# Comparing GUI Automation Testing Tools for Dynamic Web Applications

Samer Al-Zain[1], Derar Eleyan[2] and Yousef Hassouneh[3]

[1] Lecturer, Birzeit University
(Birzeit, Palestine)
*szain@birzeit.edu*

[2] Assistant Professor, Birzeit University
(Birzeit, Palestine)
*deleyan@birzeit.edu*

[3]Assistant Professor, Birzeit University
(Birzeit, Palestine)
*yhassouneh@birzeit.edu*

**ABSTRACT—** ***Automating software testing helps agile teams to release production-ready software as often as needed. There are so many automation testing tools with record/playback features available for test teams to automate functional tests against web applications. However, choosing the right tool can be vital to project success. This paper presents a survey on automation testing tools for web applications. These tools have been chosen from both open-source, namely Selenium and Sahi, and from commercial tools such as TestComplete and Microsoft Visual Studio 2010. These tools were compared based on the effectiveness of recorder/playback tools, handling of page waits, cross browser compatibility, technical support, and the number of different techniques available to programmatically locate elements on web pages. Results show that free and simple tools can be more powerful and time saving, compared to commercially sophisticated and expensive tools. This paper also summarizes best practices and guidelines when adopting automated GUI functional tests against web applications, drawing a road-map for test engineers to follow.***

**Keywords—** Web application, GUI, Software testing, Agile testing.

## 1. INTRODUCTION

Web applications are the most popular among software classes. This is partly due to the ubiquity and convenient use of web browsers as thin clients, as well as, the ability for web applications to support cross-platform compatibility. Another reason for its popularity is the ease of updating and maintenance: no need to distribute and install new versions on client machines, updates are only installed on servers.

This popularity gave web applications an enormous boost, allowing them to support various activities in business, social, medical, educational and government functions [1]. Social networks at the Internet have turned the world into a small village. Web portals on the other hand operate as an Internet gateway, offering diverse services such as search engines, news, weather, stock prices, entertainment and email. However, recent studies and reports show that web applications are not as functionally dependable as they should be. These studies showed that 29 of 40 leading e-commerce sites as well as 28 of 41 government sites showed evidence of failure in functional requirements when used by web users [1][2][3].

Testing software through the GUI is extremely important, and should be given the right amount of time and effort during software development. Brooks et al [4] identified that thousands of real faults have been detected by GUI testing and proved that a large portion of those faults belong to the underlying business logic of the application rather than the GUI itself [5]. Additionally, [6] argues that Software testing should constitute 30 – 40% of total project time and effort in order to deliver software with acceptable quality.

Software automated testing is an integral and important part of software development methods. Among those methods, agile development and testing methods are expanding and are becoming more popular recently. Many modern software development teams apply agile software development methods such as SCRUM, XP (Extreme

Programming), Crystal, FDD or DSDM to name a few. In these methods, teams need to release production-ready software every sprint, which lasts between 1 to 4 weeks.

In order for test engineers to keep pace with development, they need to employ automation testing. In fact, automation is a critical component to maintain agility, and save testers from routine, time-consuming and error prone manual testing activities [7]. However, there are so many testing tools with record/play back feature available in the market in both open-source and commercial sectors for web applications. Test teams should think carefully when choosing one tool over the other. Which tool to select can be challenging and vital to project success, especially when testing against dynamic web applications.

Another compelling issue for test engineers is to know what to automate exactly? And, how do test engineers know if their automation tests are enough, precise and effective? For example, automating functional tests against the graphical user interface (GUI) can be tricky due to the fact that GUI changes more frequently by development team. Moreover, automation test scripts can get larger every sprint, thus, making them harder to maintain, enhance and modify by test engineers. Test engineers need to have a clear plan and guidelines of what should, and what should not be automated. In this paper, guidelines and best practices are formalized, enabling test teams to apply successful test automation strategies.

In this study, automation tools for web applications, taken from both commercial and open source fields, are compared together. These tools are chosen based on their popularity and reviews from software developers and testers. Using an experiment research method, these tools were run against real-world, dynamic web application sample, built for the purpose of experiment itself and written in ASP.NET 3.5 using Visual Studio 2010. All web pages are built in run time in order to test the effectiveness of automation tools.

Results show that open source and free automation tools seemed to produce more reliable and flexible test cases than commercial automation tools.

This paper is organized as follows: section 2 discusses dynamic web applications; section 3 provides an introduction to HTML, since that HTML is what the automation tools try to access and manipulate during test play back; section 4 discusses test automation in relation to categories and importance; GUI testing tools for web applications are outlined in section 5; section 6 presents experiment design and methodology; whilst results are compared in section 7; guidelines for implementing GUI automated tests for web applications are put forward in section 8 and conclusions are drawn in section 9.

## 2. DYNAMIC WEB APPLICATIONS

Dynamic web pages form the basis of modern web applications, which is the main theme of this paper. Contents and layout of a dynamic web page are changed and updated every time that page is requested by user. Some dynamic web pages even change their layout and content during the process of viewing that page. Such pages employ a popular technology such as Ajax (Asynchronous Java and XML) that do not require the whole page to be reloaded.

Web applications follow the n-tiered structure by nature, and every tier has a designated role. Even though there are other variations, the most common form of n-tier structure is the three tiers [8].The first tier, known as the client (presentation), is represented by a web browser such as Mozilla or Internet Explorer. This thin-client web browser is mainly responsible for viewing and rendering HTML documents on screen and generating HTTP (Hyper Text Transfer Protocol) requests to a web server. The second tier is the middle tier which is the web application itself, running a dynamic web content technology such as ASP.NET, PHP, JSP, ASP, or Perl to name a few, is hosted by the web server. The dynamic web content technology is responsible for generating HTML content that is sent back by the web server to the client (as shown in Fig. 1. The third tier is the storage tier which is responsible for persisting the data, normally using a relational database.

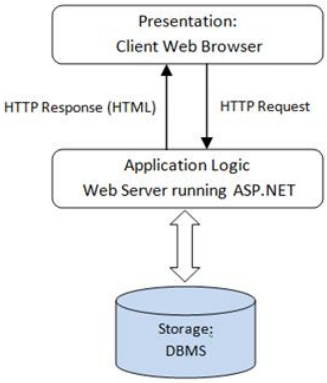


**Figure1**: Logical View of Web Applications

Dynamic web applications pose certain challenges for test engineers when automating functional tests using test tools through the GUI [21]. First: due to the fact that dynamic web pages change their content and/or layout from one page execution to another, automation testing tool's record/playback feature may fail to play back the test again successfully. This happens because during the test recording process, the test tool recognizes page elements (such as text fields, labels, lists, tables, cells in tables, menu items, links, etc.) according to their position in the page hierarchy and attribute values. However, when playing back the recorded test the requested page changes element attributes and even layout, causing some automation tools to fail finding those elements, thus, causing the test to fail [21].

The second challenge for web application automated testing is the ability for test tools to wait for web pages to load completely before starting to access page elements. As stated before, Ajax and scripts in the page can load additional page content or modify existing content after a page is returned back from the server [9]. When a test automation tool fails to wait for web page to load completely, it will start to search for web page elements before they are being loaded, causing the test to fail during playback.

These two challenges pose important reliability criteria for record/play features in automation tools and will be carefully tested and considered when comparing testing tools.

## 3. HTML TAGS

Web pages are presented as HTML mark-up. The HTML document consists of mark-up tags, which in turn describe how web pages will be presented in the web browser. HTML tags are keywords surrounded by angle brackets, such as <html>. Most tags come in pairs known as start and closing tags, such as  and  respectively [10]. Web page elements are everything between starting and corresponding closing tags, as shown in Figure 2.

```
<p>This is a paragraph </p>
```

**Figure 2:** Paragraph Tag.

Most HTML elements have attributes. Those attributes provide additional information about the element and are always specified in the opening tags. Attributes come in name/value pairs, such as id="txtName". An example is shown in Figure 3 with "href" as the attribute name.

```
<a href="http://www.myDomain.com">This is a link</a>
```

**Figure 3:** Link Tag.

Attribute **id** specifies a unique identifier for an element in a web page; but no two elements can have the same **id** value in same web page. Values for **id** attributes are normally provided by web programmers, and do not changed from one page run to another. However, not all web page elements have this attribute. This is due to the fact that web developers normally do not provide values for all web elements' **id** attributes; and because some web pages get generated dynamically.

Test automation tools for web applications work against HTML tags. Thus, it is important that these test tools provide additional mechanism to locate web page elements, other than using the **id** attribute. Such additional mechanisms can be using other combinations of attributes values to locate an element, or searching for an element

related to other elements in the page hierarchy (XPath).

## 4. TEST AUTOMATION

Having software to test software is called test automation. GUI test automation is an important part of software testing and provides software testers with early warning signs when parts of the system have changed or been broken. Constant testing is required to enable agile teams to release production-ready software regularly at the end of every sprint. In order to maintain agility, automation should be a critical component with high priority for the entire team. Agile testers should also have a team oriented approach and use a variety of testing tools within their testing efforts and activities [11].

### *4.1 Test Automation Types*

There are three main different types of automated tests according to [7]. The first category is the foundation and comprise of robust unit and component tests. They are generally written in the same programming language as the system with the help of unit test framework, such as JUnit which is used for Java unit testing and NUnit for .NET applications. Unit tests have the biggest return on investment (ROI) and provide the quickest feedback.

The second category of automation tests are the tests that operate on the API (Application Programming Interface) level or "behind the GUI". These tests target the functionality of system without going through the GUI. They usually include "story" and "acceptance" tests and cover larger sets of functionalities than unit tests. Since they bypass GUIs, they are less expensive to write and maintain compared with GUI tests. Tools can be used by test engineers in this category, such as Fit and FitNesse [7].

The third category is known as GUI automation tests, which are the focus of this paper. These tests operate on the presentation layer of the application through GUIs. They are usually written after code is completed, and run slower than unit or API tests. Automated GUI tests are important and should always go into regression tests. In fact, many software bugs only manifest themselves at the GUI. In many cases, a back-end change could affect GUI functionality [7]. Furthermore, GUI automation tests provide much more test integration compared to unit tests, since they test a whole series of events [12].

### *4.2 Importance of Automation Testing*

As argued by [7], there are many reasons why teams should employ automated testing. Firstly manual testing takes too long and gets longer and longer as software applications get bigger and bigger every sprint. In fact, depending on the complexity of the application under test, the time to test everything manually can get bigger exponentially. Test engineers need to run a full suite of passing test regressions daily. And this can never happen using manual testing. On the other hand, time is saved when employing automated testing because automated tests run faster than human tests, giving the ability to be run at night. Secondly, manual testing is error prone. As testing gets repetitive, it will get to be boring for test engineers and too easy to make mistakes, overlooking even simple bugs.

Another reason of using automated testing is that automation frees test engineers to do their best work. Test engineers will have more time to do important exploratory functional testing, exploring weak parts of the system and producing smart test cases. Moreover, automated testing provides a safety net. Knowing that the code has enough coverage by automated tests provides a feeling of confidence for development team [7]. Also, automated tests give early feedback to developers as running automated regression tests every time code is changed, gives early feedback to developers if that change breaks any functionality. This change will be fresh to programmers, and will not take much time to fix it again. Finally, automated tests are great documents of how the system actually works. This is due to the fact that automated tests will always be accurate since they are getting updated continuously by test engineers.

## 5. GUI TESTING TOOLS FOR WEB APPLICATIONS

Automation tools that automate GUI functional tests against web applications have been in the market for years, mostly equipped with an important feature known as record/playback. Automation tools work against web pages in HTML format. Using these tools, test engineers can record a functional test, and play it back as needed, possibly at regression tests. In this paper, the authors conduct comparisons between automation tools based on several important factors:

Firstly, it is important for an automation tool to have reliable record/playback feature especially against dynamic web pages. Once test engineers record a test, the tool should execute the test during test-playback without errors, in most cases. Secondly, test tools should be able to export resulting tests as programmable scripts so that test engineers

can easily understand, maintain, enhance and re-factor into modules to take advantage of future reusability. Using the resulting test script, test engineers can enhance it to address complex test cases and to employ data test-driven test scripts.

Thirdly, the diversity of techniques provided by test tool frameworks in order to find elements on web pages is an extremely important factor as well. This is due, as stated before; to the fact that dynamic web pages change their layout and element attributes form one page execution to another. The most important techniques for searching for web page elements are: searching by **id**, searching by any group of element attributes, searching in relevance to another element (searching by XPath), and searching by DOM (Document Object Model). For instance, sometimes test engineers have to write a test script that searches for an element that does not have a unique name or id. To solve this problem, using XPath, test engineer can locate that element relative to another element that does have an id or name attribute.

Another factor to consider is the ability for a test tool to support cross-browser compatibility. Many web applications have a priority feature to support different browsers, such as Internet Explorer, Mozilla Firefox and Google Chrome. And test engineers will have to automate functional tests in those browsers. Finally, test tools should be effective when it comes to waiting for web pages to load completely before they start accessing and manipulating web page elements.

Finally, test teams should carefully consider the various technical support services available by tool vendor and how much efficient, reliable and responsive those services are. Chances are very high that test teams run into an issue or problem while using the automation tool. An since the main reason for using automation testing is the speed and saving of time, test engineers cannot waste much time in developing a solution that someone else already knows.

Technical documentation is provided by all vendors and is considered important and first place to look at when searching for help. However, documentation is not usually enough and other technical support services should be provided by tool vendor. Such additional services are: forums, technical articles and bug tracker systems. Additionally, we found it extremely important that discussion forums not only include regular members (client), but also include technical personnel dedicated to answer enquiries posted by clients.

A variety of automation tools for web applications are available for test teams from open source and commercial sectors. This paper compares four different tools taken from both open source (Selenium and Sahi [13]) and from commercial fields (TestComplete and VS 2010 Web Tests).

**TestComplete**, a product of SmartBear, is one of the premium commercial automation testing tools, and most notable [14][15]. TestComplete can create, manage and run automated tests for any windows, web or rich client software. By using TestComplete, test engineers can perform several types of automated tests, such as functional Graphical User Interface (GUI) tests, regression tests, load and stress tests, unit tests and many more. Another reason for choosing TestComplete is that it provides testers with the ability to write test scripts from scratch using scripting language, such as Java Script. This ability enables testers to write complex and dynamic test scripts [21].

**Visual Studio Web Tests,** a product of Microsoft available since Visual Studio 2005, works at the protocol layer by issuing HTTP requests. Web Test has been chosen for this paper because it is famous among .NET developers since that it is shipped freely with the Visual Studio. When a tester records a test scenario, the web test records a series of HTTP requests and later, when performing a play-back, the web test executes those HTTP requests in the same order they were recorded. According to [16], Web Tests can be used to test the functionality of web applications as well as testing the application stress, which is also known as load testing. Web tests automatically handle other aspects of HTTP, such as hidden field correlation, redirects, dependent requests and HTTPS/SSL.

**Selenium** is an open source and free automation tool that can be used to record GUI tests against web applications. It is a very easy tool to learn and use, available as Firefox plug-in. During test recording, Selenium provides a context menu that allows testers to select web page elements, and then choose from a list of predefined commands to be applied on selected elements. This saves lots of time for the test engineer, and provides an easy way to learn generated scripts [17]. Apparently, Selenium has a reliable record/playback tool when run against dynamic web applications. Selenium also provides a wide range of techniques available for test engineers in order to locate web page elements. During test recording, Selenium automatically inserts commands into test scripts based on actions. For example, when a tester clicks on a link in a web page, Selenium inserts the *ClickAndWait* command so that the tool can wait until the page is loaded completely.

**Sahi,** another open source and free automation tool specially designed to support cross-browser testing for web applications with lots of AJAX and dynamic content [18]. Sahi injects JavaScript into web pages with the help of a proxy in order to automate web pages. Sahi seems to have a more reliable record/playback tool compared to TestComplete. Also, during the comparison, Sahi appeared to have much more effectiveness when processing web page waits (waiting for the web page to load completely). Test scripts are exported in JavaScript to enable good programming control [18]. Test scripts were very robust during comparisons; there was no need to script statements for page waits. Sahi supports data-driven tests, and can be easily connected to Excel, database or CSV files.

Many open source web testing tools exist. However, the selection from these tools was based on their ability to provide record/playback feature and the ability to generate test scripts that can be modified and enhanced later on by testers. Furthermore, Selenium and Sahi are highly recommended form developers and testers. Recommendations can be found in several well-known technical forums such as [20].

## 6. EXPERIMENT DESIGN AND METHODOLOGY

Since that dynamic web applications present significant challenge for test automation tools [21], the sample web application pages under test (HTML) have to be created at runtime by application itself. There are several programming languages and technologies that can be used to create dynamic web applications such as PHP, JSP (Java Server Pages), ASP (Active Server Pages), and ASP.NET to name a few. Any of these languages can be used for creating dynamic web site for the purpose of this study paper.

This is due to the fact that no matter what is the language or technology used on server, all what is returned to the client (Web Browser) is HTML. And automation testing tools run against HTML during test record and test play back. In this paper, ASP.NET with C# was chosen in this study since it provides object-oriented programming paradigm and because of related professional experience of authors.

### *6.1 Samples Web Application*

The sample web application under test was built using ASP.NET 3.5 in C# and SQL SERVER 2008 Express Database. Sample web application simulates a medical application for a clinic. Using this application, a doctor can track and maintain patients' medical and personal data. Doctor can add medical data such as incident, immunization, blood and alcohol tests as well as appointments.

When doctor first navigates to application URL, login page is displayed. Upon successful login, doctor is presented with dashboard page, which contains two parts. First part is a dynamically generated two-level main menu acting as main navigation panel. Second part is a group of data grids showing tabular information (DataGridView Server Control) about last visits (encounters), appointments and so on. With every grid having a search header to enable doctors to search for specific records. Doctor can also search for and maintain person information using "Person" link located on main navigation panel left screen. Please refer to Appendix A.

Having a detailed page for every entity in system (person, incident, appointment, etc), when a doctor clicks on certain row in a grid, detail entity information is displayed in another browser instance window. There, a doctor and update, insert new or delete entity data. It is important to open detail pages in separate browser window since some testing tools have shortages accessing web pages in several browser windows [21].

All grid view controls are built dynamically during run time when page is requested. This is done through specific class (GridGenerator) that, based on grid id, extracts related grid information such as data source, column names and properties from database, then builds grid in runtime. The main navigation panel is built dynamically as well, all navigation links are loaded from database. Author deliberately did not assign object names for links in main navigation panel. This is done to test the ability for test tool to locate web page elements that do not have an **id** attribute in their HTML tag.

The rationale behind having the grid view built dynamically is that many professional web developers in real-world applications require that all grid view controls across all pages to be consistent with same look and feel. This is usually achieved by having a general centralized routine that generates grid views at run time, taking the attributes (number of columns, column names, column types, styles, etc) from database. The result will be rendered as HTML table. However, the resulting cells and rows tags of that HTML table will normally have no **id** or "**name**" attributes. This will be challenging to automated test tools to allocate and access these tags. Sample code for creating dynamic grid view control is shown in code listing Figure 4.

```
private void CreateGridView(){
    // based on page name, the grid view attributes are
    // retrieved from database
   GridView grd = new GridView();//create grid view object
   for(int i = 1; i<=numberOfColumns; i++){
       BoundField field = new BoundField(); // create column
       field.DataField = columnName;
       field.HeaderText = columnName;
       grd.Columns.Add(field);
    }
    // get actual data from database
   DataTable dt = GetData(pageName);
    // Bind data to grid view control
    grd.DataDource = dt;
    grd.DataBind();
    //finally, add grid view control to web page
     this.Controls.Add(grd);
}
```

**Figure 4:** Sample Code for Building a GridView

### *6.2 Test Scenario Steps*

The test case scenario that will be applied to all test tools' recorder feature is as the following:

The tester runs sample web application and login page is displayed. Tester enters user name and password, and then clicks on login button. Upon successful login, dashboard is presented with all grid views showing sample data. Tester then clicks on "Person" link in main navigation panel left screen. System loads "Person Search" page and tester enters any search data and then clicks on search button. Page is reloaded and then grid shows rows matching search criteria. Tester clicks on first row and detail page for person data will be displayed. User will change some of the data in detail page then clicks save and close page. Finally, the search page is reloaded to reflect new data changes.

### *6.3 Methodology*

Applying an experiment research method, the authors implemented the comparison experiment dividing it into three main phases. First phase was concerned with comparing the effectiveness of record/playback feature for testing tools. During this phase, test scenario was recorded and played back for several times for each tool. If any errors were presented, the test log was carefully investigated to trace and identify the problem.

Second phase was based on modifying generated test scripts for each tool to explore various mechanisms to locate web page elements. During second phase, mechanisms to locate web page elements were applied to locate elements that do not have **id** attributes. Such elements were the rows and cells for DataGridViews. This was done by modifying test scripts to search for web page elements based on search by **id**, search by group of attributes (other than **id**), search by XPath and search by DOM. Test was run several times for each mechanism available.

Finally, the same test scenario was recorded and played back using Internet Explorer 9, Mozilla FireFox and Google Chrome. Results and observations were recorded based on several factor variables. First, the ability for test tool to successfully playback (execute) test script produced by record/playback feature successfully and without errors. Secondly, the ability for test tool to successfully allocate and access page elements that do not have **id** or "**name**" attributes. Finally, all tests were repeated using IE, Mozilla and Chrome web browsers.

## 7. COMPARISON OF RESULTS

### *7.1 Record/Playback Tool Reliability*

The record/playback tool on Selenium and Sahi appears to be more reliable, effective and time saving compared to TestComplete and VS Web Testing. In fact, recording and running tests using TestComplete's record/playback tool appears to be weak when it comes to dynamic web pages.

During the comparison, most tests recorded by TestComplete for dynamic web pages, failed to run again later on, because the TestComplete engine cannot recognize some of the onscreen elements, such as links, buttons, text fields, etc. As shown at Appendix A, TestComplete could not find "Person" link located at navigation panel.

SmartBear acknowledges this problem, as argued in [19]. This problem has actually been present since TestComplete 7.5 [21] and is still not solved in the current version of TestComplete 8.5, which is the version used in this paper.

This is due to the fact that, at the time of recording the test, TestComplete recognizes the web page elements through the values of a set of their attributes. Those attribute values are saved by TestComplete and used later on to find page elements when replaying the test. If one attribute for web page element changes its value, TestComplete will not recognize it and the test will fail. When the test is re-played and the tested web page is recreated, TestComplete engine records different values for some of the attributes for web page elements [19]. This problem can be easily proven using the object selector tool available by TestComplete itself. After the test fails because TestComplete could not find certain web page elements, test engineers can use the object selector tool and notice that some of the attributes for that web page element have changed.

TestComplete also was not robust when processing page waits. Test engineers have to write special scripts in order to make TestComplete engine waits for a web page to load completely before starting to access web page elements. On the other hand, Selenium and Sahi seems to be more reliable when waiting for pages to completely load before starting to access web page elements.

During the comparison, recording a web test in Visual Studio 2010 is relatively easy, and begins by starting Internet Explorer with an additional panel that represents the recorder tool itself. As the tester proceeds with the test scenario, the web test records all HTTP requests [12]. Web test is most suitable when performing simple functional tests and when testing availability and navigability of a web application. For instance, a tester can easily create a web test that tests the availability and links of all web pages for a web application [16].

However, web tests in Visual Studio do not provide the dynamic and rich programming features provided by other tools give examples of dynamic and rich features. It is true that a tester can create data-driven web tests and convert the recorded HTTP requests in C# language so as to add looping and branches: but the web test is only based on recording HTTP requests. Additionally, the resulting tests in C# are hard to understand and maintain compared to other tools in this paper.

### *7.2 Cross Browser Compatibility*

Cross browser compatibility was based on comparing tools based on compatibility for Internet Explorer (IE), Firefox and Google Chrome. TestComplete seemed to have less compatibility with web browsers, especially Internet Explorer (IE) when compared with Selenium and Sahi. This is due to the mechanism that TestComplete engine uses to access web pages, which is based on accessing IE run-time process, then access IE window, then accessing the web page itself. This mechanism produces a problem for the TestComplete playback tool because IE version 8 and above opens dictated process for each browser tab. Thus, TestComplete playback tool fails to find web pages because their processes order and name change from one test run to another and when tabs are opened and closed during testing. We had to manually configure IE to open one process for all tabs in order to make TestComplete playback more robust.

This is not a problem for Selenium and Sahi tools because these tools do not depend on run-time processes; they focus on web pages at hand and re-parse page HTML every time that page gets refreshed or updated. On the other hand, Web Tests in VS 2010 only supports Internet Explorer (IE) web browser. In fact, the recorder tool is presented as a plug-in for IE window.

### *7.3 Techniques for Finding Web Page Elements*

A summary of various techniques available in each tool for test engineers to find web elements in a web page is shown in Table 1.

**Table 1:** Summary of Various Mechanisms for Locating Web Page Elements in Compared Tools

| | Search Mechanism | | | |
|---|---|---|---|---|
| Tool | Element id | Group of Attributes | DOM | XPath |
| TestComplete | • | • | | • |
| VS Web Test | | | | |
| Sahi | • | • | • | • |
| Selenium | • | • | • | • |

### *7.4 Technical Support*

A summary of technical support services provided by each tool vendor is shown in Table 2.

**Table 2**: Summay of Technical Support Services

| | Technical Support Services | | | | |
|---|---|---|---|---|---|
| Tool | Documentation | Forums | Articles | Bug Tracking | Technical Personnel |
| Test Complete | • | • | • | | • |
| VS Web Test | • | • | • | • | • |
| Sahi | • | • | | | |
| Selenium | • | • | | • | • |

## 8. GUIDELINES FOR IMPLEMENTING GUI AUTOMATED TESTS FOR WEB APPLICATIONS

Record/Playback automation tools for web applications are plenty and appealing. Test engineers can create lots of scripts in a short time. With the help of automation tools, test teams can keep pace with the development in order to produce working software every sprint. However, there are several important guidelines that test teams and management should be aware of when adopting automation strategy.

Firstly, management needs to be aware that automation usually requires an initial investment in hardware and, probably, software tools. One of the advantages of automated testing is, as shown before, timely feedback through regression tests. In order to run daily regression tests, separate machines need to be configured for this purpose. Furthermore, test teams need the time to learn how to use selected automation tools and how to write and maintain test scripts.

Secondly, as discussed briefly in this paper, tool selection is vital to team success. Sophisticated and expensive commercial tools seem to put extra burden on testing rather than helping it. On the other hand, simple and free tools can be much more powerful and helpful. It is very important that test teams explore the capabilities of automation tools before selecting one.

Additionally, test scripts should be well designed to be simple and maintainable. GUI automation testing is tricky because GUI's tend to change during software development. It is very important for test engineers to design test scripts based on modules and common procedures [7]. For example, a common procedure that accepts a web page reference and

an array of attributes can save lots of effort and time when locating web page elements using any combination of attributes.

Another example would be, supposing that in a web application that has a common tool bar that contains certain functionality such as save, cancel, new, etc., since this tool bar will be available in several pages; it is much more helpful to design a common procedure to invoke certain functionality on that tool bar. Such a procedure can have a web page and functionality to be invoked as input parameters.

Common test script procedures should also be designed to be independent on certain test cases. This gives the flexibility for those procedures to be invoked by other test cases to promote reusability. Having test scripts written based on common procedures and modules will result in flexible and maintainable test scripts. Thus, whenever certain GUI changes are made by the development team, test engineers will be able to easily maintain tests to accommodate that new change. Additionally, when new a GUI is produced by the development team, test engineers can build test scripts in relatively short periods of time because these test scripts are based on common procedures.

Furthermore, test engineers should never hard-code page names and page URL's. Instead, this kind of data should be kept in resource files to enable test engineers to change their values without affecting the test script itself. And for large scale software projects, the test scripts will get extremely big and harder to understand. Thus, special and simple coding standard rules should be applied by test engineers to unify all test scripts. The most important things about these rules are having comments for all procedures and modules/classes, as well as having meaningful names for all variables and parameters.

For database-centric web applications, design test scripts test page save and update functionalities in both full and required-only data. The same test script should then retrieve data entered using application search features to make sure that the application saved the right data. For example, suppose in a human resource application, when testing the employee details page, test scripts should enter full data then save the new employee. After that, test scripts will search for that new employee, view all data, and compare data with the actual test data. These kinds of test scripts can cover a large spectrum of underlying application features and rules. Additionally, these kinds of scripts can be time consuming and error prone if they are done manually.

Automation tests are only good if they are likely to find bugs. Thus, test engineers should avoid testing full functionality of applications through GUI automated testing [12]. Test engineers should focus on testing the overall functionality of application, such as checking the buttons really work and do what they are supposed to do. Other types of test that should be automated are the ones that consume lots of time and effort when done through manual tests. Test engineers should avoid testing the usability of the application. Usability testing needs humans to actually use the software [7].

Finally test engineers have to make sure that developers give id for web pages' elements; this will make locating those elements much easier in test scripts. Moreover, tests for testing the links in all web application are a very important part of automation. There is no need for someone to manually check all links in web applications.

## 9. CONCLUSIONS

This paper compared several automation testing tools, namely: Sahi, Selenium, TestComplete and Visual Studio 2010, against dynamic web applications. Several comparison criteria were discussed in depth, such as reliability of recorder/playback tool, cross browser compatibility, handling of page waits, and the support of various mechanisms for locating web page elements.

The strengths, weaknesses and limitations of these tools where investigated in some depth. It has been shown that simple and free automation tools can be much more powerful than commercially sophisticated and expensive tools. This paper also discussed the importance of automated testing and why it is important. Furthermore, this paper presented vital guidelines for test teams when adopting automated testing against web applications. For future studies, the comparison of these tools can be expanded to include the support for third-party web GUI widgets, such as Telerik and Infragistics as well as the support of HTML 5.

## 7. REFERENCES

[1] Afroz, S., Rani E. &Priyadarshini, N. (2011) "Web Application – A Study on Comparing Software Testing Tools", *International Journal of Computer Science and Telecommunications,* 2(3).

[2] Business Internet Group of San Francisco (2003) *The BIG-SF Report on Government Web ApplicationIntegrity*. Available at: http://www.tealeaf.com/downloads/news/analyst_report/BIG-SF_Report_Gov2003-05.(Accessed: 9 May 2011).
[3] Business Internet Group of San Francisco (2003) *The Black Friday Report on Web Application Integrity*. Available at: http://www.tealeaf.com/downloads/news/analyst_report/BIGSF_BlackFridayReport.pdf. (Accessed: 9 May 2011).

[4] Brooks, P., Robinson, B., and Memon, A. M.(2009) "*An initial characterization of industrial graphical user interface systems*," in ICST 2009: Proceedings of the 2nd IEEE International Conference on Software Testing, Verification and Validation. Washington, DC, USA: IEEE Computer Society, 2009.

[5] Xun Yuan, Myra, B. Cohen, Atif M Memon, (2011) "GUI Interaction Testing: Incorporating Event Context*"*, *IEEE Transaction on Software Engineering*, 37(4): 559-574.

[6] Sommerville, I. (2004) *Software Engineering*, England: Addison Wesley.

[7] Crispin, L. & Gregory, J. (2009) *Agile Testing: A Practical Guide for Testers and Agile Teams*, Boston: Addison Wesley.

[8] Wikipedia (2012) *Web Applications.*Available at:http://en.wikipedia.org/wiki/Web_application (Accessed: 5 May 2012).

[9] SmartBear Software (2011) *Testing Dynamic Web Pages.*Available at:
http://smartbear.com/support/viewarticle/12725/
(Accessed 16 April 2011).

[10] w3schools.com (2012) *Learn HTML* [Online]. Available at: http://www.w3schools.com/html/html_intro.asp
(Accessed: 14 March 2012).

[11] West, J. (2012) "*Automated Testing in Agile Environments*", [Online]. Available at:http://support.smartbear.com/articles/testcomplete/automated-testing-agile-environment/ (Accessed: 10 May 2012).

[12] Levinson, J. (2011) "*Software Testing With Visual Studio 2010*", Boston: Pearson, Inc.

[13] Tyto Software Ltd (2012) *Sahi*[online]. Available at: http://sahi.co.in/w/sahi (Accessed 6 May 2012).

[14] Riley, M. (2010) *DevProConnections*. Available at: http://www.devproconnections.com/article/software-testing/Review-SmartBear-Software-s-TestComplete-8-Enterprise (Accessed 15 April 2011).

[15] SmartBear Software (2011) *Top Reasons to Try TestComplete,* [Online]. Available at:http://www.automatedqa.com/products/testcomplete/top-reasons-to-try/ (Accessed: 12 May 2011).

[16] Microsoft (2012) *Understanding Web Tests* [Online]. Available at: http://msdn.microsoft.com/en-us/library/ms182537%28v=vs.90%29.aspx (Accessed 2 March 2012).

[17] Selenium IDE (2012) *Introduction to Selenium IDE* [Online]. Available at:
http://seleniumhq.org/docs/02_selenium_ide.html#script-syntax (Accessed 3 March 2012).

[18] Java-Source.net (2012) *Open Source Web Testing Tools in Java*[Online]. Available at: http://java-source.net/open-source/web-testing-tools (Accessed 17 March 2012).

[19] SmartBear Software (2011) *Testing Dynamic Web Pages.* Available at:
http://smartbear.com/support/viewarticle/12725/
(Accessed 16 April 2011).

[20] Stack Overflow (2012) *Stack Overflow is a programming Q & A site that's free.* Available at:
http://stackoverflow.com (Accessed 16 January 2012).

[21] Zain, S., Elyan, D. (2012) 'Automated User Interface Testing for Web Applications, and TestComplete', *CUBE 2012: International IT Conference and Exhibition.* Pune, India 3-5 September. ACM, pp. 350-354.