2009

PACIFIC NW

SOFTWARE QUALITY CONFERENCE

MOVING QUALITY FORWARD

October 27-28, 2009

Conference Paper Excerpt

From the

CONFERENCE PROCEEDINGS

Some Observations on the Transition to Automated Testing

Robert A Zakes

Requirements and Testing Manager
Information Systems Division
Oregon Secretary of State
robert.a.zakes@state.or.us

Abstract

We started on our test automation effort about two years ago when we realized testing was slowing our application development and our agency faced monumental development projects.

After some research we decided to use Selenium as our test automation tool and started scripting. Lisa Crispin's workshop at the 2007 Pacific Northwest Software Quality Conference (PNSQC) gave us some valuable insights and several attendees validated our choice of Selenium. We now have extensive regression scripts developed for most of our systems and run them religiously for every build.

This paper relates our observations along the way. We found test automation had many uses besides building regression scripts, and they will be described in this paper. It will also describe the process of how to benefit from the maintenance of regression scripts for a new release. In addition, this paper will examine some lessons learned, describe our focus for ad hoc testing, and present some metrics of manual *vs.* automated testing.

Biography

Bob Zakes is currently the Requirements and Testing Manager for the Oregon Secretary of State's office.

Bob has over 40 years experience in project management and software requirements and testing and has a BS in Engineering along with an MBA from the University of Illinois. He has worked for IBM as marketing representative, instructor, and marketing and systems engineering manager; National Retail Systems West as product manager; Hanna/Sherman International as product and engineering manager; and the State of Oregon, where he has served as project manager for several successful system implementations. He is currently responsible for testing Oregon's Central Business Registry, the Public Records System, and ORESTAR (Election Reporting).

How We Tested Prior to Automation – Manual Scripts

Our earlier testing procedure started with a test plan that outlined test cases and combined them into test scripts. Initially, these scripts served as the acceptance criteria for new applications. When the scripts could be completed without error, the applications were ready for implementation. As the applications and the build process became more complex, we started using the scripts for regression testing. They were designed to prevent the age-old software development issue of fixing one thing and breaking another.

Our scripts were written in Microsoft Word in a table format. A script document contained setup instructions, references to source documents if data was not embedded in the script, and one or more test cases. There were blanks for the tester name, test date and time, and other relevant variables.

Each test case included a test case number, a description, instructions, expected results, and a place to record actual results. The instructions described every field entry and every mouse click. The expected results described what should happen after each instruction was executed, in terms of navigation, validation, status, *etc*.

BERI ePay 5.6 Migration – TEST SCRIPT

#	Test Case	Setup & Instruction	Expected Results	Actual Results
6	Cardholder Name Entry Page, validate edits and navigation	On Cardholder Name Entry Page 1. Attempt to continue without entering any fields 2. Attempt to continue by entering only a space in required fields. 3. Attempt to continue by only entering a first and only a last name and only a middle initial 4. Try to enter multiple characters in the MI field 5. Selectively test to see if each required address field (marked with *) is really required. 6. Enter Cardholders Name & Address, **Your Name** **Your Address** Click on **Contact Us** and **Back** Click on **Privacy** and **Back** Press **Previous** and **Back** Press **Next**, go to Case 7	1. Required fields must be entered. 2. Required fields must contain valid characters 3. The name must have both a first and last name. Middle Initial is optional 4. Middle Initial should be only one character 5. All required address fields must be entered 6. Name and Address does not change when going to another page and back.	1. Required fields ___ 2. Valid characters __ 3. Name test ____ 4. MI is one character __ 5. Address test ___ 6. Name and Address does not change ___

Figure 1. Sample Test Script Page.

The authoring process normally could not begin until there was a working prototype of the screen page. The script author would think through the setup and data requirements, often filling out paper forms to serve as the test data, and then enter the instructions and the expected results. Testing navigation and validation/error messages required checking the requirements and use case, then executing the function to complete the script. Once scripts were developed, they had to be tested. We tried to get someone other than the author to run the script to ensure it could be run and was not confusing.

Because this process was labor-intensive and few had the required talent or patience, scripting frequently was the bottleneck of our development cycle. A typical test case for a single web page would require 8 to 16 hours of

scripting. A typical script with 10 test cases would take 2 weeks to a month to prepare. Most applications required multiple scripts, and with only a few scripters this process could take months.

Once the scripts were complete they were run for application acceptance. After each run, completed scripts were collected, reviewed, and filed. The acceptance criterion was for all scripts to finish with no Priority A errors (no work-arounds) and minimal Priority B errors (work-arounds available, but onerous). Each rerun became more and more difficult to schedule and more and more stressful for all parties as deadlines passed.

Manual testing also was affected by many personnel issues. There is a wide variance in testers' abilities to test correctly. When non-repeatable errors occurred, there was always the question of whether the tester or the system was the cause. Scheduling issues came into play: different work schedules, illness, and vacations. Stress became a factor for user department testers when regular work was being deferred. Finally, maintaining accuracy when repeating the same script set was a continual problem.

Script maintenance also was very time-consuming because it was difficult to update instructions and expected results within a Word table. The focus was on editing, rather than testing. The revised scripts had to be printed and retested. It took considerable commitment to keep scripts up to date after an application was in production, and even more commitment to run them religiously after every build.

As the number and complexity of our applications increased, testing became the bottleneck. We were doing everything properly, but did not have the resource to keep up with the demand. Our agency's visibility did not allow reducing our commitment to quality, so we had a dilemma: how to streamline testing without sacrificing quality.

How We Test Now

We realized our problem was due neither to our knowledge nor to our commitment to quality; rather, it was due to our manual process of writing and running test scripts. We looked at how we could speed up the process. We looked at test automation but were apprehensive of the perceived cost and complexity of automated test systems. We asked our primary development contractor, Zirous, for alternatives, and they suggested Selenium as a solution. After some additional research we decided to use Selenium as our test automation tool. Some of Selenium's benefits include:

- Selenium IDE is not simply a recording tool; it is a complete Integrated Development Environment (IDE). Users may choose to record, edit and run scripts interactively.
- Selenium IDE is implemented as a Firefox extension allowing users to record and play back tests in the actual browser environment.
- Intelligent field selection uses IDs, names, or XPath as needed.
- A test debug mode includes the ability to set execution speed and set breakpoints.
- Tests can be saved as HTML, Ruby or JAVA scripts, or several other formats.

We started scripting with Selenium in May of 2007. You can download Selenium and add it to Firefox in 5 minutes and begin scripting. It is open source and has been developed and maintained by OpenQA. Within an hour you can create useful test scripts. We started using Selenium IDE as our primary scripting and testing tool and have been using it ever since.

Lisa Crispin's workshop at the 2007 PNSQC, "Getting Traction with Test Automation," gave us some valuable insight. Her workshop showed how test automation can become an integral part of agile development. Several workshop attendees had been using Selenium and related their success.

Although scripts can be created and run immediately after downloading Selenium IDE, we initially encountered several limitations. We had difficulty with scripts stopping during page loads, and the IDE would not record some Ajax elements or dynamically created links. At first we did not know how to use flow control, so our early scripts were huge due to cutting and pasting repetitive validations. Two years ago the extent of Selenium documentation was very limited. However, even our initial, primitive scripts offered such improvement that we forged ahead.

One procedure that helped our progress was to keep a detailed list of limitations and review them with peers and ask for help from the OpenQA forum. The solution often required Java Script. We used a Java developer who coded commands in Java Script for some of these situations.

Two years later, we have used automated scripts to speed the development process and have automated regression scripts for all our major Web-based applications. These scripts run for hours without stopping, and our list of limitations has diminished considerably. We will now describe some methodology and then cover observations.

Planning

Test planning is more critical when you are in an automated environment. Some of the issues we address as we plan for a new application or major upgrade include:

Script Design and Content. Our planning identifies what test cases will be tested in which scripts. One key decision is script size. We build large scripts that perform beginning-to-end testing; the tradeoffs of small modular scripts vs. end-to-end tests will be discussed below. Occasionally, it makes sense to create a separate script. For example, when the function to be tested is very complex, rather than add that complexity to every script we test it in a separate, smaller script. Another example is leftover or exclusive paths or processes that don't fit logically into any scripts; we combine these into an "everything else" script. For instance, we generally create a separate script to test side links like FAQs and Privacy.

Sufficiency. Automation makes it possible to test every permutation, but in some cases it is very easy to go off the deep end. We have a routine that checks for duplicate individuals based on name and address. There are ten variables, resulting in 3.6 million tests. This could be scripted by indexing through all the permutations, but how much is sufficient? We determined thirty tests provide adequate code coverage.

In-line vs. Subroutine. You can build subroutines to test common functions, or you can cut and paste these tests into multiple scripts. In the previous example, we built a subroutine to test for duplicate individuals and used it for the six occurrences. We could have copied and pasted it into each page to be tested, but the script would be very large and difficult to maintain. We have subroutines for testing individual and business names, addresses, telephone numbers, signature authentication, and other fields.

Need for Extensions. We try to identify the need for special tools or commands by exploring each new page with Selenium to verify we can identify all elements. We don't want to hold up a release or have to manually test some new function that we can't script.

Identify Changes That Have a Large Impact on Testing. We evaluate how new functions can be scripted as soon as possible. Sometimes minor changes can mean days of script rework. The key is to communicate to the team that the feature or change could delay the release and look for alternatives.

It is worth re-emphasizing the point: a little planning can save a lot of scripting.

Building Scripts

Open Firefox. Open Selenium. Selenium starts with Record Mode on. Sign on and open the first page. Complete each field and save. Selenium has created "type" commands for every entry field and "select" commands for every click, and we have just created a "happy path" script (one that goes through the page with minimal entry and no errors) for this page. This script now can be stored and repeated. See sample.

Sample Selenium Entry Script		
Command	Target	Value
waitForTextPresent	Authorized Representative Information	
type	indvFName	Aaron
type	indvMName	Authorized
type	indvLName	O'Representative
clickAndWait	btnContinue	

Note: The open, sign in and address entry are not shown.

Open a page and save. Normally, several validation messages for all the mandatory fields will be displayed (e.g., "First Name is required."). Click on each validation message and select the command "verifyTextPresent" from Selenium's popup command menu. Selenium records the validation for each error message. This process takes minutes. The result is a script for the page's mandatory field checks. It can also be saved or combined with other scripts.

Sample Selenium Validation Script		
Command	Target	Value
waitForTextPresent	Authorized Representative Information	
clickAndWait	btnContinue	
verifyTextPresent	First Name is Required.	
verifyTextPresent	Last Name is Required.	
verifyTextPresent	Address Line 1 is Required.	
verifyTextPresent	City is Required	
verifyTextPresent	Postal Code is Required	

Note: Only the result of the first attempt to save is shown.

We started by building small test modules and linked them together with Selenium Test Runner. Test Runner permits linking several test scripts to run sequentially. With the stack of modular mini-scripts, we were isolated from the problem and would have to open the IDE, open the script with the problem, position the application to the module's start point, and step through to find the problem. We opted to build long scripts (4,000 commands) that run for 1 to 2 hours. Running the scripts using the IDE allows us to diagnose a problem as it happens, fix the script in line, and continue. This interactive ability to debug and build scripts as you run offers a huge advantage in productivity.

Scripting During Development

We test and build scripts concurrently as the application progresses and continue to run them with each build. We review the use case and our Script Check List (see below) as we test to ensure we have tested everything for that page. As the application matures, the scripts mature and the result is a suite of regression scripts.

Scripting during development is helpful in other ways:

1. A repeatable condition can be set up rapidly for trial-and-error testing (e.g., find the best way to code a query).
2. Scripts can be run to generate an error condition and diagnostic tools can be run as the error is executed.
3. Scripts can enter field-identifying data. Field-identifying data refers to the technique of entering the field name and context into text fields, such as "Owner Four Addr 1" into the fourth owner's first address line. This can be followed through the application to ensure the original entry is saved and redisplayed and printed correctly throughout the application and its interfaces. This technique is also useful for checking interfaces, XML, database saves, translations, etc.
4. "Happy path" scripts with switches and stops can be used for debugging. Switches can be used to set different test environments and major paths through the code, and stops can be used to stop the script at any page in the application. The script can stop several pages into an application with a repeatable entry sequence to help diagnose a problem.
5. It permits easy generation of permutations or combinations for unit testing.
6. It provides a method to create repeatable test data quickly.
7. It ensures refactored code continues to provide previous functionality.
8. It identifies inconsistent element naming conventions.

9. It helps find the cause of intermittent problems three ways. First, it allows for better detection since the code is typically executed many more times so there is a higher probability of finding an intermittent defect. Second, the script can loop through the pages while diagnostics are running. Third, comments can be added to the script to document dates, times, and other pertinent information to help trace the intermittent defect.

To date, we have used Selenium successfully during the build process of three major and three smaller systems.

Script Check List

We incorporate the following tests in our regression scripts:

1. Test all paths. A list or diagram should be developed as part of test planning then the script(s) are checked to verify all paths are tested.
2. Verify static text and field labels are present.
3. Test dynamic pages (fields are displayed based on previous entry or action). Verify elements are visible or not visible, as appropriate, for each dynamic variation.
4. Test for mandatory fields. Verify error messages.
5. Test for maximum lengths. Verify error messages.
6. Test for trimming in text fields (i.e., removing spaces at the beginning and end of an entry).
7. Test for allowable character set in text fields and for exact number of characters per field specifications. For example, nine digits in a Social Security Number; no area code starting with 0 or 1; *etc*.
8. Test for inter-field edits:
    a. City, State, ZIP Code, Country validation
    b. No Address Line 2 without Address Line 1
    c. No Middle Name without a First Name
    d. No Phone Extension without a Phone Number
9. Test for duplicate entries (e.g., two owners with the same Social Security Number).
10. Test Business Rule Validations (e.g., if the ZIP Code is within the TriMet area, transit tax information must be entered). Normally, these are specified in the use case.
11. Test alternate methods of entry (e.g., phone numbers with or without spaces or dashes).
12. After all field validation tests are done, complete all fields on all pages with realistic data or field-identifying data.
13. Test dropdown lists: all selections are present, default is displayed (if any), selection can be made.
14. Test dynamic dropdown lists (those where names, addresses, or phone numbers that were previously entered are added to the dropdown list for future selection). Test adding, updating, and deleting. Verify this functionality follows business rules (e.g., an Incorporator Name is added since it may be selected again, but a Bank Name is not added to the dropdown since it's unique).
15. Test navigation and back button protection on each page.
16. Test side bar, header, and footer links.
17. Test all interfaces:
    a. To the Identify Management System (login and password).
    b. To expert systems for determining business activity type and business name availability.
    c. To the Credit Card Processing System.
    d. To all workflows based on status and user role.

e. To all back-end legacy systems via Web service calls.

18. Test transaction life cycle (i.e., test all states in the state transition diagram):
    a. User enters, stores, updates, deletes, and submits.
    b. Agency rejects. Verify all error messages can be selected.
    c. User can read all error messages selected. User can fix and resubmit.
    d. Agency updates and accepts.
    e. Back-end system is updated (full cycle).
19. Test validation at each step above and verify review and confirmation pages are correct.
20. Test user roles, verify authorized functions can be performed and unauthorized functions cannot be performed.

Maintenance

The following process helps us keep scripts up to date and minimizes the drudgery of maintenance. When a new build is about to be deployed, we review the release notes to determine the effect of the changes or new functionality on the scripts. We run the scripts to the point of change and then turn on Record Mode and start testing. Selenium records the tests, the script is updated, and the new version is saved. We run the complete script again to verify there have not been any other defects introduced and to verify the script runs with no errors or stops. We archive the previous version and document the changes made in the current version.

We use this process for deferred maintenance. If a defect is found and the team decides to defer the fix to a future release, we add a comment and a bypass in the script. When the defect has been fixed, the comment and bypass are removed and the original test is run. It does not have to be scripted again and the exact test that found the original defect verifies it has been fixed and does not reoccur. The test becomes permanent.

Evolution of the CBR Test Suite

When we started using Selenium in 2007, Central Business Registry (CBR) allowed our customers to file and pay for an Assumed Business Name (DBA) online. It has grown considerably, as the table below shows:

Measures of Oregon's Central Business Registry (CBR) System		
Measure	April 2007	April 2009
Number of Paper Forms Replaced	1	5
Number of State Agencies involved	1	3
Number of Web Pages	22	47
Number of Dynamic Pages	3	15
Web Services	3	8
XML Schemas	1	3
Transaction State Transitions	11	30

CBR's ultimate goal is to be a single site for a business to perform all government registrations, renewals, get licenses and pay fees. It now supports online filing of Assumed Business Names (ABN's), Domestic Business Corporations (DBC's), Limited Liability Companies (LLC's), and Domestic Nonprofit Corporations (DNC's) with the Oregon Secretary of State, along with a Combined Employer Registration (CER) that sets up an account at the Oregon Department of Revenue and the Oregon Employment Department.

These dates show the evolution of our test automation:

May 2007	First script, used primarily to support development.
August 2007	First full cycle regression script for ABN's.
November 2007	Added CER to ABN full cycle regression script.
February 2008	Full cycle regression script for LLC/CER.
May 2008	Full cycle regression script for DBC/CER. First with flow control.
October 2008	Duplicate Entry/Single Signature testing regression script.
January 2009	Full cycle regression script for twelve CER's.
February 2009	Full cycle regression script for DNC/CER.
March 2009	Four debugging scripts: ABN, LLC, DBC, DNP.
May 2009	Miscellaneous pages, paths, and states regression script.

From February 22, 2008, to May 20, 2009, there have been fourteen production builds. The average build cycle is 31 days. There have been 130 builds in our development environment; regression scripts were run for about a third of these. This is usually the environment where script maintenance and the development testing described above were performed.

There were 98 builds in our quality assurance (QA) environment and regression scripts were run for all of these. During this time 317 defects were discovered and fixed, and 153 enhancements were added to the system. There was one emergency build due to a defect reaching production. In the prior year there were three.

Limitations We Have Encountered

All testing tools have limitations. Our initial list was large due to our lack of experience. Here is our current list and what we have done as work-arounds.

Selenium IDE runs in Firefox, so running scripts in the IDE does not test the Microsoft Internet Explorer (IE) or Safari browsers. We perform our ad hoc user testing in IE and Safari to look for problems unique to these browsers. We plan to implement Selenium RC (Remote Control), which can run our scripts at the server, making it possible to test any browser.

Selenium IDE runs on the client. In order to run concurrent scripts, we use three PCs. Our start-to-end time for CBR, our most extensive test suite, is about 3 hours. After we implement Selenium RC we may consider the use of Selenium Grid, which allows multiple servers to execute scripts.

Extensions require Java Script. Java Script can be used directly in the script or to build actual stored commands. We use Java Script for:

- getting the date and time and appending to any field that must be unique;
- a stopwatch function;
- converting text to upper case;
- finding a link in a table where the link name varies;
- testing the browser back button; and
- testing trimming (removal of spaces).

We have not been able to test field-to-field navigation. When you execute

```
| type | firstName | John
```

Selenium looks for the element "firstName" and enters the text "John". Selenium does not know where the element is in relationship with other elements. We rely on our ad hoc testing for this. We will have the ability to address this when we implement Selenium RC.

There is no Find or Replace command in the IDE. We use comments to identify pages, sections, subroutines, etc., making it possible to find out where you are in the script. A text editor is used for large global changes.

Testing functionality outside the browser must be done manually. To test a printed report, enter an alert and pause. The alert provides instructions for the manual step that needs to be performed (e.g. click here and print report now). The pause gives the tester time to click Print in the Windows print menu and the script resumes at the expiration of the pause. We avoid this when possible to reduce manual monitoring

Observations

Time and Resources for Creating Scripts.

It is much faster to develop a test script in Selenium compared to our earlier hard-copy format. The time to develop a script in Word for an average use case was from 8 to 16 hours. This included reading the use case, running the applications, and writing out the instructions and expected results. This also included incorporating the use case into a script with other use cases and testing the resulting script by following each step and updating as necessary.

With Selenium, the time to script the same use case averages 2 hours, and we tend to be much more rigorous and add many more tests than we would in a Word script. Thus, we script four times faster with an increase in testing accuracy with Selenium.

This appears to be unrealistic until you look at our CBR application. In two years it has tripled in size and complexity, yet we are using the same resources for testing and have improved the quality of the releases.

Time and Resources to Run Scripts.

We can run a script in an hour that would have taken over 20 hours manually. This included an hour for a test manager to prepare and schedule the test, and an hour to review and document results. The remaining time was for testers to run the scripts. Running automated tests is 20 times faster than running the same tests manually and reduces manpower by a factor of 20.

With automated testing, the scripts can be started as soon as the new build is deployed. With manual testing, days could go by just getting the testers to start running scripts. After the script is run the log is examined for any errors. There is often a lengthy process of trying to re-create the error. Problem determination is much faster because the script can be stopped before the point of failure and stepped at that point. Automated testing reduces the testing cycle time from build to problem determination.

Testing Quality Increases

There is time for more planning; the tests are designed. The tendency is to be more thorough when creating an automated script. There is incentive to do it right since the test is going to be repeated hundreds of times. Besides, it's fun.

The script may have to be run several times if several builds are required before a release is ready for production. This is where manual regression starts to break down as mentioned above. With automated testing, a complete rerun of all scripts is not only feasible, it becomes routine.

Test Automation During Development

This has been the most valuable use of Selenium. It has allowed us to provide unit and functional testing concurrent with development. This interactive regression testing keeps the development from slipping backward. As the application matures, the script matures and the result is a regression script.

Script Maintenance

Script maintenance is costly and often is the reason manual scripts, as well as automated scripts, are abandoned. Selenium IDE records the maintenance testing so the script is automatically updated

Running scripts can also detect undocumented changes. If there is an undocumented change (not contained in the release notes), the script will often detect it and generate an error. Was this an omission in the release notes or an unauthorized change? Regardless, there was a documentation failure that must be addressed.

Testing Automation and Agile

Since the scripting process and testing is so much faster, we are far more agile. We are not at the stage of test-driven development, but our automated scripts flush out both development and requirement issues much earlier in the

process. Based on our CBR application, we are on a monthly Scrum cycle and have delivered significant new functions (153 enhancements over 14 months) with almost error-free code.

Return on Investment

Our return on investment is significant. Since Selenium is open source and the time to install is minimal, the only investment costs are (1) the time to learn the tool, (2) time to build scripts, and (3) extra PCs to run them. In our situation, our regressions scripts evolved from our agile development so there was not a separate scripting cost. That leaves training as the only significant investment. It took one tester a month to become productive and six to become proficient. The savings in manual script preparation and running has been offset over four times.

User Scripting

We have been successful in training several non-IT staff in scripting with Selenium. Our cashiering department manager scripted and tested a stand-alone credit card processing application, including all transactions, administrative pages, and role testing, in 2 days. Writing and running manual scripts would have taken 2 weeks.

Personnel Issues

Many of the personnel management issues are eliminated with automated testing. Selenium has no feelings that we have been able to detect. It can work long hours and does not interject opinions (we will talk more about opinions in ad hoc testing). It is as fast, accurate, and meticulous as the scripter permits. It does not whine when you need to run a script repeatedly. You don't have to bring it candy or buy it lunch.

Ad Hoc (Exploratory) Testing

Test automation has improved our ad hoc testing and vice versa. We do not need as many user testers, so we can use the most meticulous, pernicious, irreverent testers that have a knack of sniffing out bugs and breaking things. It has freed our testers from the script so they can truly run their own test scenarios and try the 'dumb things' they have observed. They can also test things "that will never happen" because they have the time and are not bound to testing the mainline functionality that is now in our automated scripts.

Our automated testing benefits from ad hoc testing through the use of test failure reviews. When ad hoc testers find a bug we conduct a review to determine how we missed it in our scripts. An ad hoc tester found a new validation that resulted in a trap. We discovered the validation was in the release notes, but the scripter missed it. Ad hoc testing caught it and our failure review pointed to the need for a double-check of the release notes.

Ad hoc testing benefits from the use of debugging scripts that can position testers at particular places in the application quickly. In CBR it can take 15 minutes to submit a registration before it appears in a reviewer's work queue. Once the registration is approved or rejected, it disappears from the queue. If the user wants to test that page, the script can generate any number of test registrations in a few minutes. The experienced tester can then focus on testing, rather than on data entry.

Ad hoc testing can also identify how we can improve the application. The drudgery and tedium of running manual scripts tends to stifle creativity. With manual scripts we would get little feedback from the testers other than the check marks in the Actual Results column of the script. Now our testers have time for suggestions to improve usability and reduce confusion.

Ad hoc testing is necessary for feedback crucial to improving quality and testing automation frees testers to provide that feedback.

Dual Monitors

This may sound like a minor point, but scripting and running tests with a single monitor is very difficult. You really need one monitor for the browser and one for the script.

No False Positives

In over two years of using Selenium, we have not seen any bug that was due to the tool.

Documentation, Help, and Training

The documentation for Selenium has improved considerably in the last two years. OpenQA is developing an excellent user manual. The OpenQA forum site is listed, as well. The Selenium site lists several firms that are addressing training and consulting. Formal training or a mentor would have helped our effort considerably.

Other Uses

The tool has many uses besides regression testing. Several have been discussed above and are recapped here along with a few others:

- Everything up? We have a script that can be run after an infrastructure change (e.g., a Solaris patch). The script opens all Web applications and performs a query in each.
- Demonstrations and Training. Anyone can demonstrate an application by just clicking pause/resume or running in step mode.
- We use "happy path" scripts with switches and stops for debugging.
- Help Desk staff are trained to use the "happy path" scripts so they can stop the application where the caller is having a problem. Then they can walk through the page, field-by-field and click-by-click. Prior to this we used hard copy screen shots for each page which did not show navigation or validation.
- Where's the Slow button? We have used Selenium to execute one or more transactions with a stopwatch function to tune database settings and look for code or interfaces that may be slowing response time. We have scripted different combinations of search criteria that can be run easily and the response time measured for each. Then we can modify the query to see the effect.

Future Directions

Some of the enhancements that we are planning for the next two years include:

1. Implement Selenium RC and possibly Grid. We will move our most stable, hardened scripts to this environment to reduce the time to run a suite of regression scripts. Since we can integrate Java with the script, we can test some things we cannot currently test. Some examples are navigation within a page, ability to drive the script with a list or file, and save and analyze log files.
2. Continuous integration of the build process, running regression scripts and build deployment, if successful.
3. Measure actual code coverage by our Selenium test scripts. We have no internal measurement of code coverage by Selenium. We would like to implement a tool that would show untested areas of code.
4. Implement automated testing for all Web-based applications to the same degree. We have more extensive scripts and testing in some applications, and would like to extend the benefits to all.

Conclusions

Test automation can be implemented in small shops and provide huge benefits. We are a twenty-plus person shop with a testing department of one. Selenium has provided a path to automation and we have reaped significant benefits.

Test automation can eliminate the Regression Test Dilemma. Good practice calls for a full regression after any change, but there is never enough time to run one. Quality suffers because there are parts of the system that were not tested and may have bugs. We currently run full regression scripts after every build. The process has uncovered bugs in areas we would not have suspected and typically not directly associated with the changes.

Automated testing can be used throughout the development cycle with significant benefit.

Automated testing does not replace manual testing. Automated testing makes manual testing more effective.

References

Home site for Selenium that includes documentation, download, support, forums, *etc.*: http://seleniumhq.org/

Home site for the Oregon Secretary of State: http://www.sos.state.or.us/