SimShip software testing using Shadow™

K Arthur, D Hannan, M Ward
Brandt
6 Faughart Terrace,
St. Mary's Road,
Dundalk, Co. Louth.
karthur@brandttechnologies.com, mward@brandttechnologies.com, dhannan@brandttechnologies.com
www.brandttechnologies.com

Abstract
We believe that our approach to automated software testing is novel. We can test several language instances of a product simultaneously, either through direct engineer interaction or by a record/playback script. The Shadow™ application can manage a situation where the user interface of the product under test is slightly different in layout, either due to localisation of the different versions, or due to the original language version running on different platforms. In our pilot studies, we examine the effect of separating out the functions of a test engineer into a product specialist and QA specialist. Our testing methodology outputs a set of screenshots of the products under test in each language. The screenshots can be used by a translator for linguistic/consistency QA or in product documentation. We performed a comparative analysis of the automation tool Winrunner with the Shadow™ testing process.

Keywords: *Automated testing, Quality Assurance testing, QA., Localisation, Localisation Testing*

1. Introduction

The purpose of this paper is to describe a new automated testing tool called "Shadow™" developed by Brandt and to illustrate how it works with the aid of case studies. Shadow™ is a software application that allows the user to control one or more operating systems (PCs, VMWare instances) simultaneously. The idea of one computer controlling another is not new and there are many products available for this purpose such as VNC (RealVNC Ltd. 2002). However, the idea of one computer controlling many computers simultaneously appears to be novel.

SimShip is the process of shipping the localised product to customers at the same time as the original language product, or within a couple of weeks. This can result in increased market share (Common Sense Advisory 2004) and possibly increased revenues. However, SimShip can also result in unrecoverable costs due to factors such as those described by Langewis (2003): localised software not taking off in foreign markets, localised software being substandard, or localised software having been created inefficiently.

In this article we propose to examine software testing, explain what Shadow™ is and how it is relevant to testing and, finally, discuss some case studies. In the first section we will introduce Shadow™ and describe in general terms what the product does. We will give an overview of the Shadow™ functionality and describe how it might be used to test original products and localised products.

2. Software testing and software quality

In this section we will examine software testing and software quality in general terms. Towards the end of this section we will introduce Shadow™.

2.1 What is quality?

In this section we are interested in the definitions of quality as applied to software development. The Crosby (1980) definition of quality defines quality as "conformance to requirements". An alternative definition states that quality is "fitness for use" (Juran 1951). The value of these context free definitions is that they can be used in domains other than software (Mass 2004). For our purposes, software quality will be defined as software that conforms to customer driven requirements and design specifications. Categories under which quality can be assessed include (King 1996): functionality, reliability, usability, efficiency, maintainability, and portability.

Software testing should be viewed as a scientific process, wherein an application is placed under known conditions of setup, hardware and configuration, where it accepts some known input and where it should result in an expected outcome. We expect that the quality of the test process and effort will have an impact on the quality of the software. Unexpected performance, errors or bugs are introduced into software in a large variety of ways during the development and localisation process. Software "testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong..." (Hower 1996). Bugs are typically of the following types:

- Logical errors - where the software runs, but produces unexpected results.
- Crash bugs - where the software fails in a catastrophic manner.
- Failed functionality - where the application fails to meet its specifications.
- Layout issues - widgets/text not displayed correctly.
- Linguistic issues - spelling, grammar.
- Localisation issues - date/time format, number format.

2.2 Software testing

Software testing is performed to find defects, to ensure that the code matches the specification and to estimate the reliability of the code. The output of testing is a defect list and the effectiveness of the testing process can be measured through the number of defects (Voas 2004). As the number of defects is reduced, there is an enhanced confidence in the quality of the product. We believe testing should be viewed as an integral part of the software development process, rather than an activity performed after the core development. Internationalisation testing targets unexpected performance in a software product when used with different character encoding schemes; it also ensures that a product is localisable. Internationalisation testing should be performed early in a product's lifecycle to identify and remedy any issues.

Testing adds to the cost of production. When balanced against the idea that the most expensive defect to fix is the one found by the customer, it is highly desirable to capture defects as early as possible in the software development lifecycle. In 2002, a US government agency suggested that software bugs cost $59.5 billion annually (Tassey 2002).

There are two approaches to software testing, namely manual testing and automated testing. Manual testing uses human engineers to aid in this process. Automated testing requires that the test script is coded, and then executed using some application. Successful software development and localisation should use both manual and automated testing methodologies. The balance between each can be decided using budgetary and schedule constraints. Shadow™ can be used to complement manual testing or it can be used as a test automation solution.

2.3 What is Shadow™?

Shadow™ is a software-testing tool for performing automated and manual tests on original or localised software products. Shadow™ allows the user to record and play back scripts that run and control multiple machines at the same time. It also allows the user to directly interact with multiple machines simultaneously, making the manual test effort more efficient. Shadow™ allows the user to simultaneously test localised software applications running:

- Different language operating systems, or
- Original language products running in different configurations.

It is in this context that Shadow ™ is ideal for use in a project where the localised and original language products must ship together, that is "SimShip".

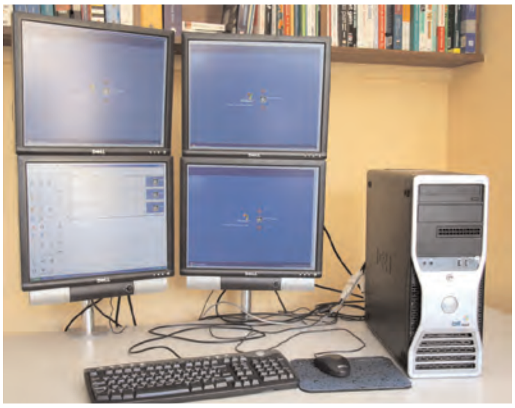

FIGURE 1: SHADOW™ SETUP

Our development philosophy has been that we avoid operating system dependencies, where possible. We want to avoid getting information about the application under test that is not available on all operating systems in the same manner. In fact, Shadow™ concentrates on getting information about the application under test that is available to the human user.

Shadow™ uses the keyboard, mouse and timing information from the user, and employs intelligent technology to identify the location of the interactions in the screen. During the replay of scripts Shadow™ finds the appropriate location in the screens under test. This allows for widgets to have changed position and size between tests. We are currently adding character recognition to the Shadow™ suite of tools. This has particular relevance to localisation, where testing is not just a matter of finding the appropriate text, but where a mapping exists from one language to another. Testing of the localised products can take place at the same time as original language product testing. The key to SimShip is having the localisation and development cycles in parallel, now with Shadow™ the test cycles of original and localised products can also take place in parallel.

The specific problems we address with Shadow™ are:

- Making automated software testing easier to use with less programming.
- Separating the roles of Test Engineer into the complementary roles of "Product Specialist" and "Quality Assurance Specialist". See the glossary of terms for proposed specifications of the two roles.
- Making software testing more like the actions of a human user.
- Making automated testing easier - reducing the barrier to automation.
  - o Reduction of the cumbersome nature of script creation (Dustin 1999).
  - o Reduction of the training time necessary for the tool to be useful.
- Increasing compatibility with 3rd party application components (widgets), see (Dustin 1999).
- Accelerating the manual testing process through Shadow™'s unique user interface.
- Recording screenshot data by default. In other automated testing applications this has to be implemented programmatically.
- Reducing script maintenance. This is accomplished by examining the developer's release documentation to identify those user interface areas that have changed in the new build. We can then compare screenshots from the newly run tests with previously recorded screenshots in those areas that have been changed.

3. Shadow™

Automated testing tools currently come in three general categories with different operating modes, namely (Skrivanek 2005):
- Simple capture - record and playback.
- Object Oriented Automation - API calls "under the hood".
- Image-based component discovery.

Simple capture utilities: Using a "simple" capture utility an engineer can create a test script containing an exact sequence of keystrokes, mouse movements and/or mouse commands. However, with the slightest change to the GUI of the software under test the recorded script becomes invalid requiring the engineer to possibly re-record the script.

Object Oriented Automation: Makes API calls to the operating system to identify information about the control being interacted with. Once the automation tool has the handle for the object, it can then manipulate the control. It is the reaction of the code to a function that makes an API call that is being tested, not the actual user interface. This is an important distinction.

Image-based component discovery: In this mode of automated testing, images are taken of regions around the mouse at the time of some mouse interaction. This image is then stored. At a later time the image is used to identify where the mouse action should be taken either during playback of a script or where the action should be taken on some other system. This is the method used by Brandt in the Shadow™ application. The characteristics of image based component discovery are similar to those of Object Oriented automation, without the need for any of the "under the hood" information.

3.1 Shadow™ setup

Shadow™ is a piece of software that can control several machines simultaneously. It can do so in several different ways, such as:

- Shadow™ can make a group of machines perform exactly the same actions at the same time.
- Shadow™ can make a group of machines perform "nearly" the same action at the same time.
- Shadow™ can record and playback scripts ("Mimic" and "Exact Match" modes).
  - o "Mimic" mode is where Shadow ™runs as a "simple capture utility".
  - o "Exact match" mode is where Shadow runs

as an "image based component discovery" application.

4. Case studies

In this section we will look at case studies involving the use of Shadow™. We will examine three case studies where Shadow™ was used, and the purpose for which it was used.

Client	Profile
A	Multinational software publisher
B	Supplier of technical authoring, documentation and localisation services
C	Brandt

TABLE 1: LIST OF CASE STUDIES

4.1 Client A

Client A produces enterprise resource planning (ERP) software for managing and analysing corporate spend. Their clients include "Fortune 100" multinationals. Brandt provides translation and engineering services to this client.

4.1.1 Task specification

Client A requires that the translated and localised user interface of its products undergo linguistic testing and functional testing. For this test case, the client used a combination of methods to perform the linguistic and functional testing, that is using Shadow™ and WinRunner. The engineer performed the following tasks:

- Wrote test scripts.
- Updated test scripts.
- Set up the hardware and software.
- Executed the test script on the machines, using both Shadow™ and WinRunner.
- LQA performed by linguists using the screenshots.
- Localisation functional QA using Shadow™ and WinRunner.

4.1.2 Shadow™ usage

In this section we examine how the output of Shadow™ is used in linguistic testing and how Shadow™ itself is used in functional testing.

Linguistic testing

Software is translated in a tool that does not show the translator the context of the strings. No matter how familiar the translator is with the product, and how much experience they have with previous versions, there is no substitute for having the translator seeing the translated strings appearing in a running build in their proper context. The translator can then make the appropriate changes, if necessary, to the software strings. There are at least two methodologies that can be used in this situation such as having the translator go through the running build with a test script to bring up every screen, or providing the translator with the screens in the form of screenshots (or MHT files in this case study). We use the second method in this case study. Some advantages of giving the translator only screens to review include that the translators do not have to have spend time negotiating their way through the product to find the areas in which an update occurs and there are cost savings in setup time. This is especially useful for small and frequent updates to a product.

Functional testing

The engineer used Shadow™ with three machines for functional testing, each machine running a different language operating system. The output of this was a list of bugs documented by the engineer. In this process the engineer manually went through the test script on one machine, with the others automatically following in Shadow™, noting bugs as they progressed. As with the linguistic testing, the process was then repeated with a different language set.

4.1.3 Results

Table 2 and Table 3 below show the time spent using both Shadow™ and WinRunner to perform the same tasks for a QA cycle. The engineer has detailed the time taken for the average amount of days spent on each task. The tables show the complete list of tasks.

The engineer notes that WinRunner can fail if the Internet connection is slow and fails to bring up a widget in a "reasonable" time. A key element of Shadow™ functionality is "Wait for feature". This means that Shadow™ waits a configurable amount of time for a feature to appear on the screen before proceeding or declaring a failure.

4.1.4 Conclusions

From the tables above we can see that Shadow™ and WinRunner take approximately the same time to setup and run a test cycle in which there are a small number of screenshots required. Where a larger number of screenshots are required, the time taken to run Shadow™ is less than the time taken to write the code and execute it in WinRunner. The engineer notes in his report that WinRunner requires the build of software under test to be specially prepared in

40 screenshots	***Shadow™***	***WinRunner***	***Total***	
Task	*Days*	*Days*	*Days*	*Comment*
Write LQA script			3 - 4	Tool independent
Update LQA script			1 - 2	Tool independent
Write TSL script		1 - 2		WinRunner only
Execution using tool for screen-shots	2	1		Both Shadow™ and WinRunner
LQA by translators			1 - 2	Tool independent
Functional QA			1 - 2	Tool independent
Total days	8 - 12	8 - 13		

TABLE 2: RESULTS OF USING SHADOW™ VS WINRUNNER FOR 40 SCREENSHOTS

400 screenshots	***Shadow™***	***WinRunner***	***Total***	
Task	*Days*	*Days*	*Days*	*Comment*
Write LQA script			20 - 25	
Update LQA script			10 - 15	
Write TSL script		25 - 30		WinRunner only
Execution using tool for screen-shots	16	8		
LQA by translators			5 - 6	
Functional QA			9 - 10	
Total days	60 - 72	77 - 94		

TABLE 3: RESULTS OF USING SHADOW™ VS WINRUNNER FOR 400 SCREENSHOTS

order to function with it, whereas Shadow™ does not. Each resource on the page has to have an AWL name (AWL is a client proprietary web language). The AWL name identifies the widget without reference to the text on it, so that the same button in French and German will have the same AWL name, but different text. This is an "under the hood" requirement of WinRunner that Shadow™ does not have. In summary, Shadow™ was used as a QA tool in this study. Shadow™ was more efficient than WinRunner on the QA of a larger product. Shadow™ did not require special preparation of the product build before its use and as a result it could be used "out of the box".

4.2 Client B

Client B is a provider of documentation, consultancy and recruitment solutions.

4.2.1 Task specification

Client B wanted Brandt to prepare a document containing screenshots of their software that could be sent to translators for review. There were 203 screenshots to be taken of the software opened in a software engineering tool. This tool is a visual localisation environment allowing engineers to view the localisable resources of software. The final deliverable to Client B was one PDF per language, each containing the 203 screenshots of the English and localised software side by side.

4.2.2 Shadow™ usage

Shadow™ was installed on 5 VMWare machines each running Windows XP Professional. The appropriate tools were also installed. Each localised software file was opened in the visual editor on one virtual machine, and then the English, giving a total of 5 connected clients to Shadow™. Usually, Shadow™ takes a screenshot of the whole screen, or if configured to do so, just the window in focus. In this project, the window in focus was the visual editor application and the required dialog was part of a screenshot that would need to be cropped. The Shadow™ code was modified, using an extension of the existing "exact match" technology so that a screenshot could be adaptively cropped to leave only a feature of interest. This project is unique among the test cases, as it showed how the Brandt software development team was integral to completing the project using Shadow™ by making appropriate modifications to the application.

4.2.3 Results

The engineer took about 1.5 hours in total to setup and screen shoot 4 languages. The integration of the screenshots into the final document MS Word document and production of the PDF are tasks independent of Shadow™ and would be the same length irrespective of how the screenshots were taken.

4.2.4 Conclusions

Shadow™ was used as a QA tool for this project, where the output was a set of screenshots for linguistic QA performed by the translation team. This project involved use of a software application that requires significant processing resources of the host operating system. We found that the process using Shadow™ was faster than the manual process, directly as a result of the ability to perform tasks in parallel.

4.3 Brandt

Brandt uses Shadow™ for the purposes of testing and as an automation tool to perform tasks that need to be repeated frequently. Brandt has found that there are tasks in the production of multimedia tours in Adobe Captivate® that are repetitious and prone to human error, such as audio integration, text integration, and font assignment. We have written short scripts, called macros, which can be activated using different keystrokes that perform these tasks.

4.3.1 Task specification

Adobe Captivate® is a tool for rapid authoring of multimedia tutorials. These tutorials contain screenshots, text, animations and audio that all have to be localised. The localised elements have to be integrated into each slide in the tour. Tours can vary in size from 30 slides to in excess of 100 slides, and are localised in a number of European and Asian languages. Brandt has run this project several times over the past eighteen months and improved the process to make it more efficient, with the result that the output is more consistent and of a higher quality.

Audio integration

This involves importing a single WAV file per slide. The WAV file name is numbered in sequence, for example "0001.wav". To perform this task manually, the engineer runs the risk of importing the wrong file into a slide. For an engineer who is not familiar with the language they will not be able to test that the audio is appropriate to the slide. This has to be repeated up to 100 times without error, for up to eight languages. That is a total of 800 cut-and-pastes.

Text Integration

There is localised text on every slide in the tour. On most slides, the text is unique. The engineer has a MS Word document with the text ordered sequentially for every slide. Once again, this is a straightforward process, but it is prone to error. This has to be repeated up to 100 times without error, for up to eight languages. That is a total of 800 cut-and-pastes. One possible error is that the text sequence goes out of line with the slide sequence. Once this happens, the process will have to be repeated from the place where the error occurred.

Font assignment

The localised text appearing on every slide has to be a certain font for European languages, the SimSun font for Simplified Chinese and MS Mincho for Japanese. Each slide has to have the font individually set. The engineer must go to every slide, bring up the text properties and set the correct font. Again, this has to be repeated for each slide and then for each language.

4.3.2 Shadow™ usage

Shadow™ is used in the following way. Three virtual operating systems are set up in VMWare, in our case MS Windows XP Professional. Shadow™ is set up to view the 3 operating systems, generally referred to as (virtual) machines. Each machine runs Adobe Captivate®, MS Word and possibly MS Notepad. Each virtual machine is set up with the correct script (French, German or Japanese as required). The macro is set to run the correct number of times, usually equal to the number of slides, and it is then started. Each run of Shadow™ can process one tour in three languages in parallel. While this is running, the engineer can usually go and perform another task.

4.3.3 Results

Shadow™ was vital for this project, as some of these tasks are repetitive and subject to human error. Table 4 shows the times taken for Shadow™ to run the individual tasks versus the manual time for the same task. Note that Shadow™ can run at least three tours in parallel so there is a further efficiency present.

Task	Automation per 30 slide tour - minutes	Manual time per 30 slide tour - minutes
Audio integration	10	25
Text integration	15	25
Font assignment	10	20

TABLE 4: RESULTS FOR SHADOW™ AUTOMATION VS MANUAL IMPLEMENTATION

Whereas one Shadow™ run will perform audio integration (or another task) for three tours in parallel, one engineer performs the manual equivalent process in a serial fashion.

4.3.4 Conclusions

Shadow™ was used as an automation tool for this project and it was essential to the effectiveness of the engineering team. For some tasks, the time it takes to perform the task manually is not much different from the time the automation takes. In this case the automation is more efficient when it is possible to perform the tasks in parallel. It might not be worth investing time in automating some once off tasks. For this project, Shadow™ was invaluable, because of the number of tasks that are repeated.

5 Conclusions

In this paper we have examined some of the advantages and disadvantages of the different modes of testing software applications. We have come to the conclusion that a mix of manual and automated testing is essential to the success of a project. One type of testing will not replace the other. The ratio of the mix between automated and manual testing can be dictated by the nature of the project, the budget and the schedule. Shadow™ can help make automated testing more efficient by separating the QA from specialist product knowledge and hardware setup. As part of the localisation process, Shadow™ can be used to take screenshots of the running software that can be given to linguists for review. Shadow™ can also be used by the engineer, with specialist product knowledge, to walk through the different language versions of a product at the same time.

References

RealVNC Ltd. (2002) RealVNC remote control software [online], available at: http://www.realvnc.com/ [accessed 28 August 2007].

Common Sense Advisory (2004) Common Sense Advisory explains how companies will become "world enterprises" [online], available at: http://commonsenseadvisory.com/news/pr_view.php?pre_id=8 [accessed 05 September 2007].

Langewis, C. (2003) Localization and ROI: Increasing Value by Eliminating Pink Ink [online], available at: http://www.ableinnovations.com/pdf/pinkink-I.pdf [accessed 05 September 2007].

King, M. (1996) The ISO 9126 Standard [online], available at: http://www.issco.unige.ch/ewg95/node1.html [accessed 30 August 2007].

Hower, R. (1996) Software QA and Testing Resource Center [online], available at: Http://www.softwareqatest.com/qatfaq1.html#FAQ1_1 [accessed 28 August 2007].

Voas, J. (2004) A Few Assertions on Information Hiding [online], available at: http://www.cigital.com/papers/download/quality-time1.pdf [accessed 28 August 2007].

Juran, J. (1951) "Quality control handbook", New York: McGraw Hill.

Maas, K.F. (2004) Introduction to Quality [online], available at: http://www.kfmaas.de/qintroed.html [accessed 28 August 2007].

Dustin, E. 1999, Lessons in Test Automation [online], available at: http://www.stickyminds.com/sitewide.asp?ObjectId=1802&ObjectType=ART&Function=edetail [accessed 06 September 2007].

Skrivanek, J. (2005) Testing GUI Applications [online], available at: http://wiki.java.net/bin/view/Javapedia/TestingGUIApplications [accessed 06 September 2007].