POINTE TECHNOLOGY GROUP, INC.
8201 Corporate Drive, Suite 700 - Landover, MD 20785 - (800) 730-6171/(301) 306-4400 - Fax (301) 306-4421

Designing an Automated Web Test Environment

Presented by:

Dion Johnson
Senior Test Consultant

May 17, 2001

Designing an Automated Web Test Environment

Dion Johnson
Pointe Technology Group, Inc.
8201 Corporate Drive
Landover, MD. 20785
djohnson@pointetech.com

Abstract

This paper offers an alternative to the typical automated test scripting method of 'record and playback now and enhance the automation environment later'. It explores a regression automation system design for testing Internet applications through the GUI, along with scripting techniques to enhance the scalability and flexibility of an automated test suite. This paper will present a basic structure for an automated test environment, and will expand on each of the items found in that structure. Web testing levels will be laid out, along with a basic approach to designing test scripts based on those web-testing levels.

Introduction

It is not very difficult these days to sell those who are involved in the IT industry on the importance of regression automation – at least the theoretical importance. It seems as though most understand and are able to spew out, with robotic precision, that:

1. Automation saves time
2. Automation can be run over and over again (repeatable)
3. Automation runs tests the same way each time, maintaining reliability
4. Automation is our friend…

The automation-inhibiting problem, however, is the belief that automation should require no additional thought other than that which requires one to executing existing manual test cases with an automation tool.

In order to facilitate the desire to crank out quick and thoughtless automated tests, the industry is creating more and more "user friendly" GUI automation testing, many of which are geared to web-based applications. These tools offer record and playback features that generate code or code-less visual scripts that may test one's application by simply navigating through it. These applications, with their increased capabilities, can be extremely helpful, but also add to the problem. One of the major selling points for some of these applications is that, "With our application, you save a lot of time because you don't have to spend time thinking about, and planning out how you want to perform your tests. You just record your steps, and our tool will put in the checks for you." What they fail to realize is that eventually, some thought will have to take place, be it proactive or reactive, and reactive thought is almost always more time consuming.

It is important to avoid falling into the trap of believing that some magical tool is going to eliminate the need for thinking and planning. This paper helps dispel that myth, along with providing some specific information for designing an automation environment. The design is specifically geared towards web test automation.

Common Approach

With many, getting an automation tool is like a kid getting a new toy – they jump right in and start playing. And the resulting test suite, much like a child's new toy, just doesn't last.

A common approach for test automation may include the following:

- Automate test only during "spare time"
- Jump right in and start recording scripts
- Add application checks with no particular method
- Add some parameterization later
- As time passes and the number of scripts increases, replace some of the redundant code with functions

This approach presents several problems. Spare time automation keeps the entire effort from getting the focus that it needs, and it is often at the forefront of all other automation problems and stumbling blocks. Stumbling blocks such as hard to maintain scripts. When scripts are simply recorded with no forethought or design, what typically happens is that they won't play back exactly as recorded, prompting the need for some patchwork solution that gets the script to behave as desired. Patchwork solutions don't consistently work, or stand-up over time, making the script very hard to maintain.

Redundancy also results from this common approach to test automation. When the same actions appear in several different places or scripts, and that action changes in the application under test (AUT), the change will need to be reflected in all of the scripts that use that action. Excessive rework and ***reactive*** function creation are the results.

The patchwork solutions and rework, along with any unforeseen issues that may occur during a test run, will affect the performance of the script.

Another problem that occurs with this jump-in-head-first-approach is that it yields a one-dimensional suite of tests. In other words, the scripts are only readily available to be run at one level, and in one location.

Improved Approach

Test automation involves developing test scripts, so the software tester is in effect creating a software application. The test automation application should, of course, be on a much smaller scale, than the larger application that it tests, because spending too much time on automation, or trying to automate too much can be counter-productive. At the same time, however, automation should be treated more like software development projects. See Pettichord's "Seven Steps to Test Automation Success" for more information on this concept.

Part of treating test automation more like a software development project, is the design phase. Manual test cases are often independent of each other; so creating automated test cases solely from manual test cases will result in a group of independent test scripts that all have to be maintained separately and differently. The improved approach to test automation that this paper addresses, deals with creating an automation test environment, instead of just creating scripts.

Prior to designing the automation environment the following considerations are made:

- Script maintainability and flexibility
- Script performance
- The different levels of testing
- The number of applications under test
- The number of environments testing takes place in
- The different browsers used to test the web applications

After the appropriate considerations are made, the following the steps can be taken to design the automated web test environment:

- Create the appropriate directory/test suite structure
- Develop initialization and configuration parameters
- Create functions
- Design test scripts based on pre-defined testing levels

Considerations

Script maintainability and flexibility

The maintainability and flexibility of automated tests pose the following question – if the application or data in the application were to change, how much work would it take to update the automated test that accesses the changed portion of the application? The automation environment and scripts need to be designed in a way that minimizes the amount of work involved in maintenance.

Script performance

Performance is a major concern for software applications, particularly internet applications. By the same token, performance of automated tests should also be of concern. As mentioned earlier in this paper, one of the advantages of automating tests is that the execution is faster than manual execution. If the performance of automated tests is ignored or taken for granted, the time saved by running automated tests can be drastically reduced.

Different levels of testing

In a perfect world, there would be an unlimited amount of time for testing…. Correction. In a perfect world, there would be ***no need for testing***, because the applications would be developed with no bugs. Since we live in neither perfect world, it is necessary to develop automated tests that are able to conform to the varying schedule that testers are given from release to release, and even from *build* to *build*. Not all builds contain the same types of things. Depending upon the nature of a build, the time given to test it may vary from a few minutes to a few days (maybe more). Full regression may not always be an option. Only certain types of testing actions might be allowed, and these different types of actions comprise different levels of testing.

The number of applications under test
A software project that requires testing may consist of more than one AUT. When responsible for automating tests for all of these applications, it is important to take into consideration the things that each of these applications have in common and how they are related, and design the automated test environment accordingly.

The number of environments testing takes place in
A software project may have several different environments that testing occurs in. Most have at least the following three:

1. Functional test environment
2. Development environment
3. Production environment

Each environment may be on a different machine, connect to a different database, and limit the types of testing that can be executed. The user shouldn't have to go to several different places to set up several different parameters every time the automation test suite is ready to be run. The automation environment should account for the variations.

The different browsers used to test the web applications
Theoretically, If an application is required to be compatible with several different browsers, it should be the same on all of those browsers it is compatible with. In actuality, there may be several subtle differences among browsers that the automated test environment should be equipped to deal with.

Design Steps

Create the appropriate directory/test suite structure
As mentioned before, test automation is a small-scale form of software development that involves creating a software package. It is important for this package to have a pre-planned directory structure, for several reasons. First, it is important because one obviously needs to know where things are going to go. Second it is important because this package may need to be moved as a whole. The environment may need to be copied and installed onto another machine. Creating a directory structure makes the environment portable.

Develop initialization and configuration parameters/scripts
These are parameters/scripts that set up the environment before actually running the scripts that test the AUT. These parameters basically control the other components of the automated web test environment.

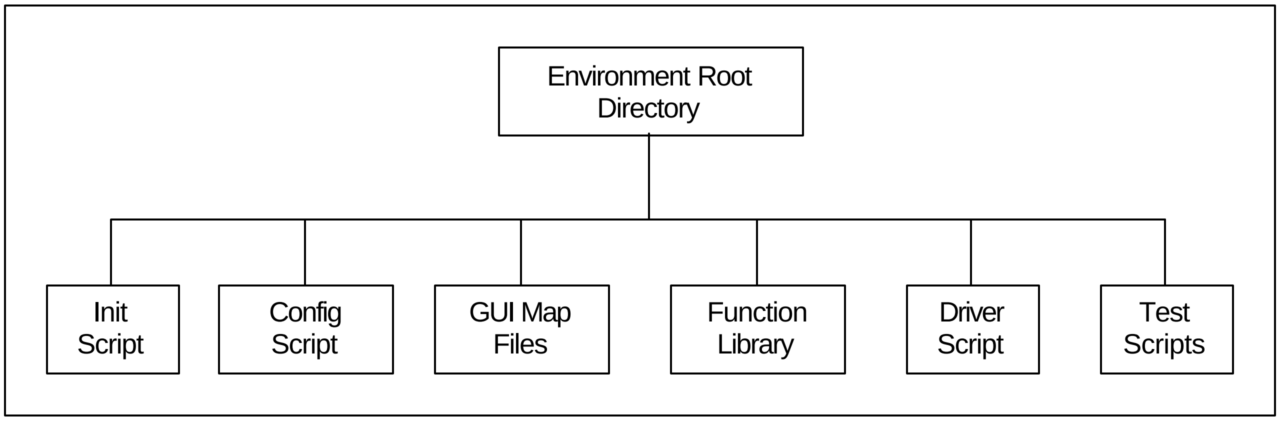
Create functions
Function creation does not have to begin after the test script creation. There are basic actions that take place during testing, that one knows ahead of time will be executed

over and over again. So why wait until the test script creation begins? Create functions for those highly repetitive actions at the beginning.

The function samples in this paper are geared towards automation tools with a scripting language, but keep in mind that functions should be created when using automation tools that use visual script, also. Many of these applications provide a way to reuse blocks of script.

Design test scripts based on pre-defined testing levels
Designing testing levels was discussed under the **Considerations** sub-heading. Specific web test levels will be discussed later on in the paper.

Directory Structure



There are several components of the automated test environment. Above is a chart displaying a pictorial representation of how the components fit into the environment. Below is a list containing component definitions. It may also be necessary to create a sub directory for each component shown.

Component Definitions

Init Script
The Init Script (initialization script) brings the environment to a controlled stable point. If the environment is initialized, there is a better chance of the test scripts running properly. Some specific initialization parameters will be discussed later.

Config Script
The Config Script (configuration script) sets certain parameters needed for the test run. Some specific configuration parameters will be discussed later.

GUI Map Files
Whether or not this component exists, depends on what tool is used. The name "GUI Map" comes from Mercury Interactive's WinRunner tool. Other tools may have

something similar to a GUI Map, but it may have a different name. The GUI map serves as an interpreter that enables the automation tool (i.e. - WinRunner) to communicate with the AUT.

Function Lib
The Function Lib (function library) is a set of functions used in the test suite. It is normally loaded at the beginning, and functions are called from it throughout the test run. Some specific types of functions will be discussed later.

Driver Script
This is the script that coordinates test activities by calling other scripts. It controls everything that takes place during the test run. The driver is what allows the execution of several scripts, without supervision. It may run initialization scripts, set up configuration parameters, load GUI maps, and run test scripts.

Test Scripts
These are the components that actually perform tests, and verify that the AUT works as desired. The two actions listed in Example 1 is a representation of a test script. Test scripts perform actions, and generate results files.

Initialization Scripts

The main difference between items that are placed in the initialization scripts and items place in the configuration scripts is that items in the initialization script are not likely to change from test run to test run, or from machine to machine. Settings in the initialization script are going to remain fairly constant throughout the project life cycle, but may change occasionally. There are two basic types of initialization scripts.

- Automation Tool Initialization Scripts
- AUT Initialization Scripts

Automation Tool Initialization Scripts

The Automation Tool Initialization Scripts initialize the automation tool, by doing several things. These scripts may load compiled modules, such as function libraries, so that they may be used throughout the time the automation tool is in use. They may customize GUI Maps, and may include global wait/think times, and directory path variables.

AUT Initialization Scripts

The AUT Initialization Scripts initialize the application under test. There are some very specific tasks that need to be executed when dealing with a web AUT, however, that are probably not a concern for other client/server applications. These tasks include:

- Clearing the cache
- Setting cookies
- Setting frame format

Clearing the cache

Miller talks about the importance of clearing the cache in his paper entitled, "WebSite Testing". Under the heading, WEBSITE TEST AUTOMATION REQUIREMENTS, he asserts that since caching can improve the apparent performance of the web site, it should be cleared in order to get an accurate picture of the site response time. This is a good point, but it may be argued, by some, that the response time is of little concern when doing functional testing; the response times, it may be asserted, will be measured during the load/stress testing. There is an additional reason that the cache should be cleared, however. When dealing with Internet applications, it is common place for an action to change the make-up of the application. The cache saves this new make-up, which may cause a problem for automated test scripts, because the test scripts may be looking for the default make-up during script execution.

Look at the following line of HTML code for example:

```
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#000080" VLINK="#FFFF00"
ALINK="#C0C0C0" >
```

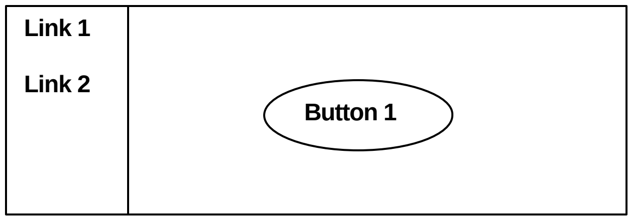
This line of code, in reference to the link, is saying that the color of the link prior to being clicked is navy; while it is being clicked, it is yellow; after it is clicked, the color is silver.

The silver color of the clicked link is cached, so that even after leaving that page and bringing it back up, the link will be silver instead of navy. This could pose a problem for automation scripts when performing link validation. If it is looking for a link to be navy, but it is silver instead, an unnecessary failure will be generated in the results. Unnecessary failures compromise the integrity of the results.

Setting Cookies

Cookies are short pieces of data used by web servers to help identify web users, and they have the ability to change certain site behavior. When this is the case, it is important to create a set of initialization steps that ensure the application behaves as expected.

Consider the following scenario. There exists a web page with two frames as seen below.



The left frame has two links, while the right frame has a button. Clicking on Link 1 will bring up a new page called Page 1, while clicking on Link 2 will bring up a new page called Page 2. Clicking on Button 1 brings up a new page, but the page that comes up all depends on which of the two was clicked last. Each time one of the two links is clicked a cookie is set on the server with information that specifies which link was clicked. Based on the setting of this cookie, Button 1 may bring up Page 1 or Page 2.

In order to have a controlled environment, it is important to know what functionality is to be expected out of Button 1. An initialization routine in this case, may simply involve automating the process of clicking on Link 1 at the beginning of a test run.

Setting Frame Format
With Internet applications, it is not unlikely for pages to change their formats (table arrangements, object names) based on certain actions, page handlers, etc. It therefore becomes important to create an initialization routine that stabilizes the frame format.

Configuration Files/Scripts

Under the **Improved Approach** heading and **Design Steps** sub-heading, it mentions initialization and configuration jointly. That's because there is a fine line between initialization elements and configuration elements. As alluded to in the previous section, the main difference between the two is that the configuration elements are those elements that will need to be changed more frequently.

The **Considerations** sub-heading under the **Improved Approach** heading lists several items that need to be considered before designing the automation environment, and these items were discussed in detail. The Configuration file/script is what manages how these considerations are put into action in the automation test suite.

More information on how configuration components are used in the scripts will be given under the **Test Scripts** heading.

Function Library

The next step in designing an automated web test environment is the creation of functions. There are some basic types of functions that will, without a doubt, be used throughout the automation effort, and should be created prior to the creation of automated tests. These function types are as follows:

- Login functions
- Navigation functions
- Error handling functions
- Loading functions
- Miscellaneous verification functions
    - Window check
    - Date/time check
    - Account number/login ID check

Login Functions

These are the functions that login and logout of the AUT, and other supporting applications. The login and logout processes are going to executed several times during a test run, and are composed of very basic steps, so these functions should not be difficult to create.

The login functions should be tied to several configuration parameters, such as the user ID, and password configuration parameters.

Navigation Functions

Most applications have several main areas that are navigated to many times during testing. For example, an application may have a Main page, a FAQ (Frequently Asked Questions) page, an Account Services page, and several Orders pages.

Navigation for these different areas may be as follows:

Page	Navigation
Main	Main
FAQ	Main → Account Services → FAQ
Account Services	Main → Account Services
Orders 1	Main → Orders → Orders 1
Orders 2	Main → Orders → Orders 2

These paths are fairly basic, and can be put into functions at the very beginning of the automation process.

Error Handling Functions

Error handling functions are functions created to deal with certain unexpected events that may occur during testing. If nothing is set up to handle these events, they could kill automation run completely. Internet applications have a fairly standard set of events that occur regardless of the application. The list of events includes popup security windows, information windows, and error windows, along with the page cannot be found frame. Good portions of the events that will have to be handle require nothing more than clicking an OK button to close the popup window. Many of these events can be turned on and off in the browser, but some can't be. For those that can't be turned off, routines should be created to handle them.

Loading Functions

Loading functions do exactly as the name implies – load things. These functions load files and compiled modules for use by the automation tool. For example, in WinRunner, GUI Map files need to be loaded. When several GUI Map files exist, it becomes much simpler to put all of the load statements into a single routine, and execute them by running that one routine.

Miscellaneous Verification Functions

Although most automation tools have built in checks that can be used to verify certain application attributes, there are times when user defined checks are much more efficient. Some user-defined checks that one may want to create include the following:

- Window check

- ❖ Date/time check
- ❖ Account number/login ID check

Window check
The Window check function's purpose is to verify that a specified window or frame appears. The biggest reward that can come from using the Window check function comes in the area of script performance. Most automation steps and verifications occur inside of window or frame. If the window or frame does not appear, the tool wastes time trying to execute steps in a window that does not exist. The Window check function creates the ability to first check for the existence of a window, then make the execution of the following steps contingent upon whether or not that window exists.

Date/time check
This is a handy function to have when the date and time need to be verified in the application. Since the date and time are dynamic fields, this function simply takes the system clock information, puts each element (day, hour, minute, etc.) into an array, and compares these elements to the date/time elements in the application.

Account number/login ID check
Web sites often have the account number or login ID listed on every page. Since the exact same thing is being checked on every page, and the results for each page should be exactly the same, it only makes since to have a function that does this check. This function will use the login configuration parameter in its verification.

The login ID will normally follow some text that reads, “Login:” or “Account Number:” so creating this function will simply involve performing some kind of text check on the data that follows the “Login:” or “Account Number:”

Test Scripts

The final step in the design phase is designing the test scripts. The way an individual test script is designed actually depends on the user's AUT, but what this section does is reveal how to incorporate web test levels into that design. The importance of this concept is presented in the Improved Approach section.

In order to understand how to design a test script based on web test levels, it is necessary to first break down the concept of web testing. The following list is comprised of steps that separate web testing into components, and displays how to put those components together to form basic web test levels.

- ❖ Examine the microscopic view of web testing
- ❖ Examine the macroscopic view of web testing
- ❖ Develop web test levels

Examine the microscopic view of web testing

Looking at web testing through a microscope yields that just about every type of web testing is composed of the following checks:

- *Server response time verification* – testing to make sure the web site's response to certain actions doesn't exceed a reasonable amount of time
- *Content Verification* – verifying the layout and size of frames and tables; verifying text and returned data; verifying URLs
- *Cosmetic Verification* – verifying actual fonts, colors, and object positioning appears correctly.

Examine the macroscopic view of web testing

From the macroscopic view, there are three testing categories:

1. Link testing
2. Application testing
3. HTML testing

Link testing

Link testing involves both checking and testing links. The difference between checking a link and testing a link is fairly simple. Checking a link involves checking the link's properties to make sure the link is pointing to the correct URL. Testing a link involves actually clicking on that link to ensure that the resulting page is correct, and has the proper attributes.

Application testing

Application testing verifies that the AUT has the correct functionality. For example, if the application places orders, a test would verify that the order was placed properly. Parameterization is normally a big part of application testing, because several sets of data are often used to test the same piece of functionality.

HTML testing

HTML testing verifies that the HTML code follows proper conventions, and doesn't have errors.

Develop web test levels

When developing web test levels, it is important to evaluate the needs of the AUT's project environment. This section presents some basic levels based on an evaluation of web testing in general, but one should feel free to expand upon these levels in one's own environment.

As discussed before, different levels of testing are developed to give the ability to focus automation executions on varying aspect of the testing cycle. This is important; especially when time is limited. Often times when performing manual testing, a project lead may tell the test lead that the test team has two days to focus on a full build, or a half day to focus on a build that incorporates only application changes. It is possible that the project lead may go and tell the test lead that the test team has thirty minutes to test a build that just focuses on link changes, after which time the build is going to be pushed into production as an emergency patch.

Just as one should be equipped to honor the request when executing tests manually, so should one be equipped honor the request when executing tests through an automated process. The testing levels listed in this section give that ability.

From the macroscopic view of web testing, it can be deduced that there are two basic types of test scripts:

1. Link test scripts
2. Application test scripts

HTML testing will not be address in this paper, because a developer normally does it.

Then combining the elements from the microscopic view of web testing with the elements from the macroscopic view of web testing the following web test levels can be derived:

- ❖ Application Tests
- ❖ Link Tests
- ❖ Link Checks
- ❖ Parameterization On/Off
- ❖ Cosmetic Checks On/Off
- ❖ Content Checks On/Off
- ❖ Combination of all of the above

The application test level and the link test levels can be controlled by simply picking which scripts to run. The other test levels can be incorporated into the test script and controlled by mapping each to a configuration parameter that accepts the values “On” and “Off”.

Bibliography

Bret Pettiford, “Seven Steps to Test Automation Success”, StarWest 1999. http://www.io.com/~wazmo/papers/seven_steps.html

Bret Pettiford, “Success with Test Automation”, Quality Week, 1996. http://www.io.com/~wazmo/succpap.htm

Dion Johnson

Dion Johnson is currently a Senior Test Consultant for Pointe Technology Group, Inc. Mr. Johnson is responsible for providing IT consulting services in a manner that demonstrates expertise in the system development life cycle, particularly in the areas of Quality Assurance (QA) and Quality Control (QC). He must handle all aspects of the delivery of onsite customer services on behalf of Pointe, and manage engagements within the stated objectives of the statement of work.

Mr. Johnson joined Pointe Technology nearly two years ago following nearly three years at Bell Atlantic, where he spent much of that time leading major test efforts, including an automation testing initiative.

Mr. Johnson holds a Bachelor of Science degree in Electrical Engineering from Morgan State University, where he graduated with honors.