the Transition to Automated Testing. We started automated testing concurrent with our shift to Java web development. A very agile methodology evolved where scripts have closely followed system development. We have found this to be a significant benefit of automated testing, and we believe agile is dependent on automated testing. Since we started, we have built about one hundred and fifty test scripts for most agency applications.

We build three types of scripts, quick (aka smoke, happy path or steel thread), full regression and special scripts for load testing (light), interface testing, performance tuning using timed loop scripts, timeout testing, etc.

Web Application	Quick Scripts	Regression Scripts	Special Scripts	Total	Lines of Code
Central Business Registry	**7**	**5**	**32**	**44**	**31,526**
Uniform Commercial Code Lien Filing	**1**	**7**		**8**	**7392**
Notary Registration	**1**	**1**	**2**	**4**	**2860**
Elections Reporting and Voter Registration	**15**	**17**	**30**	**62**	**20867**
Municipal Audit Report Billing and Filing	**1**	**1**	**3**	**5**	**3850**
Identity Management (user authentication)	**7**		**12**	**19**	**3212**
Historic Records	**1**	**1**	**3**	**5**	**1408**
Cashiering and Revenue Accounting	**1**	**4**		**5**	**4,246**
Totals	**34**	**36**	**82**	**152**	**76,021**

As soon as a build is deployed, we run our scripts, log and report results. The scripts are run on a bank of five PC's each with Selenium IDE. When an error occurs, the script stops and the failing instruction is highlighted in red. It can be rerun and analyzed at that point.

1.3 Anatomy of our Quick and Regression Scripts

The quick scripts are run first after each build and frequently find issues prior to running our full regression such as a missing library or an interface that was not updated. Our model for these scripts is

- Create a script for each major transaction so all major navigation paths are covered,
- Each script is cradle to grave, and will test all interfaces if possible. Here's a sample scenario.
  - Start with customer sign in
  - Customer completes application, makes payment and submits
  - Internal reviewer rejects
  - Customer reopens, fixes and resubmits
  - Internal reviewer approves
  - Customer searches and verifies completed status
- Provide switches to change test environments and stop at key points in the script for debugging,
- Complete all fields with realistic data, e.g., first, middle, last name and suffix,
- Use a date and time stamp or token to append to the primary name to make completed transaction unique
- Choose the predominant path through dynamic pages
- Provide logging for run date and time, duration, environment, reference to entities saved,
- Typically run for one to two minutes.

The regression scripts are spawned from the quick scripts and are also run after each build. Our model for these is

- Include all of the features of the quick scripts
- Validate content on all pages
- Test all default values
- Test all paths for the function or transaction
- Test all paths for dynamic pages
- Test all validations, mandatory fields, options, min/max sizes, ranges, allowable characters and character sets, valid dates, emails, phone numbers, addresses, etc
- Test interface exception handling, e.g., payment failure types
- Typically run for twenty minutes to an hour.

1.4 A Change in Culture

As automated testing became part of our development process we noticed a culture change. Our user department testers depend on these scripts and they can now focus only on new functionality and ad hoc testing. Becoming a tester in our user departments is recognized as privilege and an activity leading to advancement. It used to be drudgery. Our developers depend on our script building as part of our agile development to catch inadvertent errors early and to reduce unit testing. Management depends on quality being developed throughout the development and maintenance cycle and the low number of production issues reported.

1.5 A Perfect Storm

We started to recognize deficiencies in our test automation methodology. It started at our help desks manned for each of our applications. We were getting an increasing number of calls regarding browser/platform specific issues. A page would not open in a current version of IE. A button was missing in Safari. Table headings were wrapping on an iPad. Selenium IDE is a Firefox add-on, and only works in that browser. While our functional test coverage was excellent, we did not have a good way to look for bugs and compare differences between different browsers and platforms. We had depended on our users testers to test with IE to address the Firefox limitation. When we started we only had to worry about IE and Firefox, but this was no longer the case. Google Analytics showed our users were accessing our systems with a wide distribution of browsers and platforms. The Selenium IDE limitation to Firefox became a major issue.

We also had a few situations where our applications went down due to high traffic. Selenium IDE is not a load let alone a stress test tool. At best, we could get ten PC's running a loop script but this would only generate a fraction of the load required. We tried JMeter load testing tool and found it was useful for testing our application server, but did not stress the database or external interface connections. We could not correlate the test results to our application behavior at heavy loads. We needed a load/stress tool that would replicate our transaction loads and give us the ability to stress to determine our peak stress level.

About a year ago a decision was made to upgrade the Central Business Registry system to improve usability, streamline design, and modularize the sections designed for specific agencies. That decision would obsolete the CBR test scripts, the largest and most complex in our library. That forced the question, should we use Selenium IDE for the new application?

1.6 What tool to use?

There were a lot of alternative test frameworks we could consider, and we did look at several, but the choice of Selenium WebDriver was fairly simple. Selenium WebDriver is the companion product designed to provide what is missing in IDE. There are tools to partially convert IDE scripts to WebDriver. WebDriver is open source and no procurement was necessary (a big factor in the public sector). It is widely accepted and has a very active user group. We have been pleased with the stability and reliability of Selenium IDE and it appears the Selenium Project support will continue.

2 Selenium IDE versus Selenium WebDriver

Once we had focused down on the two products, we started drilling down into the comparative details of them. We wanted to assess which existing IDE scripts would benefit from being re-developed into WebDriver scripts. In addition, we wanted to determine which product would be used for new script development going forward, and when. To help make these determinations, we started to compare and contrast the various advantages of the two products, as detailed in this paper.

2.1 Browser support

Selenium IDE, a Firefox plug-in, is obviously limited to testing with Firefox. Selenium WebDriver offers support for Firefox, Internet Explorer, Chrome, and Safari with very little added effort. With a moderate amount of additional setup, it can also execute your test scripts against an iPad or Android tablet. Being able to easily validate web applications play nicely with the myriad of browser idiosyncrasies was a huge selling point in this regard. One of the first scripts developed was a simple page viewer that we used to test our agency's new website. The script opens pages from a list of URL's for a variable length of time in any browser selected. Content managers were able to test the site on the browsers and tablets mentioned above. Since the script opens every page without any navigation, the tester can focus on the layout. The script was a big help in rolling out a near perfect new website.

2.2 Script portability

The WebDriver API built into a java codebase, can be compiled into an executable, standalone JAR file. These can also package up any required resources such as images, into one file that can be shared or linked to. They're platform independent, so a Mac or Linux user can run them. No install files are required; you can even run them over a network without copying the file locally. This process makes the scripts highly portable. The page viewer mentioned above was packaged in this way.

One of our agency goals has been to encourage content experts to be able to utilize the automated test scripts themselves, in addition to the testing team. These staff members are highly familiar with the business rules and details of their respective departments, but not necessarily technically proficient with testing software. They may or may not have the Firefox browser and/or plugin installed, but if given access to a standalone executable script, they are more prone to use it. The high portability and ease of

use of these jar files was a large factor in helping us determine which IDE scripts needed to be converted into WebDriver scripts.

2.3 Switch setting versus an options dialogue

Setting up variables to manage test behavior in the IDE was somewhat primitive. We initialized global variables that held values based on how we wanted the test script to run. For example if we wanted the script to test the DEV environment rather than QA, we would save a string value to a variable which is logically checked throughout the script. Before running the script, you would move the command that sets the wanted environment last. In the example below the value is set to "qa". A gotoIf command executes other code based on the value of that string, as shown below.

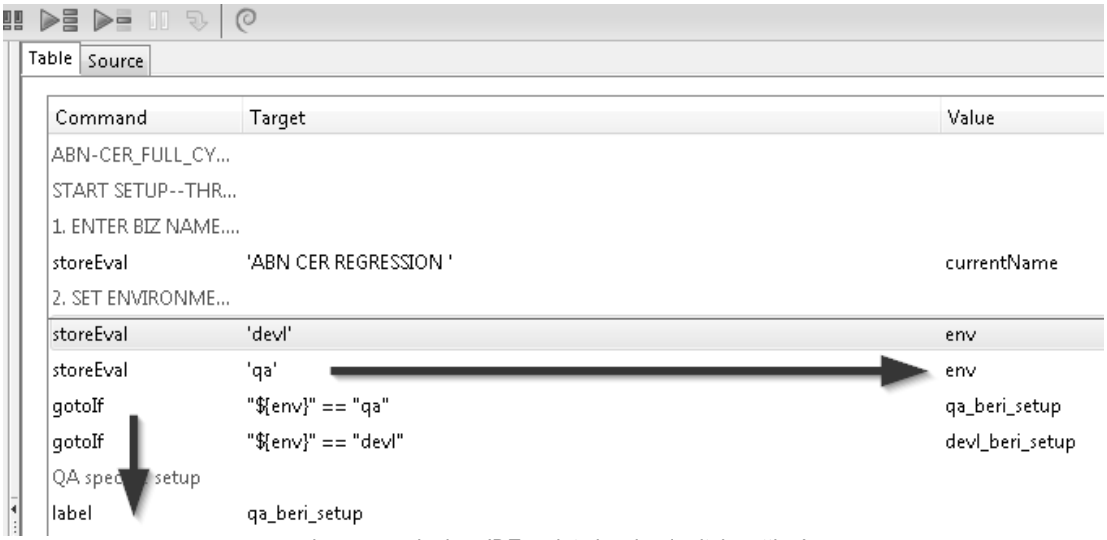

Figure 1: *Selenium IDE script showing 'switch setting'*

The Selenium WebDriver API is built on top of an existing program language, as additional API. We used Java in our implementation. Because the existing Java API's are also included, we were able to develop user interfaces using the Java Swing libraries.

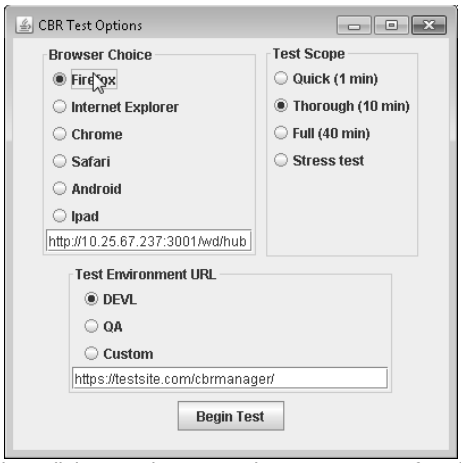

Figure 2: *Options dialog used to set testing parameters of WebDriver script*

This allowed us to easily change the testing parameters before executing the test. Not only did this make the test scripts more powerful, but also very user friendly. Scripting is also considerably easier since you

are not limited to gotoIf. Again this capability allows us to build test scripts that can be easily run by non-technical staff, even project managers.

2.4 Data analysis functionality

Using an automated script to test an application is one thing, but what do you do with the results you gather? After your script has executed, how do you know if it was successful or not, and if there were any bugs detected? Gathering and displaying test results is a critical aspect of automated testing, and testing in general.

Data gathering and results tracking is possible in the IDE. Throughout the script, you can store variables with various pieces of data, and then display them at the end with an alert window showing various metrics and statistics

Figure 3: *Selenium IDE script showing a test result dialog*

Although this is somewhat open-ended and up to the developer, the IDE has no native ability to write to files. The WebDriver API, when extended into java code, allows much more control over data storage, tracking, and reporting. Using Java Swing elements, we built a test harness into our scripts that would track and update test results in real time. We could also write those results and other metrics to a log file for later analysis. This allowed much more control and organization of defect tracking and development progress.

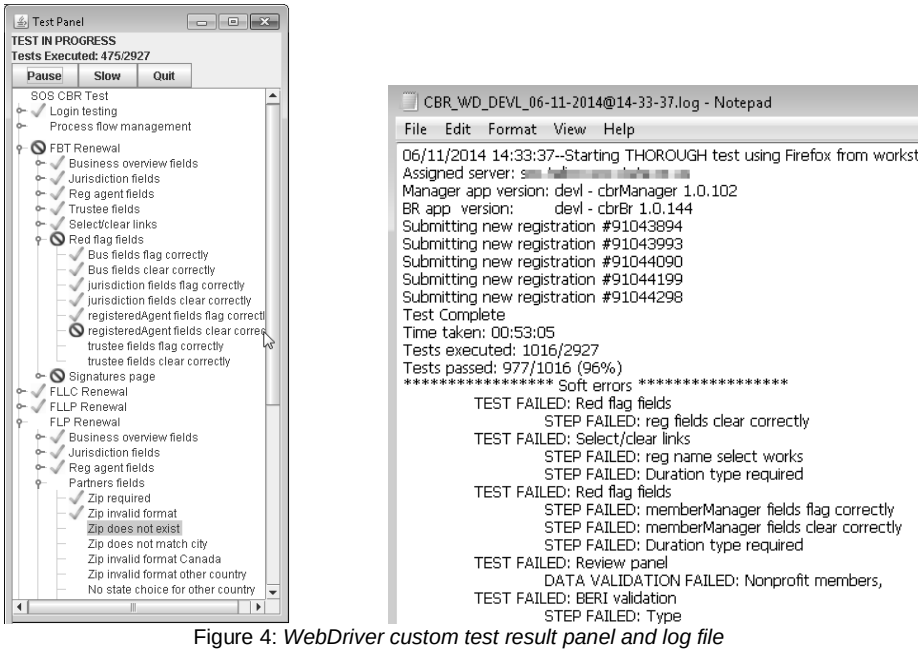

Figure 4: *WebDriver custom test result panel and log file*

2.5 Script developer skill set required

One advantage of Selenium IDE is the ease of use and speed at which someone can develop a test script. The IDE itself is fairly straight forward and easy to use, and there is a vast knowledge base of documentation available to assist script developers. By far the biggest advantage is the record function. When enabled, you browse the target application as you would normally, and the IDE will automatically generate script code based on what you are clicking on, interacting with, etc. Users can also right click and select test commands from a menu. The recorded script not only navigates the application, but can run various tests as it goes as well. The script can be played back at any time and at any place to watch how it interacts with the application.

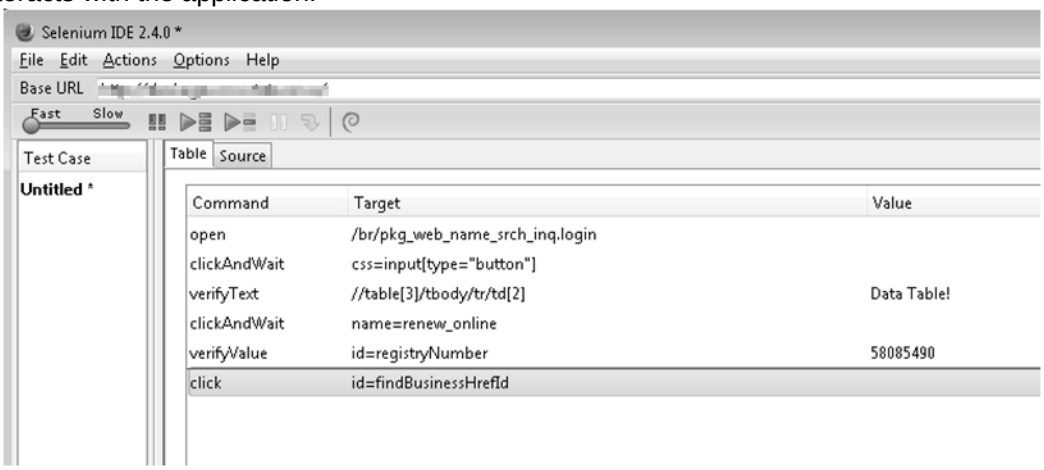

Figure 5: *The Selenium IDE recording a script*

There are some downsides of recording; Scripts generated in this way tend to be fairly 'raw' and need additional refinement; they may not be very readable by testers without comments/documentation. Verifying results of your actions aren't built by the recording process, and need to be added manually. (Verifying that a checkout screen includes all the items you added to a shopping cart, for example) There could also be timing issues, where the script sends a command before a web element has loaded in the browser. A major task in refining an IDE script is adding pause or waitfor commands to resolve timing issues. Recording is dependent on the IDE's locator strategy. Although the IDE can normally record elements using the ID, name, XPath or CSS, occasionally elements are hidden from the IDE's locator within Ajax code or they are dynamic so the recoded element names are useless. We have used JavaScript to overcome these issues, but this requires programming experience so that skill must be available or the scripter needs to search the Selenium blog.
To address these issues, experience with Selenium IDE is required so scripts can be fine-tuned with pauses/waits, validations, and comments that can help it run completely and consistently.

An experienced test engineer can create a detailed IDE script in a matter of hours. None of our developers were familiar with Selenium initially, but learned it quickly and easily. Perhaps even more importantly, a non-technical user can use the IDE to create a script fairly quickly as well. This rapid prototyping can be highly advantageous when a quick script is needed for debugging. This is very useful since both the browser session and script are both active after a failure, so the page can be restored and script can be restarted immediately before the point of failure. It's also useful for scripting utility tasks: create a new user in all test environments; finding and deleting old test data; and creating large transactions or batches to test reports.

While a non-technical user can start scripting quickly, the scripts will be somewhat limited. The scripts we have listed above have many gotoIfs for flow control and some are nested. (These are very difficult to read and debug) The locator strategy for recording elements on a page is good, but sometimes JavaScript is needed. We have probably pushed the IDE beyond its intended purpose, but have overcome limitations with experience using the tool and some help with JavaScript. The key point is that all have been developed by an experienced tester without programming skills.

Selenium WebDriver, on the other hand, has built a package of API's that are extended and built into several established programming languages. There is a higher barrier of entry to these scripts. Test developers need to have some programming experience in addition to testing analysis/quality assurance experience. We chose to use Java as our language, so as a result needed a java programmer to be able to adapt and build WebDriver scripts.

It is also very helpful for a WebDriver developer to have some experience using Selenium IDE. Some of the testing concepts are similar, and it's also possible to convert/translate scripts from IDE into Java/WebDriver code.

2.6 Higher Maintenance and Complexity due to IDE's limited instruction set

We have pushed the envelope of the IDE by using plugins (See Selenium site for a list of the plugins) and JavaScript. This has enabled us to build extensive regression scripts without a programmer except for some help with JavaScript to perform operations outside the IDE's capabilities. The downside is the resultant complexity and high maintenance. WebDriver and Java provide the opportunity to use the power of the API and Java to perform complex functions natively and to build well structured code. Some examples will show the contrast.

- **Flow Control**
  The IDE uses a plugin that provides goto, gotoIf and While commands for flow control within a scrip. We have used these for subroutines, looping, skipping tests that fail awaiting fixes and conditional navigation. These scripts can be difficult to follow and to debug. Java provides the ability to create methods, has more powerful flow control capabilities and it is native to the language.
- **Date and Time functions**
  In IDE you need to execute JavaScript to create dates and times and use JavaScript for date and time calculations. Again these are more powerful and native to Java.
- **Data Manipulation**
  There are no native commands for data manipulation or formatting. You need to use JavaScript to convert case, pad, slice, etc. Again these are native to Java.

2.7 Selenium IDE's editor is limited compared to a full development language IDE

Selenium IDE's editor is designed for online scripting and inserting test commands and comments. As such, the IDE is very easy to create and run test scripts. It has a very interactive run time environment and it has supported plugins primarily focused at run time and logging. It has a native test suites build facility. The editor does have a command auto-complete capability and it does provide the ability to cut copy and paste commands. However, it is a very limited editor, e.g., it does not have a find or replace. It is not a complete development environment compared to Eclipse for Java. Full function development environments offer full editing capabilities designed to build navigate and manipulate code. One result is the ability to build one WebDriver script to perform all of the testing done with twelve IDE scripts.

2.8 Load testing

Selenium IDE is not designed for load testing. We have done light load testing by writing tight scripts focused on iteratively testing an interface, a query or database update and then running the script on multiple PC's or VM's. This is awkward and in our agency getting ten PC's is about the limit. WebDriver provides a much better approach by creating jar files that can be replicated and run. We have run seven concurrent sessions on one PC and could run more until memory is maxed. There is also a Selenium project, Grid, which is designed to run and manage multiple sessions across several servers. Grid has been merged into the current Webdriver. We do not have experience with this facility, but this will be our choice for future load testing.

3 Conclusions

After adding one and a half java developers to our test team and building and running WebDriver scripts we have formed some conclusions.

- The IDE scripts that we have built over the years continue to be useful for several reasons. For older applications in maintenance mode, these scripts function well and require minimal maintenance.
- For applications where we are upgrading or adding new functions, the IDE scripts can continue to test the portions in maintenance mode.
- For those applications where we are continuing to use the IDE scripts, converting one or more of the IDE quick scripts to WebDriver provides multi-browser/platform testing and load testing. It also provides a portable JAR file for any user.
- The IDE scripts are still useful for positioning, i.e., getting to a particular page and field consistently in an application is valuable for debugging.
- When converting an IDE script to WebDriver, the existing quick scripts serve as templates to design the new. The IDE regression scripts can be mined to capture all relevant test cases.
- Building a WebDriver script from scratch takes about the same time as converting an IDE script to WebDriver.
- For new software development, we are using WebDriver to develop more advanced scripts. The ability to test multiple browsers, manage the code base, display and save test result metrics and portability outweigh the ease-of-use benefits of the IDE.

For now at least, Selenium WebDriver fits the development and testing needs of our agency best.

References

Selenium Project website (http://docs.seleniumhq.org/)

JMeter (http://jmeter.apache.org/)

Eclipse (https://www.eclipse.org/)