Reducing Testing Effort using Automation

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

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

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

ABSTRACT

Software quality is a major concern in the development of modern software systems. Software testing is the process of putting the developed system under testing to ensure its high quality. 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 especially in the large systems as it requires a lot of effort in terms of time spent in testing process. So, there was always a need to decrease the testing time through automating tests. This paved the way to "Automated Software Testing". Using automation, the high testing effort can be dramatically reduced and the overall costs related with it can be decreased as well. This leads to a more need to invent new efficient automated scripting techniques to ensure high quality systems. This study aims to provide a new scripting technique that facilitates the process of automating the execution phase through software testing in an industrial context.

Keywords

Software Testing, Automated Software Testing, Test Data, Test Case, Test Script, Manual Testing, Software Under Test, Graphical User Interface.

1. INTRODUCTION

Software testing has evolved since 1970's as an integral part of software development process, because 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 is generally described as a group of procedures carried out to evaluate some aspect of a piece of software [1]. Ehmer Khan shortly defines it as a set of activities conducted with the intent of finding errors in software [2] .

Since software testing process is a very expensive process, complete testing is practically impossible and it is also not acceptable to reduce testing effort by accepting quality reductions. Testing effort is often a major cost factor during software development. Many software orgjun nanizations are spending up to 40% of their resources on testing [3]. Therefore, an existing open problem is how to reduce testing effort without affecting the quality level of the final software.

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 [4]. 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 [5].

2. RELATED WORK

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 listed five different types of scripting techniques that will be discussed in this section [6].

2.1 Linear Scripting Technique

John Kent 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 [7]. Gerald Everett suggested that the linear scripts are 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. Figure 1 illustrates record/playback steps.

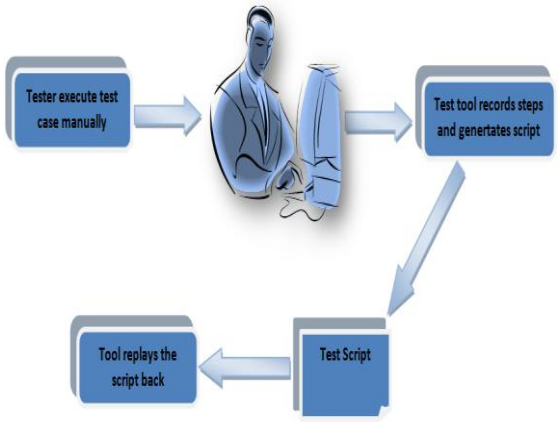

Figure 1: Record/Playback steps

2.2 Structured and Shared Scripting Techniques

Structured scripting technique uses structured programming instructions, which either be control structures or calling structures [6]. 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 back 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 is passed or failed according to these conditions. However, the script has now become a more complex program and the test data is still tightly coupled within the test script itself. Besides, implementing structured scripts require not only testing skills but also programming skills [6].

Shared scripting technique enables common actions to be stored in only one place. This implies that we require a scripting language that allows one script to be called by another one. 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 we do not have to spend time for implementing common actions many times across all scripts [8]. It works well for small-scale systems to be tested using relatively few test scripts. Figure 2 illustrates using shared scripting technique [8].

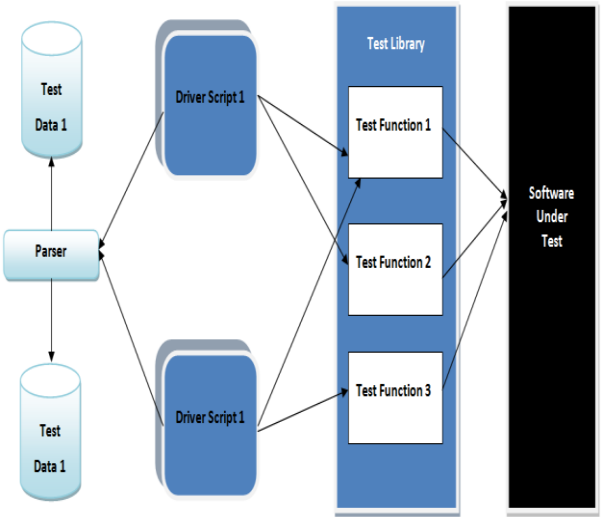

Figure 2: Driver scripts and a test library

2.3 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 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, we can store these values in an external excel sheet and implement test script to read test data to use it while executing the test script [9].

To automate new test case, we have to implement new control script and add new data records into existing data file, or create new one in the same format to be read by the control script. Figure 3 illustrates using data-driven scripting technique [8].

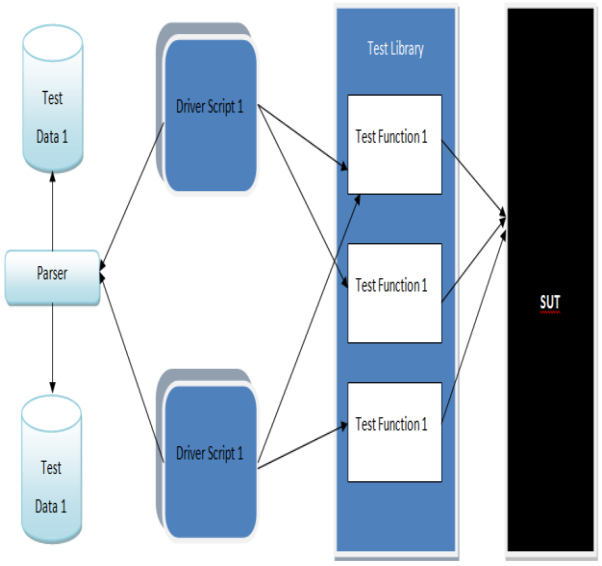

Figure 3: Data-driven approach

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 [9].

2.4 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 [8]. Fewster and Graham state that the keyword-driven scripting technique is a logical extension of the data-driven scripting technique [6]. 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. But keyword-driven technique takes out some of the intelligence from the script and put it into the external file with the test data and leave the task for reading both steps and data for the control script. Thus, instead of having data file in data-driven, we now have a complete test file. 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.

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. Also, a driver script which parses the test data and calls the appropriate keyword handlers is needed [8]. Figure 4 demonstrates these layers.

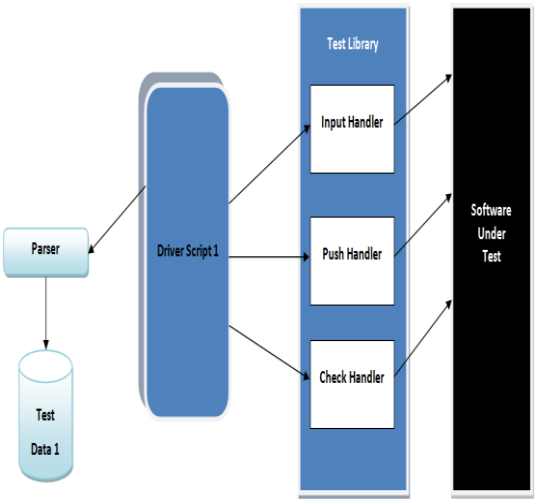

Figure 4: Handlers for keywords

3. PROPOSED SCRIPTING TECHNIQUE

Most of testing effort (measured in man-hours) is spent in execution specifically, followed by the development of test cases, then planning and analysis. The execution phase can be considered as the most important and critical phase of the software testing. Therefore, automation is better suited in clerical activities (e.g. test case execution and comparison activities) which are repeated many times than intellectual activities (e.g. creating test case or generating test inputs) which needs a human brain more than a machine. Figure 5 demonstrates both clerical and intellectual activities.

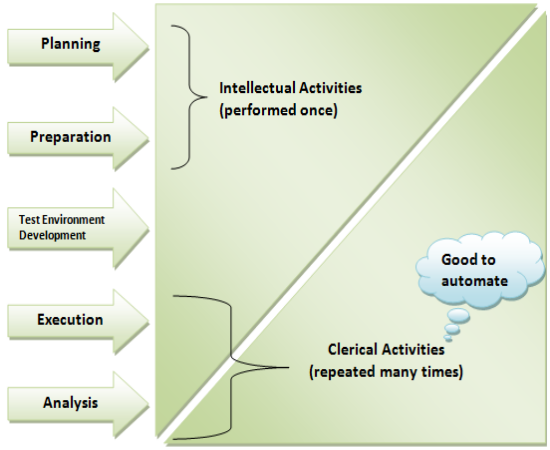

Figure 5: Software Testing Activities

The proposed solution can save the effort spent in the execution phase by reducing the amount of manual work involved in creating test scripts. For example, consider a situation when a combo box is replaced with a text box in the next release of SUT (or any other modification in SUT). Statements that select different values from this combo box will not work now when executed on a textbox. This simple modification may invalidate many statements in test scripts that reference this GUI object. The proposed solution can help in solving this problem by automatically generating code that gets HTML web controls and fills in them with sample data.

To implement the proposed solution, we will use:

- WatiN framework for .NET languages (version 2.1)
- Microsoft Visual Studio
- External data file (Microsoft office excel)

WatiN library is a testing framework for .NET languages to get HTML web controls. It is a testing framework that enables web application testing through any web browser. It also lets you open many web browser instances and interact with the elements of SUT [10]. It facilitates automated testing of web applications through browser interaction. It can, for example, fill in all input controls of a web page, and test for output. It is an open-source library for automating web browsers using .NET language. There is a lot of work behind it, but it is not our area of interest.

Generally, creating a test script process usually consists of two main parts:

- The first part is filling HTML web controls (e.g. text box, combo box…etc.) with sample data and firing events *(Clerical activity)*. This part is good to automate.
- The second part is building the script logic (e.g. if, switch case, for loop…etc.). The tester must implement this logic, as the machine cannot implement it *(Intellectual activity)*.

The proposed technique reduces part of the scripting effort spent through generating the first systematic part of the test script automatically instead of creating whole the script manually from scratch. The tester now has only the task of building up the logic of the script. The below table demonstrates who is the responsible about each part of the test script before and after using the proposed technique.

Method to implement test script	**The first part** *(Clerical part)*	**The second part** *(Intellectual part)*
Using data-driven scripting technique	Human tester	Human tester
Using proposed scripting technique	Computer (using the proposed technique)	Human tester

Table 1: The responsibility of creating each part of test script

By using any of the scripting techniques discussed in the previous chapter (linear, data-driven…etc.), the tester has to implement whole the test script manually from scratch, but using the proposed technique, the tester concentrates only on the intellectual part of the test script (e.g. dividing code into set of methods, parameterizing variables, creating settings…etc.).

The proposed technique is a hybrid scripting technique which consists of automating the first part of the test scripts and then applying data-driven scripting technique to avoid high maintenance costs on the long run as in figure 6. The main idea behind the proposed technique is to iterate on all HTML web and generates test script which fills in these controls with sample data values.

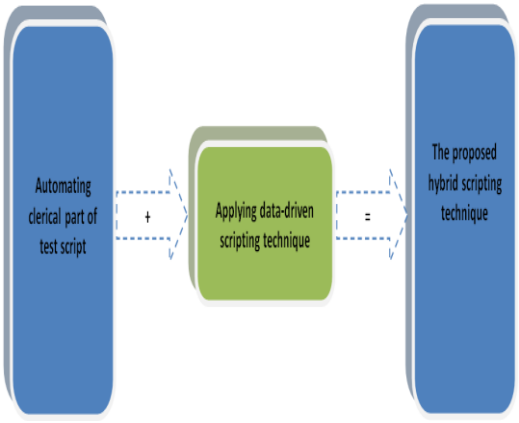

Figure 6: Proposed scripting technique

Following is the steps for creating the clerical part of the test script:

1. Set the page URL of SUT with the parent control type and name.
2. Create empty external excel data sheet (*.xlsx) to save data values.
3. Create empty class file (*.CS) to save the test script.
4. Navigate to the URL of the SUT.
5. Get source HTML of the web page.
6. Create HtmlDocument and StringBuilder objects.
7. Select the parent HTML node only (inserted above in step #1) and set it into the HtmlDocument object.
8. Get all children of this parent node.
9. Store the children nodes in a new list of HTML nodes.
10. Iterate on each single child node.
    a. Get both child Node Id and Type.
    b. Switch on node type (e.g. text, radio button, checkbox...etc).
    c. Generate the corresponding statement in WatiN format.
    d. Append this generated statement to the main string builder object (defined above in step #6).
    e. Insert new empty line after the added statement.
    f. Insert data values used in this line of code as a new row in the external data file (created above in step # 2).
11. Store the generated test script into the new class file (created above in step # 3).
12. Print the generated result script on the output screen of the tool.
13. Save the excel sheet file that contains all data used in the test script (created above in step # 2).
14. Print user-friendly message informing user whether test script generated successfully or not.

After applying the proposed technique that generates the clerical part of the test script, the tester must implement the second part of the test script (intellectual part). The whole test script (both activities) will be used in automating the execution phase. The below stepsdemonstrates the process of implementing the second part of the test script.

1. Choose specific test case to automate.
2. Run "Automation Helper Tool".
3. Set the page URL of SUT with the parent control type and name into the appropriate textboxes.
4. Click "Generate Script" button.
5. The test script is generated and stored in an external file *(clerical activity)*.
6. The sample data is generated and stored in an external data file.
7. Implement the logic for the complete test script manually *(intellectual activity)*.

3.1 Input and Output of the Tool

The input for the proposed tool:

- Target URL of system to be tested.
- HTML parent control type (e.g. div, span, form…etc.).
- HTML parent control identifier that contains all child controls.

The output of the proposed tool:

- Class file that contains the generated test script (clerical activity).
- Data file that contains all data values used in the test script. Thus, test data is stored in a separate data file instead of being hard-coded to the test script itself.

While performing tests, test data are being read from the external data. Applying the proposed scripting technique through running the developed tool can be considered as a preprocessing step before start implementing the whole test script. The tester can copy and paste the clerical part of the test script generated from the tool into the appropriate place in the test method. The below image demonstrates both input and output of the proposed tool while running.

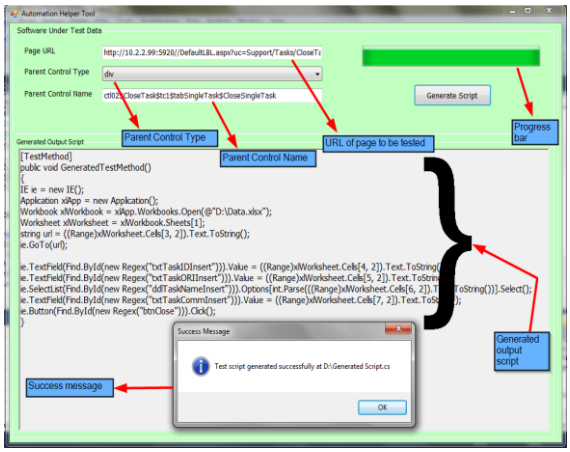

Figure 7: Snapshot of the proposed tool after generating script

3.2 Example

This section shows a practical example for applying the proposed scripting technique on a real project. The example system is LinkdotNet tracer project. It is a web application that is used to serve LinkdotNet employees to serve customers with internet subscriptions.

Since "Add New Subscription" module in the project will be used many times later, so it is a good idea to automate it. Adding new subscription functionality for new customers contains two main steps. The first step is to create new profile for the new customer; the second one is adding products for this customer.

We applied both the following two methods to implement the same test script:

- Implement whole the test script manually from scratch (before using the proposed tool).
- Using the proposed tool to generate the first clerical part of the test script automatically, then the tester implements the intellectual part manually.

Effort is being measured using data-driven scripting technique and using the proposed technique in terms of time needed to implement the test script. For example, creating the test script using the proposed technique consumes X time and creating the same script manually consumes Y time. Time X and Y are being compared later to measure the saved time.

4. RESULTS AND DISCUSSION

The tester tests the mentioned functionality 10 times manually and measure the time consumed. Then, the tester applies automated software testing 10 times also using both data-driven and proposed scripting techniques and measure the time consumed also. These results of the time spent to test the same test case manually and automatically are shown in Table 4, Figure 8 and Figure 9. All the below numbers are in minutes.

Test Case	Duration of manual testing	Duration of automated testing			
		Using data-driven		Using proposed technique	
		Time to implement	Time to run test	Time to implement	Time to run test
Test case 1	40	25	10	15	10
Test case 2	300	150	20	85	20
Test case 3	350	200	40	115	40
Total	**690**	**375**	**70**	**215**	**70**

Table 2: Time needed to test add new subscriber functionality

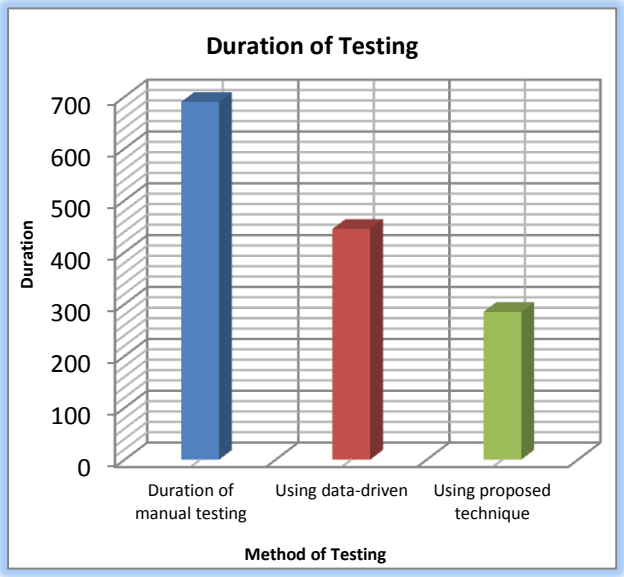

Figure 8: Time needed to test add new subscriber functionality

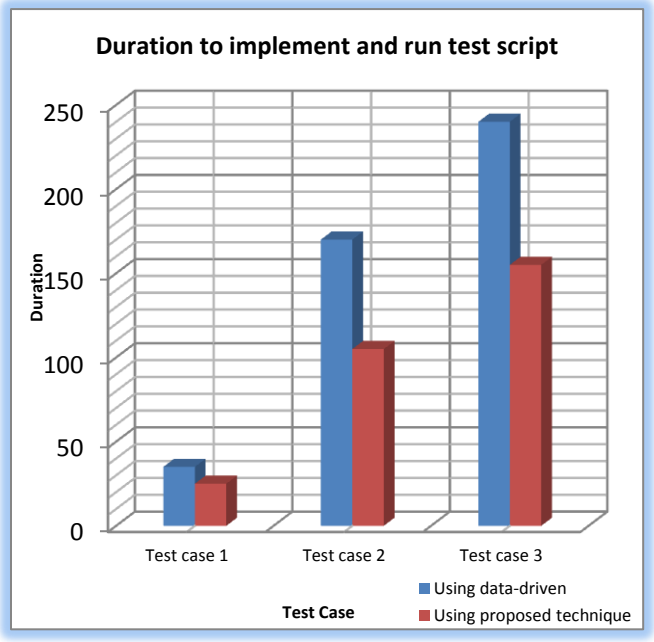

Figure 9: Time needed to test each test case separately automatically

According to the above table, it is obvious that automated software testing is better than manual testing. Also, applying automation using the proposed scripting technique is better than using data-driven scripting technique. The total time consumed during manual testing is 690 minutes (100%). The total time consumed of applying automation using data-driven scripting technique is approximately 445 minutes (64%) and that of using the proposed scripting technique is 285 minutes (41%). The most important point is that creating the test script using data-driven scripting technique consumed about 375 minutes (54%) to be implemented (for intermediate skill tester) and 215 minutes (31%) to be implemented with the help of the proposed tool. Thus, using the proposed scripting technique can save approximately about 42% of the total effort involved in scripting process.

The total number of LOC of applying automation using both data-driven and proposed scripting techniques is being compared in Table 5 and Figure 10.

Method to implement test script	LOC implemented by the tester	LOC implemented automatically
Using data-driven scripting technique	210	0
Using proposed scripting technique	145 (69%)	65 (31%)

Table 3: LOC implemented by the tester and the proposed tool

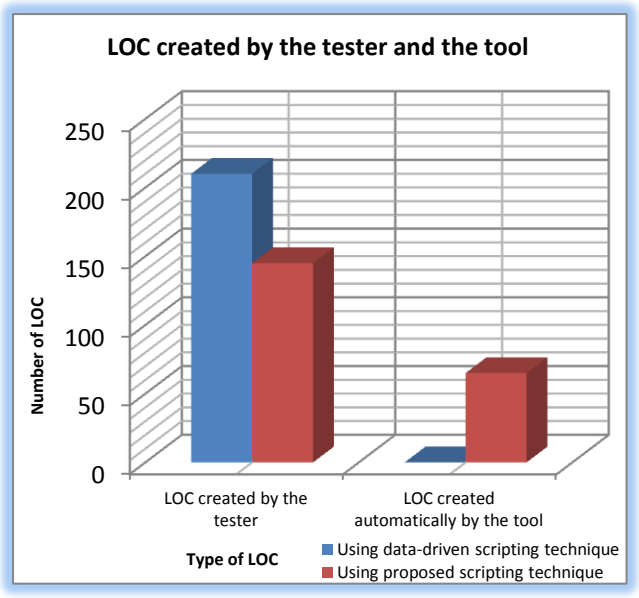

Figure 10: LOC implemented by the tester and the proposed tool

Table 3 demonstrates that using data-driven scripting technique; the tester will need to implement all the test script (210 LOC - 100%). But using the proposed scripting technique, the tester will need to implement the intellectual part of the test script (145 LOC - 69%) and the proposed tool generates the clerical part automatically (65 LOC - 31%). Figure 11 illustrates the alternative methods that can be used in implementing test scripts.

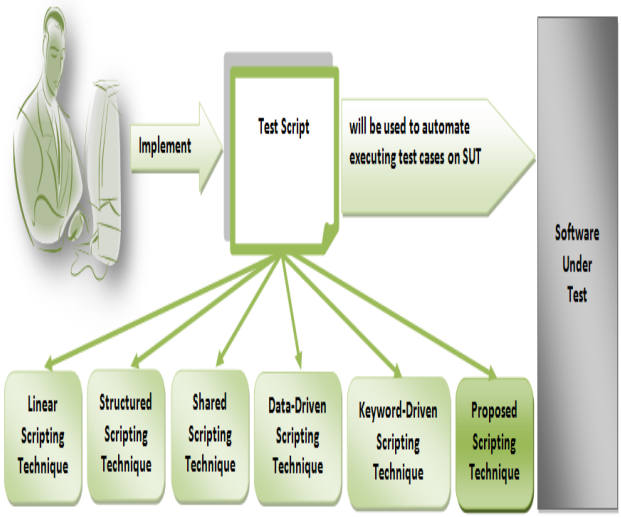

Figure 11: Different methods to implement test script

5. RELATIONSHIP BETWEEN PROPOSED TOOL AND NATURE OF SUT

The amount of saved effort using the proposed tool is dependent on the nature of SUT or even on the nature of each page alone. It is expected that the saved effort will increase in pages that contain large number of input controls. The more input controls found in the SUT, the more saved effort from the proposed tool.

6. 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, we have only two options, 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.

7. 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] 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.

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

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

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

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

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

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

[10] J. Menen, E. Wilde and J. Brown, "Web Application Testing In .Net," July 2013. [Online]. Available: http://watin.org/.