Automating Testing - Saving Time and Money

David Barrett (2013)
VLE Application Manager, E-Learning Development Team, University of York.
Email: david.barrett@york.ac.uk

1.0 Abstract

Automated testing frameworks, once the preserve of software developers, are now becoming more accessible and useful to the VLE application manager. Automated testing tools which record the clicks and playback tests are just a plugin away. These tools have the capability of allowing ‘front end’ testers to record once and repeat tests in a variety of browsers, ultimately speeding up testing and automating repetitive tasks.

This paper explores an experimental approach to upgrade testing (Blackboard Learn 9.1 SP12), and task automation (exporting data from Campus Pack 4), via web browser automation developed at the University of York using Selenium (http://www.seleniumhq.org/). A formal set of pre-upgrade tests, conducted by hand in a variety of supported browsers are used at York to validate service pack upgrades. Some of these tests have been automated and can now be run on demand to support change management processes. This has freed up staff time to appraise new features and tools rather than getting bogged down in testing.

Automated testing has also been deployed to support the transition away from a cloud based blog/wiki service. Development of data driven automated scripts to mass export content has saved time and money and provided a neat solution to reclaiming data hosted off campus.

A review of how Selenium has been utilised will be followed by an evaluation of the pros and cons of automated testing. Looking forward, the potential to reuse automated tests internally for regression testing, service validation and even performance load testing will be examined. We will also consider how practical it is to exchange tests securely between institutions.

2.0 Introduction

Software testing is expensive in terms of time and effort, yet is an essential part of IT service delivery. Time spent checking ‘out of the box’ software works when operating with local data and resolving problems can lengthen the software deployment lifecycle. These factors contribute to large upgrade projects becoming the norm as making changes to production systems are avoided due to rollout overheads, such as pre-deployment testing. Automating functional software testing has the potential to reduce the amount of time spent on testing and open up opportunities to reuse test scripts in other areas of service management, such as service availability monitoring, load testing and automating bulk operations. This paper looks at...
an experimental approach to performing functional tests and automating bulk operations with
two examples from the E-Learning Development Team, University of York.

3.0 The Problem

The initial problem, and the reason automating functional testing was considered by the E-
Learning Development Team, was to reduce the amount of time spent on pre-deployment
testing of Blackboard Learn. Pre-deployment testing had proved essential, and the discovery of
several ‘show-stopping’ problems in previous upgrades reinforced this. Whilst Blackboard as a
software company was inevitably conducting their own functional tests of Learn, the E-Learning
Development Team often found that problems arose when changes to the VLE were made. This
is because software development teams at Blackboard do not have access to local data when
developing and testing their product. Problems arising from zero day bugs to issues with local
data causing problems, ultimately manifest as errors to users and degrade their experience of
the VLE.

The key business driver in having a test strategy is to reduce the risk of making changes to the
VLE that introduce software problems by proactively seeking and resolving problems during
testing. What automated testing offers on top of that is the recording of repeatable and
consistent tests across a range of user supported technology. Once the initial task of creating
an automated test was completed, the automated test should be reusable, repeatable and
independent of other tests, to be run as part of a test suite or individually. Having tests
conducted by a machine with little human intervention is appealing for a number of reasons.
After the initial investment in time is made creating the automated scripts, staff time is freed up
to work on other things. Automated tests are consistent, so if logically constructed, they will
always test the same thing. Human testers can be inconsistent. Testing a user interface (UI) can
be mundane and different testers interpretation of the same test (for example uploading a file to
the Content Collection) can differ as there is often more than one way to achieve the same goal.
This inconsistency can lead to tests being passed that should not have passed, and ultimately
problems going unnoticed.

The second problem considered was how, if possible, can web browser automation be
employed to automate repetitive tasks. Whilst some batch operations can be performed via
back end processes the E-Learning Development Team were faced with a specific problem of
extracting a large amount of data from a cloud based content solution, Learning Object’s
Campus Pack suite. Exporting cloud based blog or wiki data was possible via the Campus Pack
UI, but exporting data for the entire instance via the UI manually was not practical as it would
take too long. Web browser automation, already being developed for automated testing, was
employed for this task.
4.0 The Solution

A review of automated testing technology revealed that a number of proprietary software products have automated testing components, for example Ruby’s Watir, or Test Unit or Microsoft NUnit. A lot of these technologies however, are based on ThoughtWorks Selenium Core. Selenium is a testing framework that can automate web browsers. It runs by taking control of the browser and using JavaScript to manipulate page elements. Selenium can be used to test websites on all the major web browsers in use today (Mozilla Firefox, Google Chrome, Microsoft Internet Explorer and Apple’s Safari). Selenium is also fairly platform agnostic and can be used on Windows and Mac based computers. Selenium was therefore identified as the right automated testing framework to use in the project.

One challenge in using Selenium was to try and keep the creation of automated tests simple enough so that a detailed level of programming was not required. Staff creating the test scripts do have a good working knowledge of HTML, however were not familiar with the creation of scripts in languages that used the Selenium framework. The solution to this problem was the Selenium IDE, a Mozilla Firefox plugin created by Samit Badle. The Selenium IDE offers recording and playback capabilities via the Firefox browser, but also allows test scripts to be organised into structured test suites. In this way it was utilised in this project as the automated test script integrated development environment (IDE).

![Figure 1: The Firefox Selenium IDE with a test suite loaded.](image)
4.1 Organising test scripts

Before any VLE tests were automated testing was performed manually to a formal test plan. The test plan was organised by user role (student, instructor, system administrator) and tests appropriate to these roles were documented and outcomes recorded in a Google spreadsheet. This pre-existing test plan was used as a basis for the creation of a lightweight Configuration Management Database (CMDB) which underpinned the new automated test system. Tests were already categorised by role and functional area, for example:

<table>
<thead>
<tr>
<th>Test Script ID</th>
<th>Category</th>
<th>Tool</th>
<th>Test</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1.1</td>
<td>Collaboration</td>
<td>Blog</td>
<td>Create Blog</td>
<td>Instructor</td>
</tr>
<tr>
<td>A1.2</td>
<td>Collaboration</td>
<td>Blog</td>
<td>Create Post</td>
<td>Instructor</td>
</tr>
<tr>
<td>A1.3</td>
<td>Collaboration</td>
<td>Blog</td>
<td>Create Post</td>
<td>Student</td>
</tr>
<tr>
<td>A1.4</td>
<td>Collaboration</td>
<td>Blog</td>
<td>Comment</td>
<td>Student</td>
</tr>
</tbody>
</table>

These test scripts were uploaded to a database so they could be used more easily in the test automation process. Running test scripts via the Selenium IDE and checking after the completion of a test run for any failure, whilst very useful, does not offer the complete test management solution. In conjunction with automating tests, a new test management solution was developed in house. Its purpose was to catalogue test scripts in a CMDB and act as a repository of test runs, cases and results. The system was developed using rapid application development tools supported within the University of York, namely Oracle’s MySQL (as the database engine) and Adobe’s Coldfusion (as the web front end). The solution was structured as follows:

![Figure 2: The Test Manager system](image)
The Test Manager system adds a level of control to using test scripts that run in the Selenium IDE. Tests results were permanently recorded and progress on a specific test run can be tracked.

4.2 Creating test scripts

Test scripts were created line by line in the Selenium IDE. As the quantity of test scripts grew the speed at which new ones could be created increased. This was because lots of tests performed similar operations. For example, interaction with the left hand menu or the content editor in Blackboard Learn, whilst reasonably complicated to script initially only needs to be scripted once. These scripts can then be reused within other scripts. When this occurred the source script was defined as a ‘block’. Defining these blocks within other test scripts allows the blocks to be easily identifiable and updated if required en masse within a test suite. An example of blocks being used within a test script is as follows:

<table>
<thead>
<tr>
<th>TEST SCRIPT (A1.3)</th>
<th>ADD BLOG POST (ROLE STUDENT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK 1</td>
<td>Login as test student</td>
</tr>
<tr>
<td>BLOCK 2</td>
<td>Navigate to the test course site</td>
</tr>
<tr>
<td>BLOCK 3</td>
<td>Navigate the left hand menu to access the blog tool</td>
</tr>
<tr>
<td>NEW SCRIPT</td>
<td>Enter the blog and click the Add New Entry button</td>
</tr>
<tr>
<td>BLOCK 4</td>
<td>Add content to the content editor</td>
</tr>
<tr>
<td>BLOCK 5</td>
<td>Click the submit button</td>
</tr>
<tr>
<td>BLOCK 6</td>
<td>Take a screenshot for human inspection/verification</td>
</tr>
<tr>
<td>NEW SCRIPT</td>
<td>Verify text on the page is present</td>
</tr>
<tr>
<td>BLOCK 7</td>
<td>Logout of the VLE</td>
</tr>
</tbody>
</table>

In this example, “add a blog post as a student”, almost all the script is made up of reusable code blocks which control the left hand menu, content editor, take a screenshot and log in and out of the VLE.

Guidance on creating a good test script was issued to testers. Research in effective use of test execution tools (Fewster & Graham 1999), provides a useful insight into how automated tests should operate, which is still relevant to today’s tester. They state when tests are constructed there should be careful consideration to what the test is doing. Does it mirror typical human interaction with the tool? How will the test script be able to verify if the test was successful or not? Are the tests independent enough from one another, so that if one does fail, the whole
suite doesn’t stop working? To make the tests as credible as possible, human interaction was taken into account with the tester asked to map out the steps involved in the test before scripting anything.

### 4.3 Test verification

In order to ensure that verification of tests was accurate a two-step approach was used with some automated verification taking place and some human verification. Automated verification looked for text or an html element to be present on completion of the test. Whilst this was effective to rule out complete failure of a test it did not take into account any style related problems. For example, following a test which involved posting to a blog, the automated script checks for the presence of ‘test A1.3 this text was inserted into content editor’ on the page. Following that initial automated verification, the test script then takes a screenshot of the page, which can be reviewed by a human later on. Following a test run, tests reported a status back to the Test Manager system via a web service. Tests returned one of the following statuses: passed, failed, passed pending review. Test returning passed pending review were tests that took a screenshot that required human inspection. Screenshot images were displayed via a simple UI in the Test Manager system with reference to what the test was, making it simple for a human to quickly go through the screenshot images and mark the test as passed or failed.

### 5.0 Evaluation

The first problem posed in this project, to reduce the amount of time spent performing functional tests has been realised, however, it should be noted that a significant amount of time was spent developing the test scripts in the first place and creating the Test Manager system to control test runs. In addition, some time will need to be spent on up keeping the existing suite of test scripts. As Blackboard Learn evolves and the UI changes some scripts will stop working and require updating. Overall the investment has been worthwhile and has freed up testers time from having to perform functional testing, to looking at the application of the software, a far more useful pursuit.

In addition to running tests to support a specific change to the software, test runs can be performed to support regression testing. Web browsers are constantly updated and whilst Blackboard seek to keep their software functional on the latest version of supported browsers, this is not always possible. Updates to web browsers pushed to users from the browser manufacturers inevitably cause problems. Regression testing can help identify those problems, and whilst the problem will already have made it to production systems, the E-Learning Development Team will better placed to investigate workarounds and advise users.

In the future test scripts could also be repurposed for inclusion of load and performance testing and in service availability monitoring. Having a set of tests that reflect the type of actions users perform could greatly improve the ability of a load testing to truly analyse a system. In addition
availability monitoring cannot always gauge the true health of an IT system. Often only cursory checks for database connectivity or application node health are made, where in reality a problem with a specific tool could be missed. Reusing automated tests could help build up a more complete system health check and contribute to improved service availability metrics, beyond system uptime/downtime statistics.

One unexpected benefit of automated testing was the use of the technology to perform bulk operations via the VLE UI. The challenge was to export data from the Campus Pack hosted service as part of an institutional exit strategy from the service. The problem was how to click the export button in over 10,000 separate content items within the cloud based service. The solution was to use web browser automation. In this case Ruby’s Watir testing framework was utilised. Watir is based on the Selenium Core and as this was a once-only operation having a scripted solution was the simplest option. As links to the content items within Campus Pack could be guessed a downloading service was developed that visited each export page, initialised an export and then collected the resulting zip file. Being able to automate this process enabled the University to recover data from the cloud based system and organise the export files in a logical way.

### 6.0 Conclusions

Automated testing has surely saved time and money and will allow the E-Learning Development Team to realise other benefits moving forward, however using test automation does bring its own set of challenges.

The advantages of automated testing include: time saved performing functional testing allowing learning technology staff to evaluate the software under investigation, instead of just testing it. Automated testing can be used to support service transition as well as regression testing. In the future tests could be reused to perform more realistic load testing and service availability monitoring.

The challenges of an automated testing model include maintenance of tests scripts. As UIs change, new scripts require development and existing scripts updating. Unless test scripts are going to be reused and maintained then the initial investment in time required writing the scripts may prove too onerous. In addition, some scripts still require human intervention to verify the test was successful, for example checking a screen shot output.

It would be wrong to solely rely on automated scripts as a definitive testing approach, however I feel they can provide a solid basis for core testing, and if maintained, a good set of realistic tests for other areas of application management.
Reference List: