**Journal of Theoretical and Applied Information Technology**
30th September 2012. Vol. 43 No.2

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

# SOFTWARE TEST AUTOMATION - THE GROUND REALITIES REALIZED

**PRAKASH.V, SENTHIL ANAND .N BHAVANI.R**

Assistant Professor, Department of Computer Applications, SASTRA University
Assistant Professor, Department of Computer Applications, SASTRA University
Assistant Professor, Department of Computer Science, PRIST University

E- mail: prakash125@gmail.com, nsanand73@gmail.com, bhavprax@gmail.com

**ABSTRACT**

Test automation has always been looked upon as a magic formula to improve the quality processes of products/applications right from the day when first commercial product/application was made. But when one actually starts automating the testing, the ground realities are realized. Defining the scope of automation and selection of right tool for automation are over-whelming in the first place. And even if these teething troubles are overcome, the automation tool developed is usually inefficient as lots of important considerations are over-looked.
This paper aims at the following it suggest the solution of the above mentioned issues, It suggests best practices to be followed while doing the automation. It also aims at organizations who consider automating their testing process. It analyses the key factors to be considered in selecting a tool for automation.

**Keywords:** *Testing, Software testing, Test Automation*

## 1. INTRODUCTION

For achieving better quality products and a long-lasting solution for reduced costs Automation is the only savior. But these aims are achieved only when certain best practices are followed before and while developing the automation suite. Fig I shows the software testing life cycle

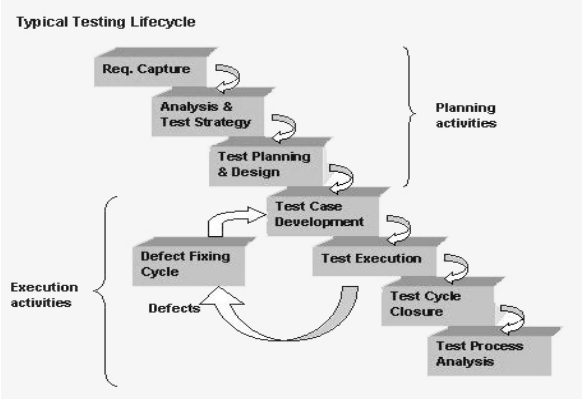


*Fig 1. The software Testing Life cycle*

Howard Fear has aptly stated, "Take care with test automation. When done well, it can bring many benefits. When not, it can be a very expensive exercise, resulting in frustration" [3].
More often than not, after automating the testing of a product, the automation team finds the automation tool more of a headache because of the unplanned and thoughtless approach adopted while developing the tool. Generally, lots of effort is spent in developing the tool, only to discover that the tool is limited in scope, lacks user-friendliness and requires frequent re-work every now and then.
And if sufficient care is exercised and proper practices are followed before and while automating the same product/application, the resulting automation tool not only saves time and effort, but is also a sheer beauty in itself because of the amount of user-friendliness, flexibility, reusability and extensibility it ensures. Let us, therefore, discuss what all needs to be taken care of before going for test automation and also while actually doing the automation[1].

## 2. AUTOMATION WHEN?

Fig 2 shows the various software automation activities. Lots of effort has to be spent even before you actually start with automation [4].

Journal of Theoretical and Applied Information Technology
30th September 2012. Vol. 43 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

It needs to be ensured that following things have been taken care of: -


*Fig 2. The software Test Automation*

### 2.1 Stability Of The Product/Application Is Ensured:

The first thing that needs to be ensured is that the product/application is fairly stable in terms of functionality. Even if it is slated to incorporate new features, the new features should not disturb the existing functionality. There is no sense in automation the testing of a product that is supposed to change functionality-wise.

Besides, the error messages generated by the product/application should remain consistent across different releases. If the testing is GUI-based, then it needs to be ascertained that the future releases of the product would not be undergoing GUI changes which might prove critical for the automation suite.

### 2.2 Interface To Be Tested Has Been Identified:

Three different interfaces a product might have are command line interfaces (CLIs), application-programming interfaces (APIs), and graphical user interfaces (GUIs). Some may have all three, but many will have only one or two. These are the interfaces that are available to you for your testing. By their nature, APIs and command line interfaces are easier to automate than GUIs. Find out if your product has either one; sometimes these are hidden or meant for internal use only. After this, you need to decide which interface's testing has to be automated.

Some relevant points are: -

GUI test automation is more difficult than test automation of the other two interfaces.

This is because firstly, GUI test automation will invariably require some manual script writing. Secondly, there will always be some amount of technical challenge of getting the tool to work with your product. Thirdly, GUI test automation involves keeping up with design changes made to a GUI. GUIs are notorious for being modified and redesigned throughout the development process.

- Despite the reasons for not depending on GUI test automation as the basis for testing your product functionality, the GUI still needs to be tested, of course, and you may choose to automate these tests. But you should have additional tests you can depend on to test core product functionality that will not break when the GUI is redesigned. These tests will need to work through a different interface: a command line or API.
- In order to simplify the testing of an API, you may want to bind it to an interpreter, such as TCL or Perl or even Python. This enables interactive testing and should also speed up the development cycle for your automated tests. Working with API's may also allow you to automate unit tests for individual product components.

### 2.3 Scope Of Automation Has Been Defined:

Before setting out to automate the testing of your application/product, it is essential to define the scope/intended coverage of the automation tool.

The scope may encompass functionality testing, regression testing or simply acceptance testing.

You can even select to automate the testing of certain particular features or certain selective test cases of different features [4].

### 2.4 Individual Test Cases To Be Automated Have Been Identified:

Automation suite should be looked upon as a baseline test suite to be used in conjunction with manual testing, rather than as a replacement for it. It should aim at reducing the manual testing effort gradually, but not doing away with manual testing altogether. But the fact is automation can assist testers but it cannot replace the manual testing effort. What machines are good at and humans are slow at should be chosen for automation[3]. Setting realistic and achievable

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

goals in early stages of test automation is a crucial factor for achieving success at long-terms. So, even after defining the scope of the automation tool in terms of acceptance/regression testing, etc, it needs be made sure that following kinds of test cases are eliminated from the scope of automation: -

- Test cases that are long and complicated and require manual inspection/intervention in between.
- Test cases that take tremendous amount of time in automation and it is difficult to ensure re-usability even if they are automated.
- Test cases pertaining to usability testing. Usability testing means testing in a true end-user environment in order to check whether the system is able to operate properly in accordance with the exact set of processes and steps applied by the end-user, including user's interface and system convenience estimation[2]. It's very important to include the right test cases in the suite. If the selection of test cases for the automation suite is not meticulous, you might end up discovering nothing really important about the software you are testing even if you develop a highly robust and reliable test suite.

**2.5 Test Cases Have Been Fine-Tuned:**

The test cases need to be fine-tuned for automation. The expectation level from the test cases for automating is widely different from the expectation from manual testing point-of-view.

The salient features that need to be taken care of include: -

Manual regression tests are usually documented so that each test picks up after the preceding test, taking advantage of any objects or records that may already have been created. Manual testers can usually figure out what is going on. A common mistake is to use the same approach with automated tests. But because of this approach, a failure in one test will topple successive tests. Moreover, these tests also cannot be run individually. This makes it difficult to use the automated test to help analyze legitimate failures. So, it is advised to revamp the test cases so as to make them independent. Each test case should setup its test environment[3].

The test cases need to be equipped with proper test-data. E.g. – If there is a test case for uploading of a file, then it should explicitly tell which file to upload. If there is a test case for creating a folder with invalid characters, then it should state which characters to use for creating the folder. Such fine-tuning of the test cases before starting automation ensures reduction in the actual time for developing the tool. It also guarantees that the tool actually executes the test cases in a way that checks the desired functionality.

**2.6 The Right Tool Has To Be Decided:**

There are hundreds of automation tools available in the market. A careful effort has to go into deciding which tool would be most suitable for automating the testing of your product/application. Following criteria would be useful in making the decision:

1. Is the automation suite required to work on different platforms? If platform independence is required, the demands on the automation suite will be very high. E.g. – If the suite has to support different flavors of Unix, it might be suitable to go for platform independent things like perl, etc.
2. If the testing to be automated is GUI-based, it might be preferable to use a tool like SilkTest, WinRunner, Rational Robot, etc. But every tool will have its own technical limitations that prevent efficient automation. So, it is necessary to monitor and evaluate the necessary testing tools for critical interfaces of the application that is under automation.
3. Sometimes, it might be best to develop a scripting tool using a suitable scripting language instead of going for the ready-made tools available in the market. This is especially preferable when the testing is on the server side.

**2.7 The Right Mode (Script Recording/Script Development) Has Been Decided:**

Most of the GUI automation tools have a feature called 'record and playback' or, 'capture replay'. Using this feature, you execute the test manually while the test tool sits in the background and remembers what you do. It then generates a script that you can run to re-execute the test. Script development, on the other hand, implies writing the scripts for running the test cases in the language used by the tool. Script development is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

akin to programming in a language like C or C++, but the purpose is to execute the test cases in an automated style. If you are going for GUI test automation, then points worth-considering while making a sane
decisions are:

1. Record and Playback approach which is commonly available in all automation testing tools for the creation of test scripts and test suites are easy to develop but difficult to maintain.
2. Error recovery cannot be incorporated by just recording a test script.
3. In data driven tests, achieving reusability of test scripts will be very limited.
4. Creation of test data and integration of them with test scripts is the most time consuming part. When a function is coded for the same purpose with data input file, maximum re-usability and ease is ensured.

More often than not, it will be required to strike a careful balance between the two modes, instead of using one of the two modes. Using the recording mode alone will render the automation suite non-reusable and using the scripting mode alone will require more investment of effort and time.
Though a middle path will be suggested generally, it might be worthwhile spending some time to decide the right mode or right mix of modes as per the application/product under consideration.
Most of the further discussion will be useful only when the right mix is adopted or scripting is followed altogether.
All in all, the suggested steps to be followed before starting with automation can be depicted in the figure below: -

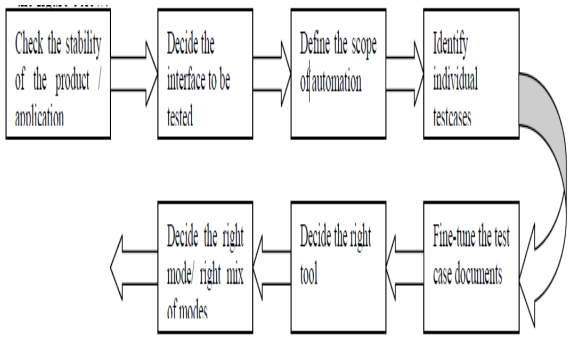


*Fig 3. The pre automation Stage Cycle*

## 3. AUTOMATION HOW?

After taking care of the above stipulations, the right direction has been identified and now the stage is all set to go for automation full-fledged. But in order to reach the destination, a lot more attention needs to be paid to. So, here we go: -

### 3.1 Following Proper Test Scripting Standards:

Automated testing involves a mini-development cycle. So, proper coding standards for test scripts should be prepared. Checklists should be developed for review of test scripts. On the whole, all the software practices followed in the development of an application or a product, which is applicable to the development of the automation suite, should be put in place. Whatever tool is chosen, ultimately, a tool will be only as good as the process being used to implement the tool.

### 3.2 Identifying Common Steps And Converting Them Into Functions:

At the outset, the steps common amongst different test cases should be identified and converted in the form of functions. Such functions can be placed in a common file, from where they can be called by different test cases by passing suitable parameters as per the need. This will encourages re-usability of code and save effort and time. Besides, these functions can be used again when newer test cases are added to the automation suite at a later stage.

### 3.3 Identifying Other Peripheral Functions:

After the functions as stated above have been identified, it is advisable to identify the peripheral functions that will be required by all the test cases in general. E.g. – A separate function for writing into log files can be written. This function can take the error message, severity level of the error message and the path and name of the log file as the input parameters. Depending on the requirements, more of such reusable functions can be identified. Such functions will simplify and streamline the process of test script development in the long run.

### 3.4 Providing Room For Extensibility:

The automation suite should be written in a manner such that additional test cases can be added to it. The additional test cases may cater to testing enhanced functionality of an existing feature as well as testing new features incorporated

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in the application/product. The suite should have such architecture that it is extensible both in terms of being able to add more functions and also in terms of being able to add more test cases by calling the existing/new functions.

**3.5 Generating Proper Logs:**

A common problem is what to do when automated tests fail. Failure analysis is often difficult. Was this a false alarm or not? Did the test fail because of a flaw in the test suite, a mistake in setting up for the tests, or an actual defect in the product? Hence, it is required that the suite should generates logs of its own. But a good automation suite with ambiguous logging messages is worse than manual testing. Few points that need to be taken care of from logging point-of-view are: -

An ideal automation suite should explicitly check for common setup mistakes before running tests and generating detailed logs of its own. And logging needs to be as user friendly as possible [2].

The logging should be done in a manner that facilitates statistical analysis of the results.

This implies that the log file should have the results in such a format such that can be processed by parsing, and useful statistics can be generated.

**3.6 Independence Of Selective Execution:**

The scripts should be written/arranged in such a manner that they provide the independence of executing individual test cases or at least test cases belonging to the same module. This is important when the need is not to execute the entire suite, but to verify particular bugs.

**3.7 Signal-Handling And Clean Exit On Abrupt Termination:**

It needs to be ensured that the suite does all the clean up when terminated abruptly, consciously or unconsciously, by the user. It may be required by the script to handle the termination/kill signal for a while so as to get the time for cleanup (and may be, complete the currently executing test case, if the suite desires). Such signal handling is extremely important in some particular cases. E.g. - When an automation suite is run through command line on a Unix terminal as a foreground process and the user does a Ctrl-D in order to stop the suite for whatever reasons. The suite might have changed some configuration files or properties files before it received the signal. So, if the changes are not reverted back before the termination of the suite, then things will go for a toss.

**3.8 Self-Sufficiency In Individual Test Cases:**

Test cases should not be dependent on preceding test cases for execution. If there is dependency on test cases occurring before in the sequence, then the subsequent test cases will fail without any reason. If at all such dependence is unavoidable, the error message in the log file, when such test cases fail because of the failing of preceding test case, should be explanatory enough.

**3.9 Equipped With Test Data:**

The automation suite should be equipped with all the test data required by the different test cases. The test data may consist of simple data input as required by the test cases to supply parameters to the functions for testing different conditions like numeric input, alpha-numeric input, non-alpha-numeric input, etc. It may as well consist of specific files to be supplied to the test cases to test particular functionality of the application/product. The automation suite has to be accompanied with such test data and this test data has to be prepared for the suite with precision. Example: - A particular feature of the application/product may have to be tested with files of different sizes, say 0 bytes, 64 KB, 1MB, 30 MB, etc. So, the suite requires having the files precisely of these sizes only. All such files may be kept in a particular folder from where the suite picks them up. The regular input data, which is required by the functions as parameters can be supplied through the input data files. The individual test cases may parse the parameters to be supplied to the test cases while reading them from the input data files. The tools available in the market support different types of file to be used as data input files. Example: - Win Runner uses excel sheets for reading data, while SilkTest uses .dat files [4].

**3.10 Dynamic Generation Of Names For Temporary Files And Input Data:**

Sometimes, the automation suite would require creating certain temporary files. If the suite does not delete the temporary files created by itself, then they will get over-written in the next run of the suite. Besides, if a file by the same name exists even before the first run of the suite, then that file may get over-written in the first run itself. The consequence will be even worse if the write permission is not there on the already existing file. The script will fail to over-write also in such a case and the test case might eventually bombard. Similar problems are faced when the suite contains positive test cases like creating a folder with a

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

given name. If the suite does delete this folder created as a part of the clean-up process, the test cases fails unnecessarily when the suite is run again with the test case trying to create a folder with the same name [4].

A solution to all such problems is to dynamically generate the names for temporary files and all such input data at the run time. This way the names will not conflict with those of the existing files and fresh data. Such dynamic generation of names can be accomplished by several ways. One typical way of generation can be stripping the microsecond part of the current time and appending it to a name. This way there will be an extremely rare probability (10 to the power of –6, to be precise) that a conflict in the names will take place.

**3.11 Cleaning-Up**:

It has to be ensured that the automation suite brings the application/product back to the original state it was in before the suite executed. If any configuration or properties file was changed for the execution of some test case, then the changes must be reverted back. If the suite generates some temporary files, they should be deleted by the suite towards the end.

**3.12 Incorporating User-Friendliness:**

The automation suite should be as user-friendly as possible. Some basic points for ensuring user-friendliness are: -

1. The user should have freedom to put the test data files anywhere on the m/c.
2. The suite can be run from anywhere on the m/c.
3. It can be installed anywhere on the m/c.
4. Once it is run, the suite should not require any manual intervention till completion. The

user should be able to run the suite unattended. For incorporating such user-friendliness, the suite needs to be designed in a proper way. A separate configuration file can be created that contains all the variables that the user might want to change. E.g. – The user might want the log files to be generated on the desktop instead of a hard-coded path. The user might as well want the suite to pick-up the test data/files from a directory of his choice. All such entities can be placed in the configuration file in the form of variables that the user can change easily. The suite can read these variables from the configuration files every time it is run. If such a design is used, all that the user would need to do before running the script is to change the configuration file as per his needs. Thus, the user will get a tremendous amount of flexibility and independence.

**3.13 Developing An Efficient Error Recovery Routine:**

Error Recovery routine enables the test suite to run continuously without any intervention. The function of this is to anticipate errors, make a decision on the corrective action, record the error and proceed further with next test, if possible. E.g. – If an unexpected termination of application under test happens, the routine should be able to realize the interruption and restart the application. This prevents reporting wrong defects after a test suite execution. In brief, this will ensure that failures in test execution are effectively confined, interpreted and allows suite to run continuously without failures. Without such an error recovery system, automated test suite runs will never take off. Manual presence will become a necessity during test suite execution.

## 4. TEST SCRIPTS FOR TEST DATA SETUP AND CLEANING UP:

If the automation suite does not take care of test data setup, it will have to be done manually by the user, which reduces the fun of test automation. This becomes all the more important when test data setup requirements are huge and as a result, the whole exercise become highly time consuming. Hence, an ideal automation suite should incorporate dedicated scripts for test data set-up. These scripts are then being executed before any other functionality test can be done on the product. E.g. – When the application/product in focus is an ERP suite or banking software, the test data setup part itself may take 3-4 man-days of effort. With the automation in place for this setup, the effort is reduced drastically. Similarly, scripts for cleaning-up should also be incorporated in the automation suite [4].

Such scripts will aim at bringing the application to the ground state it was in before the automation suite was run, i.e., they will undo all the changes that any test cases in the suite brought about while executing. E.g. – If there is a test cases for creating a folder, then the clean-up action will delete this folder.

## 5. TESTING THE TEST SCRIPTS:

Test scripts should be tested before they are used in a test suite. Testing of all test scripts

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

should be made in test automation activity. Adequate tests need to be accomplished for each test script. When test error simulation and rectification is difficult and time consuming process, reporting false errors can cost more and defeat the objective. The goal for the automation team has to be that a test program should never report a false problem [5]. All scripts should satisfy the following criteria: -

1. When given a valid input, they produce the correct output.
2. When given an invalid input, they correctly and gracefully reject the input.
3. Do not hang or crash, given either valid or invalid input.

## 6. CONCLUSION:

Test automation is a great idea. But it is not a magic wand. Proper time and effort has to be spent for the development of the test automation suite. And the key is to follow the right processes. In eagerness to achieve fast results, the desirable processes are compromised. And that is the reason why, more often than not, it only promises and raises hopes, but simply fails to deliver.

## REFERENCES:

[1]. Bach, James. 1996. "Test Automation Snake Oil." *Windows Technical Journal*,(October). http://www.satisfice.com/articles/test_automation_snake_oil.pdf.

[2]. *Success with Test Automation* by Bret Pettichord (bret_pettichord@bmc.com)

[3]. *Howard Fear on Test Automation* by Howard Fear (hsf@pageplau.com)

[4]. *Automated Testing: A Practical Approach for ERP product* by Kishore C.S.(cs@rsi.ramco.com).

[5]. Yongxiang Hu , "*The Application and Research of Software Testing on Agile Software Development*" , IEEE International Conference on E-Business and E-Government (ICEE), 2010.