Automation Framework of Browser Based Testing Tool Watir

Er. Harwant Singh Gill  
Astt. Prof Deptt. Of CSE/IT  
GIMET Amritsar

Er. Manjit Kaur  
Astt. Prof Deptt. Of CSE/IT  
GIMET Amritsar

Mr. Amit Puri  
Astt. Prof Deptt. Of MCA  
GIMET Amritsar

Er. Anil Jaswal  
Astt. Prof Deptt. Of CSE/IT  
GIMET Amritsar

Abstract— This paper presents the automation framework of Browser based testing tool Watir (web application testing in `Ruby). The main advantage of a framework is the low cost for maintenance. If there is change to any test case then only the test case file needs to be updated and the Driver Script and Start-up script will remain the same. So, there is no need to update the scripts in case of changes to the application. Choosing the right framework helps in maintaining lower costs. There are several frameworks available suited for the particular application. Specifically, this paper discusses Keyword-driven framework for Watir and its various aspects in detail that can guide web application testing actively.

INTRODUCTION

With the development of internet technology, the Web application becomes more and more complex and the scale of it changes more and more great. The Web application program testing is much more difficult than the traditional. Meanwhile, software development cycle becomes shorter and shorter; this makes web application testing, especially functional regression testing, more challengeable. Traditional ways of testing can no longer meet the need of the requirements of software development, thus automation testing becomes the only way out. As a frequently used method, the automation scripts created by Capture-Replay technology are too difficult to maintain. Thus developing effective web testing tool and framework becomes a hotspot in software testing research area [1].

BACKGROUND

A test automation framework is a set of assumptions, concepts and tools that provide support for automated software testing. Choosing the right framework/scripting technique helps in maintaining lower costs. The approach of scripting used during test automation has effect on costs. Automation Framework is not a tool to perform some specific task, but is an infrastructure that provides a complete solution where different tools work together in a unified manner [2, 3].

Fig. 1. Testing Framework for Web Applications
The test automation tool vendors market their product as the main feature of the tool is the ability to capture the user actions and later to playback them. Here is the basic paradigm for GUI-based automated regression testing – the so called Record/Playback method (also called as Capture/Replay approach). The basic drawback in this method is the scripts resulting from this method contain hard-coded values which must change if anything at all changes in AUT. The costs associated with maintaining such scripts are astronomical, and unacceptable. These scripts are not reliable, even if the application has not changed, and often fail on replay (pop-up windows, messages, and other things can happen that did not happen when the test was recorded). If the tester makes an error entering data, etc., the test must be re-recorded even if the application changes, the test must be re-recorded. All that is being tested are things that already work. Areas that have errors are encountered in the recording process. These bugs are reported, but a script cannot be recorded until the software is corrected. So logically nothing is tested by this approach. This method is fraught with problems, and is the most costly (time consuming) of all methods over the long term. The record/playback feature of the test tool is useful for determining how the tool is trying to process or interact with the application under test, and can give some ideas about how to develop test scripts, but beyond that, its usefulness ends quickly.

A. Types of Test Automation Frameworks

With the elimination of Record/Playback method, the shift is towards more powerful automation methodologies. The Automation testing framework is responsible for defining the format in which to express expectations, creating a mechanism to hook into or drive the application under test, executing the tests and reporting results. There are several test automation frameworks available, among these the selection is made based on the factors such as reusability of both the scripts and the test assets. The different test automation frameworks available are as follows [4].

- Test Script Modularity
- Test Library Architecture
- Data-Driven Testing
- Keyword-Driven or Table-Driven Testing
- Hybrid Test Automation

KEYWORD OR TABLE DRIVEN TEST AUTOMATION

Keyword-driven testing is growing in popularity. It involves the creation of modular, reusable test components that are built by test architects and then assembled into test scripts by test designers. Keyword-driven test looks very similar to manual test cases. In a keyword-driven test, the functionality of the system-under-test is documented in a table as well as in step by- step instructions for each test [5]. It requires the development of data tables and keywords, independent of the test automation tool used to execute them and the test script code that "drives" the application-under-test and the data. In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test. In this method, the entire process is data-driven, including functionality. It is typically an application-independent automation framework designed to process the tests. These tests are developed as data tables using a keyword vocabulary that is independent of the test automation tool used to execute them.

![Fig.2 Keyword Driven Testing Framework](image-url)
A. Action, Input Data, and Expected Result ALL in One Record:

The data table records contain the keywords that describe the actions we want to perform. They also provide any additional data needed as input to the application, and where appropriate, the benchmark information we use to verify the state of our components and the application in general.

For example, to verify the value of a user ID textbox on a login page, we might have a data table record as seen in Table 1:

<table>
<thead>
<tr>
<th>WINDOW</th>
<th>COMPONENT</th>
<th>ACTION</th>
<th>EXPECTEDVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginPage</td>
<td>UserIDTextbox</td>
<td>VerifyValue</td>
<td>&quot;MyUserID&quot;</td>
</tr>
</tbody>
</table>

Table 1

Another advantage of the keyword driven approach is that testers can develop tests without a functioning application as long as preliminary requirements or designs can be determined. All the tester needs is a fairly reliable definition of what the interface and functional flow is expected to be like.

Take login example. We do not need the application to construct any login tests. All we have to know is that we will have a login form of some kind that will accept a user ID, a password, and contain a button or two to submit or cancel the request. A quick discussion with development can confirm or modify determinations. We can then complete the test table and move on to another.

The essential guiding principles to be followed when developing overall test strategy are:

- The test design and the test framework are totally separate entities.
- The test framework should be application-independent.
- The test framework must be easy to expand, maintain, and perpetuate.
- The test strategy/design vocabulary should be framework independent.
- The test strategy/design should isolate testers from the complexities of the test framework. [6]

B. Example

In order to open a window, the following table is devised, and it can be used for any other application, just it requires changing the window name.

<table>
<thead>
<tr>
<th>Window Name</th>
<th>Control</th>
<th>Action</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window Name</td>
<td>Menu</td>
<td>Click</td>
<td>File, Open</td>
</tr>
<tr>
<td>Window Name</td>
<td>Menu</td>
<td>Click</td>
<td>Close</td>
</tr>
<tr>
<td>Window Name</td>
<td>Pushbutton</td>
<td>Click</td>
<td>Folder Name</td>
</tr>
<tr>
<td>Window Name</td>
<td>Verify</td>
<td></td>
<td>Results</td>
</tr>
</tbody>
</table>

Once creating the test tables, a driver script or a set of scripts is written that reads in each step, executes the step based on the keyword contained the Action field, performs error checking, and logs any relevant information. Keyword-driven testing is a tool used in automated testing where pre-defined keywords are used to define actions. These actions correspond to a process related to the application.

It is the first step for creating a domain testing language. These keywords (or action words) represent a very simple specification language that non-programmers can use to develop automated tests. If these keywords are extended to emulate the domain language, customers and non technical testers can specify tests that map to the workflow more easily [7].
C. FRAMEWORK OVERVIEW

Definition 1: A keyword identifies an operation or atomic action in test execution. In order to enable the automation test drivers to recognize the keyword, there are some restrictions on naming keyword. For example, we define “Click”, “SetValue” and “VerifyValue” in GUI application as keywords. Similarly, “Connect” and “Cmd” belong to the keyword sets of database driver.

Definition 2: A command is used to simulate an action of software. In general, a test case can be parsed into sequences of test steps. To model the steps in test cases, we constitute a series of standards in our framework to formalize the steps of test cases into framework-recognizable input. We call them commands. Each step can be parsed into a command, and the command is based on keyword-driven.

A command can be defined as: \( (TS, DriverName, Keyword, Arguments, ExpectedResults, And VariableNames) \), where:
- \( TS \in \mathbb{N}^+ \): The number of test steps in test case;
- \( Keyword \) is a string which belongs to the pre-defined keyword sets;
- \( DriverName \) is a string to represent the test driver for executing the command;
- \( Arguments \) is a string list to define the properties and input of the keyword;
- \( ExpectedResults \) is a string to specify the expected results. It cannot be NULL if the purpose of the command is to verify a value;
- \( VariableNames \) is a string to specify the name of Variables. It cannot be NULL if we need to fetch a variable value during the test execution.

Definition 3: A Command Sequence contains a series of commands to be executed in a specific test driver in sequence. It can be defined as: \( (DriverName, CommandList) \), where:
- \( DriverName \) is a string to represent the test driver for executing the command;
- \( CommandList \) is a list of commands for execution in the test driven.

In this framework, the test cases are described in XML, which include the test steps, predefined keywords and expected results, etc. In automation engine layer, it will parse the test cases into the command sequence according to the driver type, and dispatch the command sequence to different test drivers in the test driver layer. The test drivers will include UNIX Driver, Database Driver, GUI Driver and FTW Driver, etc. to dispatch the commands to the corresponding test applications to generate the test scripts automatically according to the keywords predefined in the commands. During the execution of the test scripts by the test applications under different test environments, the automation engine layer will coordinate the interfaces and results among the test drivers to ensure executing the process automatically [5].

D. Merits of keyword driven testing

The merits of the Key Word Driven Testing are as follows,
- The Detail Test Plan can be written in Spreadsheet format containing all input and verification data.
- If "utility" scripts can be created by someone proficient in the automated tool’s Scripting language prior to the Detail Test Plan being written, then the tester can use the Automated Test Tool immediately via the "spreadsheet-input" method, without needing to learn the Scripting language.
- The tester needs only to learn the "Key Words" required, and the specific format to use within the Test Plan. This allows the tester to be productive with the test tool very quickly, and allows more extensive training in the test tool to be scheduled at a more convenient time.

E. Demerits of keyword driven testing

The demerits of the Key Word Driven Testing are as follows,
- Development of "customized" (Application-Specific) Functions and Utilities requires proficiency in the tool’s Scripting language.
- If application requires more than a few “customized” Utilities, this will require the tester to learn a number of "Key Words" and special formats. This can be time-consuming, and may have an initial impact on Test Plan Development. Once the testers get used to this, however, the time required to produce a test case is greatly improved.
CONCLUSION

The main concern of this paper is to describe the comprehensive framework for automating the testing process for Browser based testing Tool Watir. The framework provides a means for experimenting with different strategies for each testing component, within an integrated and automated framework. It includes how to execute the keyword based test case by using different test applications under different test environments to meet the specific requirements. We propose a novel and adaptive framework, which includes three layers: automation engine layer, test driver layer and test execution layer. Through the interactions among these three layers, the keyword-based commands in the test case can be converted into the test scripts to be executed under different test environments automatically. Additionally, it is important to note that this does not suggest that these tests can be executed automatically as soon as the application becomes available. The test record in a Table may be perfectly understood and executable by a person, but the automation framework knows nothing about the objects in the particular record until we can provide that additional information. That is a separate piece of the framework that can be discussed through application mapping as a future work.

REFERENCES


