A Review of Scripting Techniques Used in Automated Software Testing

Milad Hanna
Department of Computer Science,
Faculty of Computers and Information,
Helwan University,
Cairo, Egypt

Nahla El-Haggar
Lecturer of Information Technology,
Faculty of Computers and Information,
Helwan University,
Cairo, Egypt

Mostafa Sami
Professor of Computer Science, HCI lab, Faculty of Computers and Information,
Helwan University,
Cairo, Egypt

Abstract— **Software testing is the process of evaluating the developed system to assess the quality of the final product. Unfortunately, software-testing process is expensive and consumes a lot of time through software development life cycle. As software systems grow, manual software testing becomes more and more difficult. Therefore, there was always a need to decrease the testing time. Recently, automation is as a major factor in reducing the testing effort by many researchers. Therefore, automating software-testing process is vital to its success. This study aims to compare the main features of different scripting techniques used in process of automating the execution phase in software testing process. In addition, an overview of different scripting techniques will be presented to show the state of art of this study.**

Keyword— Software Testing; Automated Software Testing; Test Data; Test Case; Test Script; Manual Testing; Software Under Test; Graphical User Interface.

I. INTRODUCTION

Software testing has evolved since 1970's as an integral part of software development process. Through it, the final quality of the software can be improved by discovering errors and faults through interacting, checking behavior and evaluating the System Under Test (SUT) to check whether it operates as expected or not on a limited number of test cases with the aim of discovering errors that are found in the software and fixing them. According to Ilene Burnstein, software testing describes as a group of procedures carried out to evaluate some aspect of a piece of software [1]. Ehmer Khan [2] shortly defines it as a set of activities conducted with the intent of finding errors in software. In addition, according to Ammann and Offutt [3] software testing means evaluating software by observing its execution.

Since software-testing process is a very expensive process, complete testing is practically impossible and it is not acceptable to reduce testing effort by accepting quality reductions. Testing effort is often a major cost factor during software development. Many software organizations are spending up to 40% of their resources on testing [4]. Therefore, an existing open problem is how to reduce testing effort without affecting the quality level of the final software.

Automation is one major solution for reducing high testing effort. Automating certain manual tasks from software testing process can save a lot of testing time. It can help in performing repetitive tasks more quickly than manual testing.

II. MANUAL TESTING VS. AUTOMATED TESTING

Software testing can be divided into two main categories, manual testing, and automated software testing. Both categories have their individual strengths and weaknesses.

With a manual testing, the more traditional approach, tester initiates each test, interacts with system, reports and evaluate the test results. To satisfy the test results manually, testers should prepare and execute test cases on SUT. These test cases will best test the system using defined processes trying to find bugs. So, they can be fixed before releasing the product to the public [5].

Automation is one of the more popular and available strategies to reduce testing effort. It develops test scripts that will be used later to execute test cases instead of human [6]. The idea behind automation is to let computer simulate what the tester is doing in reality when running test cases manually on SUT. AST is more suitable for repetitive tasks during different testing levels such as regression testing, where test cases are executed several times whenever the source code of SUT is modified or updated [7].

Katja Karhu [5] summarizes the difference between the two categories by suggesting that automated software testing should be used to prevent new errors in the already tested working modules, while manual testing is better used for finding new and unexpected errors. The two approaches are complementary to each other, automated testing can perform a large number of test cases in little time, whereas manual testing uses the knowledge of the tester to target testing to the parts of the system that are assumed to be more error-prone.

III. SCRIPTING TECHNIQUES

Test scripts are the basic element of automation. Test script is a series of commands or events stored in a script language file to execute a test case and report the results. It may contain logical decisions that affect the execution of the script, creating multiple possible pathways, constant values, variables whose values change during playback. The advantage of test scripts development process is that scripts can repeat the same instruction many times in loops, each time with different data. There are many types of scripting techniques that can be used in automation. Fewster and Graham [8] listed five different types of scripting techniques that will be discussed in this section.

A. *Linear Scripting Technique*

John Kent [9] explains the idea behind linear technique, which is simply to set the test tool to the record mode while performing actions on the SUT. The generated recorded script consists of a series of testing instructions using the programming language supported by the tool. Gerald Everett suggested that the linear scripts are being created by recording the actions that a user performs manually on interface of the system and then saving test actions as a test script. These test scripts can then be replayed back to execute the test again. So, linear scripting technique is called Record/Playback [10]. 2Figure 1 illustrates record/playback steps.

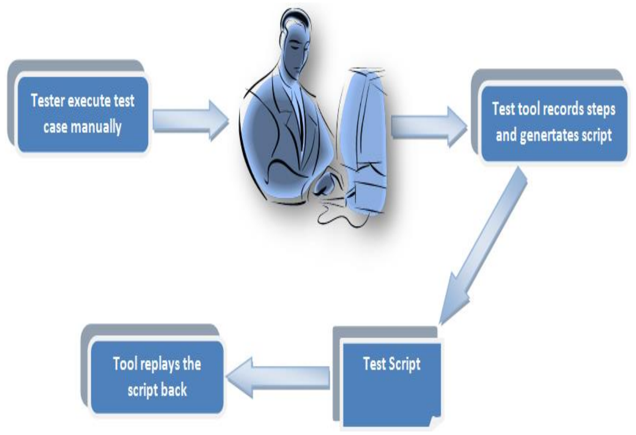

Tester execute test case manually
Test tool records steps and genertates script
Test Script
Tool replays the script back

Figure 1: Record/Playback Steps

Microsoft® Visual Studio® Team Edition is an example for tool applying linear scripting technique. It enables testers to perform record and playback to be used to create and execute the tests [11].

Every time a test case is being automated using linear technique, new test script is generated. Thus, the more test cases are automated, the more lines of code are generated. This means that the number of Lines of Code (LOC) is proportional to the number of automated test cases [9]. Thus: Lines of code α Number of automated test cases

Figure 2 and Figure 3 shows a practical example for applying linear scripting technique on a simple login page, which followed by the recorded test script that presents the sequence of actions performed manually on that page.

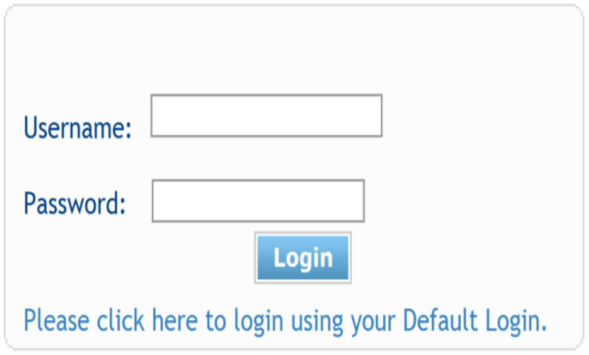


Figure 2: Login Page

```
                    //Filelds
/// <summary>
/// Go to web page 'http://TestSite/login.aspx' using new
browser instance
/// </summary>
public string UIWelcometoTestSiteWinWindowUrl
="http://testsite/login.aspx";

/// <summary>
/// Type 'test-user' in 'txtUserName' text box
/// </summary>
public string UITxtUserNameEditText = "test-user";

/// <summary>
/// Type '{Tab}' in 'txtUserName' text box
/// </summary>
public string UITxtUserNameEditSendKeys = "{Tab}";

/// <summary>
/// Type '********' in 'txtPassword' text box
/// </summary>
public string UITxtPasswordEditPassword =
"to+VpC5U2lKdiNhE9v4dzPA0ZmKuc60K";

/// <summary>
/// Type '{Enter}' in 'txtPassword' text box
/// </summary>
public string UITxtPasswordEditSendKeys = "{Enter}";

                    //Actions
// Go to web page the webpage using new browser instance
this.UIWelcometoTestSiteWinWindow.LaunchUrl(new
System.Uri(this.LoginParams.UIWelcometoTestSiteWinWindowU
rl));

// Type 'test-user' in 'txtUserName' text box
uITxtUserNameEdit.Text =
this.LoginParams.UITxtUserNameEditText;

// Type '{Tab}' in 'txtUserName' text box
Keyboard.SendKeys(uITxtUserNameEdit,
this.LoginParams.UITxtUserNameEditSendKeys,
ModifierKeys.None);

// Type '********' in 'txtPassword' text box
uITxtPasswordEdit.Password =
this.LoginParams.UITxtPasswordEditPassword;

// Type '{Enter}' in 'txtPassword' text box
Keyboard.SendKeys(uITxtPasswordEdit,
this.LoginParams.UITxtPasswordEditSendKeys,
ModifierKeys.None);
```

Figure 3: Linear Script for Login Page

John Kent [9] mentioned main advantages for linear scripting technique as listed below:

- It enables tester to start automating quickly as no planning is required, tester can just simply record any manual test case.

- The tester does not need to have any programming skills.
- It is good for demonstrating the SUT.

John Kent [9] mentioned the shortcomings for linear scripting technique as listed below:

- The generated scripts are very difficult to be maintained because they are made up of long lists of actions of objects interacting with interface, it contains its own hard-coded data, and this is not the best way for saving them.
- The recorded script can only work under exactly the same conditions as when it was recorded at the first time. If simple error happened or unexpected normal events (e.g. file not found) during a test run, it will not be handled correctly by the test script.
- Linear test scripts are not reliable enough, even if the application has not changed. They often fail on replay because other things occurred that did not happen when the test was recorded.

B. Structured and Shared Scripting Techniques

Both structured and shared scripting techniques are being formed by using structured programming instructions that are used to control the flow of execution of the script.

Structured scripting technique uses structured programming instructions, which either be control structures or calling structures [8]. Control structures is used to control the different paths in the test script (e.g. If condition). Calling structures is used to divide large scripts into smaller and more manageable scripts. For example, one script can call another script to perform specific functionality and then return to the first script where the subscript was called. The most important advantage of structured technique is that the test script can validate for specific conditions to determine if the executed test passed or failed according to these conditions. However, the script has now become a more complex program and the test data still tightly coupled within the test script itself. Besides, implementing structured scripts require not only testing skills but also programming skills [8].

Figure 4 shows applying structured scripting technique on a simple login web page.

Shared scripting technique enables common actions to be stored in only one place. This implies that a scripting language that allows one script to be called by another one is required. The idea behind shared scripts is to generate separate script that performs one specific common task that other scripts may need to perform later.

Thus, different test scripts can call this common task whenever they needed and testers will not have to spend time for implementing common actions many times across all scripts [12]. It works well for small-scale systems to be tested using relatively few test scripts. Figure 5 illustrates using shared scripting technique [12].

```
 [TestMethod]
public void Login_TestMethod()
{
    WatiN.Core.Settings.WaitForCompleteTimeOut = 120;
    IE ie = new IE("http://testsite/login.aspx", true);
    ie.TextField(Find.ById("txtUserName")).Value = "test-
user";
    ie.TextField(Find.ById("txtPassword")).Value =
"12345678";
    ie.Button(Find.ById("btnLogin")).Click();
    ie.WaitForComplete();
     // If "Welcome" message is displayed, then the test
is passed
     if (ie.Text.Contains("Welcome"))
     {
            Console.WriteLine("Testing Passed");
     }
     else
     {
          //If not, then the test is failed
          Console.WriteLine("Testing Failed");
      }
}
```

Figure 4: Structured Script for Login Page

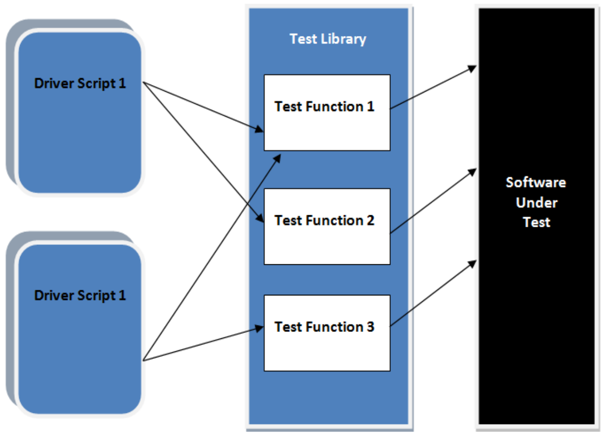
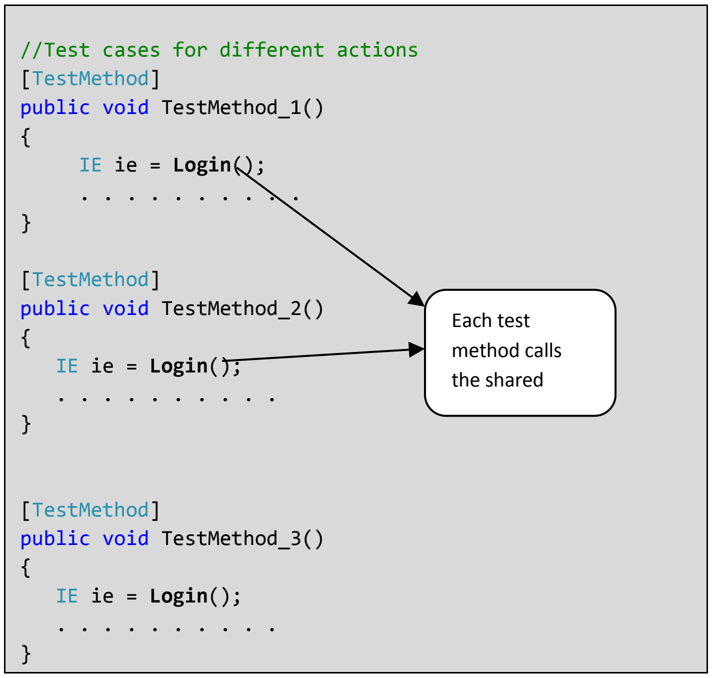


Figure 5: Driver Scripts and a Test Library

For example, instead of having the same login action repeated in a number of scripts, tester could simply implement it once as shared script and each test script just have to call this common function as illustrated in Figure 6:

```
public IE Login()
{
      WatiN.Core.Settings.WaitForCompleteTimeOut = 120;
      IE ie = new
IE(Telco_Automation.Properties.Settings.Default.SiteURL,
true);
      ie.TextField(Find.ById("txtUserName")).Value =
"test-user";
      ie.TextField(Find.ById("txtPassword")).Value =
"amv1234!@#$";
      ie.Button(Find.ById("btnLogin")).Click();
      Assert.IsTrue(ie.Text.Contains("Welcome"));
    return ie;
}
```



```
//Test cases for different actions
[TestMethod]
public void TestMethod_1()
{
     IE ie = Login();
     . . . . . . . . . .
}

[TestMethod]
public void TestMethod_2()
{
   IE ie = Login();
   . . . . . . . . . .
}


[TestMethod]
public void TestMethod_3()
{
   IE ie = Login();
   . . . . . . . . . .
}
```

Figure 6: Shared for Login Page

C. *Data-Driven Scripting Technique*

New additional scripting techniques are required to form test scripts in such a way that the maintenance costs of the test scripts can be reduced than in the previous scripting techniques. Data-Driven scripting technique proposes better organization of test scripts and hence lower maintenance costs of the test scripts. Bhaggan [13] demonstrates that test data is stored in a separate data file instead of being tightly coupled to the test script itself. While performing tests, test data is read from the external data file instead of being taken directly from the script itself. It allows both input data and expected results to be stored together separately from the script itself. For example, instead of having username and password data input values within the login script, these values can be stored in an external excel file and implement test script to read test data to use it while executing the test script.

In this technique, it is important that the external data file must be synchronized with the control script. This means that if any changes applied to the format of the data file, then the control script must be updated also to correspond to it.

To automate new test case, new control script has to be implemented with new data records inserted into external data file. Figure 7 illustrates data-driven scripting technique [12].

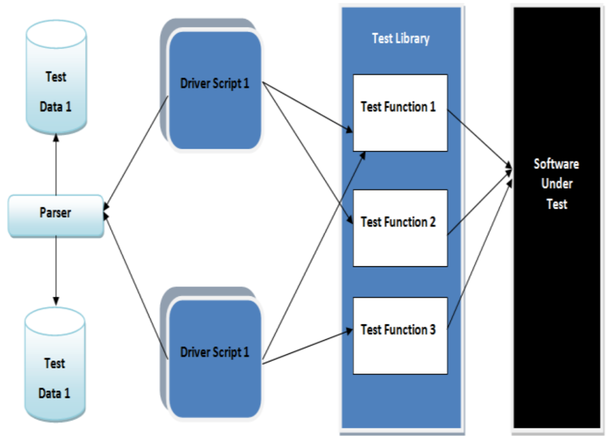


Figure 7: Data-Driven Scripting Technique

In data-driven scripting technique, the maintenance costs are lower than the costs of rerecording the tests from the beginning. Therefore, tests will not have to be rerecorded, but only maintained [13].

Linda G. Hayes [14] presents the main advantages of this approach as below:

- Similar tests can be added very quickly with different input data as the same script can be used to run different tests with different data. xIt may useful when testing large number of data values using the same control script.
- Data files are stored in easily and maintainable text records, so it can be updated.
- The format of data files can be modified to suit the testers with some modifications in the control script. For example, the data file can contain special column for comments that the control script will ignore while execution. This make the data file more readable, understandable and therefore maintainable.

The disadvantages of data-driven technique by Linda G. Hayes [14] are listed below:

- It requires high level of programming technical skills in the scripting language supported by the tool. Such tests need to be well managed, as it requires maintaining data files used by various test scripts. This may increase the cost for the project.
- One script is needed for every logically different test case. This can easily increase the amount of needed scripts dramatically. Laukkanen considered that this is the major problem in this technique [12].

The following test script with the external data file show applying data-driven scripting technique on a simple web page in Figure 8.

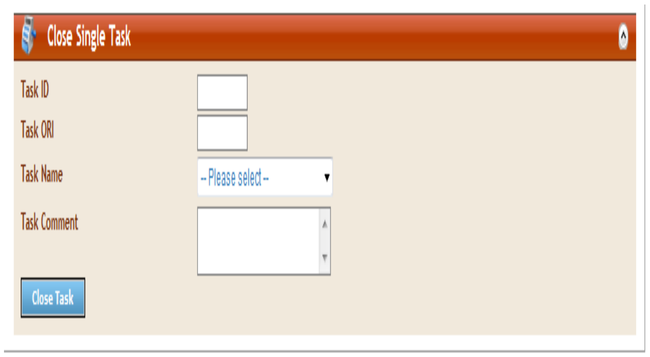
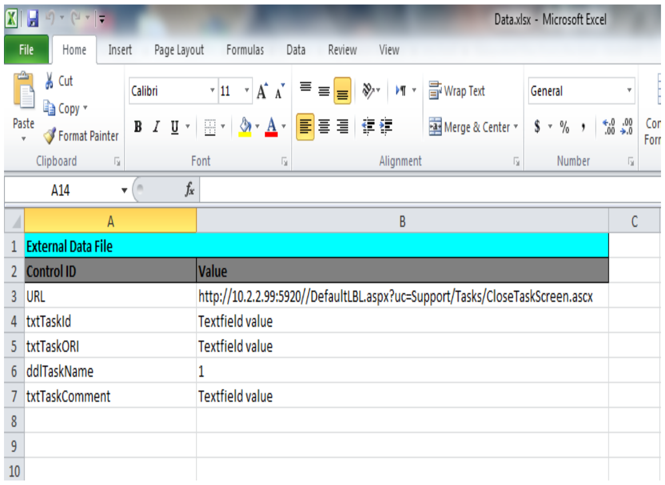


Figure 8: Sample Web Page

```
[TestMethod]
public void GeneratedTestMethod()
{
IE ie = new IE();
Application xlApp = new Application();
Workbook xlWorkbook =
xlApp.Workbooks.Open(@"D:\Data.xlsx");
Worksheet xlWorksheet = xlWorkbook.Sheets[1];
string url = ((Range)xlWorksheet.Cells[3,
2]).Text.ToString();
ie.GoTo(url);

ie.TextField(Find.ById(new Regex("txtTaskId"))).Value =
((Range)xlWorksheet.Cells[4, 2]).Text.ToString();

ie.TextField(Find.ById(new Regex("txtTaskORI"))).Value =
((Range)xlWorksheet.Cells[5, 2]).Text.ToString();

ie.SelectList(Find.ById(new
Regex("ddlTaskName"))).Options[int.Parse(((Range)xlWorksh
eet.Cells[6, 2]).Text.ToString())].Select();

ie.TextField(Find.ById(new
Regex("txtTaskComment"))).Value =
((Range)xlWorksheet.Cells[7, 2]).Text.ToString();

ie.Button(Find.ById(new Regex("btnClose"))).Click();
}
```

Figure 9: Generated Test Script for the Web Page



	A	B	C
1	External Data File		
2	Control ID	Value	
3	URL	http://10.2.2.99:5920//DefaultLBL.aspx?uc=Support/Tasks/CloseTaskScreen.ascx	
4	txtTaskId	Textfield value	
5	txtTaskORI	Textfield value	
6	ddlTaskName	1	
7	txtTaskComment	Textfield value	
8			
9			
10			

Figure 10: Output Data File Snapshot for the Web Page

D. *Keyword-Driven Scripting Technique*

Keyword-Driven scripting technique is a very similar to manual test cases. The business functions of the SUT are stored in a tabular format as well as in step-by-step instructions for each test case. Keyword-driven approach separates not only test data for the same test as in data-driven scripts but also special keywords for performing business function in the external file. The tester can create a large number of test scripts simply using predefined keywords. All what the tester needs is just to know what keywords are currently available to be applied on SUT and what is the data that each keyword is expecting. Additional keywords can be added to the list of available programmed set of keywords to enlarge the scope of automation. It is more sophisticated than data-driven technique [12]. Fewster and Graham [8] state that the keyword-driven scripting technique is a logical extension of the data-driven scripting technique. A limitation of the data-driven technique is that the detailed steps of what the tests are doing are implemented within the control script itself. Therefore, keyword-driven technique takes out some of the intelligence from the script, put it into the external file with the test data, and leaves the task for reading both steps and data for the control script. Thus, instead of having data file in data-driven, complete test file is needed in keyword driven scripting technique. It doesn't contain test data only but also a complete description of the test case to be automated using a set of keywords to be read and interpreted later on while test case execution. The test file states what the test case will do, not how to do it.

Laukkanen [12] supposed that in order to execute the tabular automated test cases, there have to be a middle layer that converts the special keywords to the source code that interacts with SUT (the source code that implements the keywords are called “handlers”). The translation of keywords is implemented outside of the control script itself. Now, the control script only reads each keyword in order from the test file and calls corresponding supporting script. In addition, a driver script, which parses the test data and calls the appropriate keyword handlers, is needed. Figure 11 demonstrates these layers.

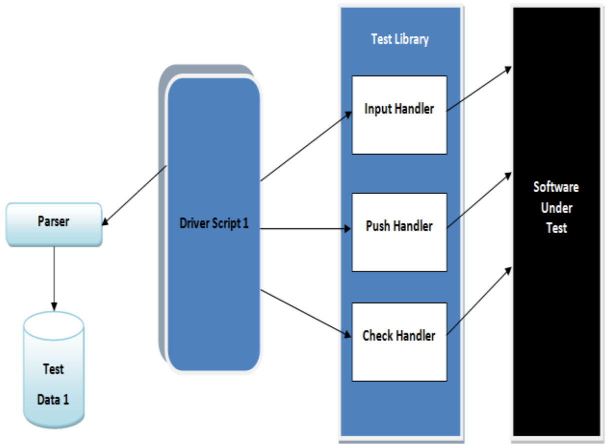


Figure 11: Handlers for Keywords

He also divides the keywords into two different levels of test keywords: high level and low-level keywords. Low-level keywords are more suitable for detailed testing on the interface level (e.g. Input, Click, and Select…etc.). High-level keywords are more suitable for testing higher-level functionality like SUT business logic (e.g. Create Account, Login…etc.). Multiple low level keywords can be combined together to form high-level keywords [12].

Zylberman and Shotten [7] show that keyword-driven technique is the next generation approach of automation that separates the task of automated test case implementation from the automation infrastructure. They state that keyword driven testing can be divided into two main layers:

1. Infrastructure Layer: It is a combination of the three types of keywords. It receives the different keywords as inputs to perform operations on the SUT.
2. Logical Layer: This layer helps manual testers to build new test scripts using the pre-defined keywords (that is already implemented in Infrastructure Layer).

They also divide the keywords into three different kinds (item/base level keywords, utility functions, and sequence/user keywords) which described below [7]:

1. Item Operation: an action that performs a specific operation on a given GUI element. Parameters should be specified to perform an operation on a GUI item such as name of GUI item, operation to be performed and the values needed.
2. Utility Functions: a script that executes a certain functional operation that is hard or ineffective to implement as a sequence. For example: Run Application, Close Application, Wait X seconds, Retrieve Data from DB,...etc
3. Sequence: a set of keywords that produces a business process such as “create customer” keyword. Sequence keyword is made by combining various items and functions.

They also suggest reducing the number of keywords by creating multi-function keywords. For example, "Update_Subscriber_Status" keyword is a better approach than creating two special keywords for "Activate_Subscriber" and "Deactivate_Subscriber" [7]. Although it can be argued that may be it is more useful not to combine keywords together because this allows using them again in creating another test script. For example, tester can use "Deactivate_Subscriber" keyword in another sequence of keywords (e.g. Delete_Subscriber).

Rantanen [15] suggests a new method for dividing system to multiple user stories. Each user story consists of one or multiple test cases. Each test case is to be mapped to the actual code interacting with the SUT while execution. Every test case contains one or more sentence format keywords. Every sentence format keyword consists of one or more user keywords which written in understandable text (they can be understood without technical skills). A user keyword consists of one or more base keywords. Finally, the base keywords contain the source code interacting with system to be tested. Figure 12 illustrates dividing SUT to multiple user stories.

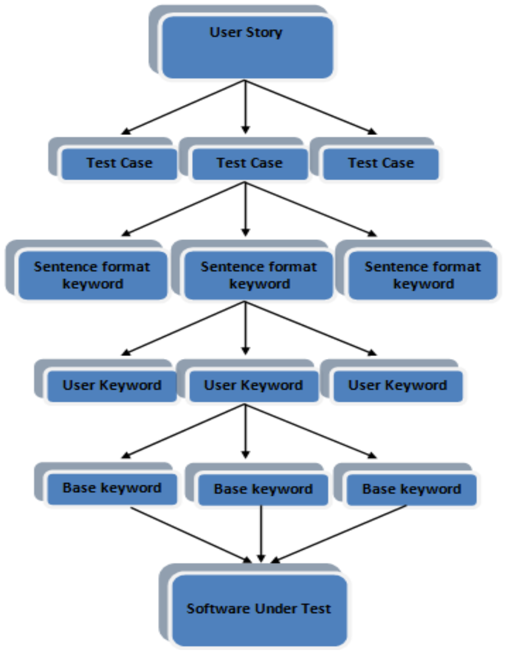


Figure 12: Mapping from User Story to SUT

Rashmi and Bajpai [16] proposed a new contribution for keyword-driven framework based on the concept of recording as shown in Figure 13. To start automating process, enter the URL of the system to be tested. Like linear scripting technique, the keyword driven testing framework records the steps while user navigates the web application manually. The user name and password are entered in the appropriate text boxes, and then the user clicks the Log-In button. The tool records all the operations performed manually in the web browser until the test is stopped.

When finishing the recording, a corresponding test script file is generated that contains all user actions. The user actions consists of items clicked, items selected and value typed…etc. These steps are generated in tabular format, representing each operation performed in the form of keyword, value and operation.

When the test is finished and replayed back, the tool runs keywords that were saved in the output test script file. The SUT opens in a new web browser and all recorded steps are performed again automatically, as it was originally performed manually in the test.

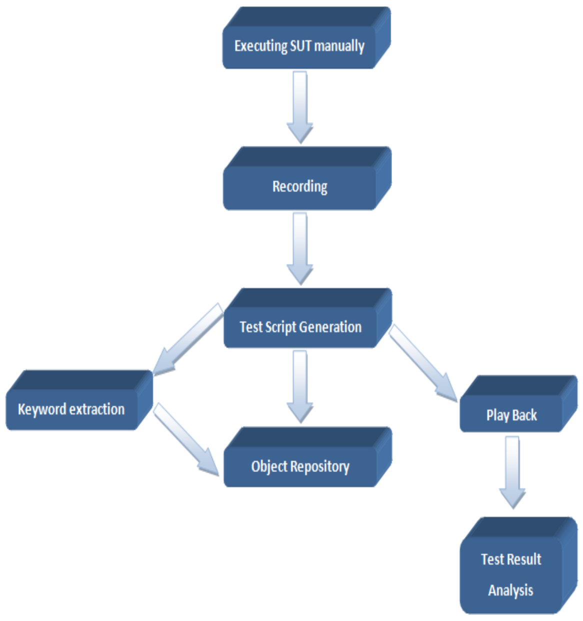


Figure 13: Keyword Driven Framework Based on Recording

After the new test run is completed, the test results are displayed to indicate the status of the test whether the test is passed or failed. The test results window displays two key elements of the test run for analysis purpose. The first one presents the steps that were performed while test execution while the second element presents the test result details.

Object Repository is a centralized place for storing the properties of available objects in SUT. Websites are developed using many different objects (e.g. textbox control, input tag). Each object is identified based on the object type. It has properties (e.g. name, title, caption, color, and size) and specific set of methods, which help in object identification.

Wissink and Amaro [6] state that the principal feature of the keyword-driven scripting technique is the separation of engineering tasks into a set of roles. These roles include test designer, automation engineer, and test executor. To automate test cases in the keyword-driven scripting technique, the next steps are to be followed:

1. A set of actions need to be defined by the *test designer* and then documented in an external file with other keywords, input data, and expected results.
2. The *automation engineer* implements the different keywords defined above by the test designer in the programming language of the tool.
3. The *test executor* just runs the tests directly from the spreadsheet.

The advantages of keyword-driven scripting technique mentioned by Linda G. Hayes [14] are:

- Using keyword-driven scripting technique, the tester only needs to know keywords and learn how to use them.
- The number of generated scripts required for keyword-driven is dependent of the size of the SUT rather than the number of tests. This means that many more tests can be created without increasing the number of scripts.
- Like data-driven scripting technique, the way in which tests are created can be modified to suit the testers rather than the test tool, using the format and tools that the testers are most comfortable with.

The disadvantages of keyword-driven scripting technique mentioned by Linda G. Hayes [14] are:

- The costs for development of customized application specific functions (framework) are very high in terms of both time and human resources for technical skills. Such specific framework development can be considered as standalone software development that needs to be tested before using in testing other software.
- If the SUT requires more than just a few customized keywords, then testers should learn a high number of keywords.

IV. Discussion

According to the above review of the paper about the different scripting techniques demonstrated by Figure 14, which illustrates moving from linear to keyword scripting technique in addition to a comparison of the main features for each of them as in Table 1. We recommend applying the data-driven scripting technique for automating the execution phase through software testing process as it is considered as the most cost effective scripting technique.

It is necessary to spend time building the test to avoid high maintenance costs on the long run. If the tester spends more time to develop test scripts, maintenance cost will be lower. However, if tester uses the fastest way to create test scripts (record/playback), then the maintenance cost will be very high. The following Table 1 and Figure 14 present a comparison between different scripting techniques. Numbers used in the table range from 1 (Lowest) to 5 (Highest).

TABLE 1: COMPARISON BETWEEN DIFFERENT SCRIPTING TECHNIQUES

Property	Linear	Structured	Shared	Data-Driven	Keyword-Driven
Ability to use reusable functions	No	No	Yes	Yes	Yes
Data separation from test script	No	No	No	Yes	Yes
Logic steps separation from test script	No	No	No	No	Yes
Access to code required	No	Yes	Yes	Yes	Yes
Use structured programming instructions	No	Yes	Yes	Yes	Yes
Ability to compare test results with expected	No	Yes	Yes	Yes	Yes
Ability to using script in regression testing	No	Yes	Yes	Yes	Yes
Special framework required	No	No	No	No	Yes
Programming skills level	1 (Low)	2	3	4	5 (High)
Effort needed to create test script	1 (Low)	2	3	4	5 (High)
Maintenance costs needed to update test script	5 (High)	4	3	2	1 (Low)
Reusability of test script	1 (Low)	2	3	4	5 (High)

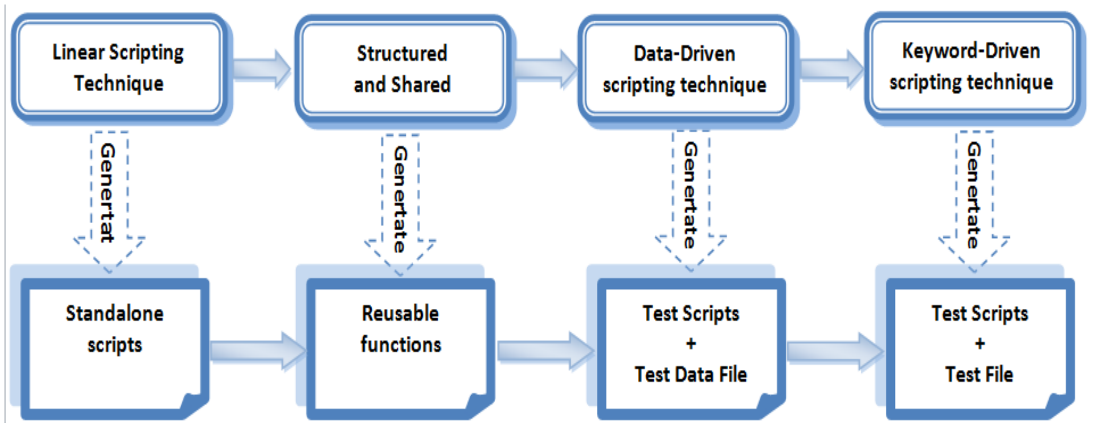

Linear Scripting Technique
Structured and Shared
Data-Driven scripting technique
Keyword-Driven scripting technique
Genertat
Genertate
Genertate
Genertate
Standalone scripts
Reusable functions
Test Scripts + Test Data File
Test Scripts + Test File

Figure 14: Evolution of Test Automation

V. CONCLUSION

Across many organizations, it is well known that testers lack the time needed to fully test the SUT within the time allocated to testing phase. This often happens because of unexpected environmental problems or problems in the implementation phase of development process. This normally shifts the software final delivery date. As a result to this delay, only two options is found, either to work longer hours or to add other resources to the test team to finalize testing in the required limited time. Automation can be one solution to this problem to accelerate testing and meet project deadline. Automation of testing phase offers a potential source of savings across all the life cycle. Automation using scripting techniques can save the costs for the overall software testing automation process, improve the speed of testing, shorten the product's launch cycle and it can achieve an amount of work that manual tests are impossible to finish.

REFERENCES

[1] I. Burnetein, "Practical Software Testing: process oriented approach," Springer Professional Computing, 2003.

[2] M. E. Khan, "Different Forms of Software Testing Techniques for Finding Errors," International Journal of Software Engineering (IJSE), vol. 7, no. 3, 2010.

[3] P. Ammann and J. Offutt, Introduction to Software Testing, New York: Cambridge University Press, 2008.

[4] F. Elberzhager, A. Rosbach, J. Münch and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," Information and Software Technology 54, p. 1092–1106, 2012.

[5] K. Karhu, T. Repo and K. Smolander, "Empirical Observations on Software Testing Automation," International Conference on Software Testing Verification and Validation, 2009.

[6] T. Wissink and C. Amaro, "Successful Test Automation for Software Maintenance," in 22nd IEEE International Conference on Software Maintenance (ICSM'06), 2006.

[7] A. Zylberman and A. Shotten, "Test Language: Introduction to Keyword Driven Testing," http://SoftwareTestingHelp.com, pp. 1-7, 2010.

[8] M. Fewster, Software Test Automation: Effective Use of Test Execution Tools, Addison-Wesley Professional, 1999.

[9] J. Kent, "Test Automation From RecordPlayback to Frameworks," http://www.simplytesting.com/, 2007.

[10] M. Fewster, "Common Mistakes in Test Automation," Grove Consultants, 2001.

[11] "How to: Generate a Coded UI Test by Recording the Application under Test," August 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/dd286608%28v=vs.100%29.aspx.

[12] P. Laukkanen, "Data-Driven and Keyword-Driven Test Automation Frameworks," Helsinki University of Technology, Software Business and Engineering Institute, 2007.

[13] K. Bhaggan, "Test Automation in Practice," Delft University of Technology, the Netherlands, 2009.

[14] L. Hayes, The Automated Testing Handbook, Automated Testing Institute, 2004.

[15] J. Rantanen, "Acceptance Test-Driven Development with Keyword-Driven Test Automation Framework in an Agile Software Project," Helsinki University of Technology, Software Business and Engineering Institute, 2007.

[16] N. Bajpai, "A Keyword Driven Framework for Testing Web Applications," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 3, 2012.