Software Testing: When, How and How-Much to Automate

[1]Priyanka Rani, [2]Devendra Nagal
[1]Computer science department, [2] Assistant Professor, Electrical Engg
[1] Jodhpur national university, 2Jodhpur national university

***Abstract*-People in software industry have always considered Test Automation as a magic formula which helps in improving quality of the application/product. It has always been considered that automating the software tests is the best way to increase the effectiveness, efficiency and coverage of your software testing. But when one actually starts automating the test scenarios, he/she starts realizing the ground realties. The general problems of actually deciding the right time of starting test automation (When), deciding on the tools or frameworks (How) and finalizing on the actual scope of automation (How Much) starts arising. And when these factors has been considered, there will be multiple factors that are generally over-looked during these phase and leads to some in-efficient results or multiple iterations of When, How and How Much phases.This paper focuses to suggest some of the best practices that can be followed during Test Automation and will result in more accuracy in defining the When, How and How-Much phases for a Test Automation. Following these practices will ensure maximum efficiency and minimal re-work iterations during any Test Automation cycle.**

***Index Terms*—STLC, Software Test Automation, Requirements impact on testing**

I. INTRODUCTION

We know that repetition of software tests is kind of mandatory during software development cycles for ensuring software quality. Each time a piece of software code changes; the tests need to be repeated for verification of that. Also, it needs to be repeated for all supported configurations of hardware and operating systems. Now in the agile environments as of today, where requirements and software's are changing on a very rapid pace, manually repeating of these test cases is very costly and time consuming, which arises the need of automated tests, which once created, can be repeated any number of times without any additional cost and even they are much faster compared to manual test cases. So if considered from a long term perspective, automation is the most suitable solution for delivering better quality products with a reduced code. But these aims are achieved only when certain best practices are followed before and while developing the test framework/tool and test-suite.*Howard Fear has mentioned that "Take care with test automation. When done well, it can bring many benefits. When not, it can be very expensive exercise, resulting in frustration". [3]*Generally, it has been observed that automating test cases of the application under test (AUT); the automation team himself find that the tool or framework developed is not worth and more of a headache, due to unplanned and thoughtless approaches taken during development of the same. So a clear planning around what actually needs to be automated and how much part of that needs to be automated , should be a priority exercise, before working on the how part of the test automation cycle.So, if sufficient care is exercised and proper practices are followed while automating the AUT, it not only saves the time and effort, but also results in a sheer beauty because of the amount of user-friendliness, flexibility and extensibility it ensures.

In further topics, let's discuss which things/practices need to be taken care before starting the test automation and also while actually doing the test automation.

II. SOFTWARE TEST AUTOMATION: WHEN

Identifying the when part needs the most planning before starting anything on the test automation phase. Deciding on the time, when to kick-off for the test automation is one of the most crucial phase, as it only will ensure that it should not be so early that a lot of time/money is wasted in waiting for stuff which is not yet ready for testing and shouldn't be so late that it is no more used for delivery of the application under test. Following things needs to be ensured before kicking of automation for any application:-

A. REQUIREMENT/STUBS FINALIZATION OF THE AUT (APPLICATION UNDER TEST):

The first thing before kicking off automation for any application is to ensure that functional requirements for the AUT have been finalized and frozen. It needs to fully ensured that the Business Analyst, Developer and the QA teams are on same page, as per the functional requirements are concerned. It needs to be ensured that same UI mocks and API stubs have been provided to all stake-holders, to minimize all the requirement gaps going further. Once the UI mocks and API stubs are available, dev team can start their development and in parallel the QA team can also start with development of their test skeleton for the automation suite.

B. DECIDE AUT FUNCTIONAL STABILITY:

Next step for starting with actual test automation is to ensure that the AUT is functionally stable, before it is handed over to the QA team for test automation. Even if the application is expected to incorporate new features, they shouldn't have any direct impact on the existing delivered functionality. It doesn't make any sense in automating the features of the AUT, which are expected to change as per functionality.

For a GUI based application, it should be ensured that there should not be any uncommunicated UI architectural changes across releases that can break the automation suite. Similarly, for a backend API based application, it should be ensured that there should not be any uncommunicated changes in the API request signature and the response, which can directly break the validations in the automation.

C. PRIOR IDENTIFICATION OF INTERFACES TO BE TESTED:

An application generally possesses three types of interfaces to access the application:

- Graphical user Interfaces (GUI)
- Application Programming Interfaces (API)
- Command Line Interfaces (CLI)

It is very important to identify all these interfaces before starting any testing. Sometime APIs are specified for internal use only and not exposed to outside systems, so make sure to identify such scenarios in advance, as that will help in finalizing the scope of testing and automation.

For each of the interface type, consider following points:-

- Generally UI changes are very frequent while development of application, so for initial automation workflows needs to be identified (preferably using APIs and CLIs), which will cover all happy flows of the functionality of the AUT and will have very minimal impact on the functionality when UI of the application changes. This way initial part of the automation will be more robust and will not break with continuous changes and development of the application.

D. AUTOMATION SCOPE AND TEST CASE FINALIZATION:

Before initiating any sort of automation, it is very important to identify the scope of automation and various test cases that will be covered under each of the scopes identified.

Scope will cover various levels of testing like functional testing, regression testing or may be just acceptance testing. So, a proper call must be taken that for which particular phase, you are kicking off the automation. There can be multiple factors that can impact this decision, like if time to delivery is a major factor, and then it's recommended to first cover positive flows in automation, so as it covers bare minimum acceptance scenarios and will provide quick feedback for early failures. Then it can be further extended for functional and regression tests.

While defining the scope, it is also very important that test cases have been identified for all types of test suites. With this, it should always be in mind that automation is never a replacement for manual verification; it is just an aid-on for the manual testing. So before kicking of automation, make sure that at least following type of test cases (which will make more sense in manual execution) has been identified :-

- Test cases that require manual intervention and inspection.
- Test cases which takes very long time in execution and not much of repeated nature.
- Test cases from usability and end user perspective.

Thus it is very important to identify right set of test cases that should be included in the automation suite, otherwise you will end up in discovering no issues, and no matter how robust is the test suite.

E. FINE-TUNING OF TEST CASES:

Test cases need to be more fine-tuned before considering them for automation. The expectation level from the test cases for automation is widely different for the expectations from test cases for manual execution. At least following features needs to be taken care before including any of the test cases in the automation suite :-

- Manual test cases when executed in a sequence take benefits of any objects of records that are being created as part of prior test cases, as it is very easy to figure out manually what sequence is being followed for execution of test cases. But a major mistake happens when the same approach is being taken for the automated test cases as well. It will turn out to be a disaster, as failure of one test case will cascade to failure of all other following test cases. Also, it will have a limitation that these test cases can't run in isolation. So, it is always advisable to design and select the test cases for automation in such a fashion that all of them can run independently.
- Each test case should have its own proper test data. E.g., if any sort of input file or precondition needs to be satisfied for the test case, then they should be available as a pre-step before execution of each of the test case.

Such fine tuning of the test cases before starting the automation makes sure that the desired functionality is validated by the automation suite and ensures that overall time spent for development of the test suite and test tool are very less comparatively.

F. AUTOMATION TOOL AND FRAMEWORK FINALIZATION:

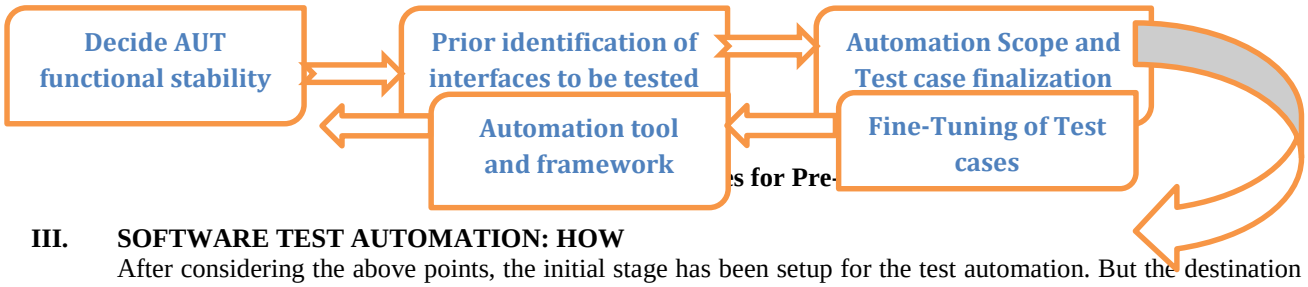
There are thousands of automation tools and frameworks available in the market, but not all are suitable for each and every application under test. So a careful effort has to go in deciding the tool which would be most suitable for automating the testing of your application. Following are few of the criteria that can help in making the decision:-

- Will the test suite execute on multiple platforms? If yes, then platform independence is the basic ask from the automation tool. So, will be better to go for Java, Perl or Python based tool or framework.
- Will it be used to automate UI interfaces? Are the UI interfaces web based or desktop based? E.g., for web-based applications selenium-web driver is the most suitable tool whereas for other desktop applications tools needs to identify that will be suitable for the specified platform.
- Will need customization? If none of the tools available in market are directly suitable for your test automation, then a customized framework can be developed either using any scripting language or developing a wrapper over any of the existing tools.

- Record/Playback vs Script Mode? Most of the UI automation tools have a feature of record/playback. Using this feature, you execute the test manually, while the tool in background generates a script for all the steps performed. On the other hand, there are tools which allow writing the test script directly in a programming language of your choice.

  More often than not, it will be required to strike a careful balance between the two modes, instead of using one of the modes. Using the recording mode alone will render the automation suite non-reusable and using the script mode alone will require more investment of efforts and time. So it is worth spending some time to decide the right mode or right mix of modes as per the application under test.

Overall, all steps needed before starting automation of the application can be depicted as below:-



s for Pre-

III. SOFTWARE TEST AUTOMATION: HOW

After considering the above points, the initial stage has been setup for the test automation. But the destination is still far away. For starting with the actual development of the automation, following points needs to be considered:-

A. TEST SCRIPT AND CODING STANDARDS:

Automation framework should define proper coding and scripting standards. This allows reusable code to be incorporated throughout any given test. It can also assist in set-up tasks, such as preparing an environment or creating or refreshing database tables. The framework should define standards which would be helpful in making the test scripts consistent, readable and easily maintainable.

In actual, test framework development is kind of a mini development cycle in itself, so all coding standards followed while developing the application under test, also fits best for the framework development cycle also. A proper checklist should be developed for review of all the test scripts.

B. SEPARATION OF UTILITY FUNCTIONS:

All methods which provide common functionality that can be used all the test scripts should be put in a common file, which can be invoked by all the test scripts by passing parameters as per their needs. This will encourage re-usability of code and will save effort and time. E.g., Logger functions can be put in a separate class/file. These functions can take error message, severity level of error messages and log file path as parameters for each different test script. Such methods will simplify and streamline the overall process of test script development in long run.

C. DATA INPUTS FOR TEST SCRIPTS AND AUTOMATION FRAMEWORK:

One of the most important points regarding test automation framework is how would you test large volumes of data quickly on the system quickly, and more importantly how would you get that data. You may have specific scenarios that require an object in a specific state. As a good measure of verification, you can use Microsoft Excel spreadsheets (data-driven) or database queries (keyword driven) to gather information to be tested. You can also leverage this to include database verification after a task has been completed. You must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort.

So, a proper support should be provided in the framework to read any volume of data from various datasets as per need of the test scripts.

D. DOCUMENTATION AND REPORTING:

A test framework should define what should be a correct and well defined test result and report format, which should be generated automatically after a test run and show the results in an easy-to-read format. The reports should provide specifics about where application failures occurred and what test data was used. Proper logging of each test step and providing screenshots for execution of each step helps in easily identifying the root cause of failure during test execution. Reports should also be in a format that can be easily shared across the entire QA and development teams.

Apart from logs and reports, a proper documentation should be generated for all functionality the test framework, so that the users can easily get out, how to use the framework and which all functionality is provided by the framework that can be used by them for development of their test scripts.

IV. TEST SETUP/CLEAN-UP AND DYNAMIC GENERATION OF TEST FILES/DATA:

The automation test suite is expected to take care of test data setup before imitating the test execution; otherwise it will need to be done manually, which reduces the fun of test automation. This becomes very important when test data setup is really huge and very time consuming. Hence an ideal test automation suite should include specific scripts which take care of all initial test data setup and are expected to be executed before executing any of the functional test cases.

Similarly, scripts for clean-up should be incorporated in the automation suite. Such scripts would aim at bringing the application to the ground state it was in before starting the automation suite, i.e., they will undo all the changes that any test case in the test suite has bring up.

Apart from the initial test data, the test scripts would also need to generate some data on the fly while execution. Sometime this data would also need to be saved into some temporary files, for sharing among multiple test steps. So the

framework is expected to provide the functionality to generate these files on the fly and would be better if file naming is also dynamic on basis of may be date and time, which will minimize any chances of overlapping of data.

Also, the clean-up activity is expected to clean-up all these intermediator files also, which were generated while execution of the test scripts.

A. ERROR RECOVERY ROUTINES:

Error recovery routines enable the test suite to run continuously unattended. The function of this routine is to anticipate errors, decide on corrective action, log the error and proceed further with next test. E.g., if the application under test terminates unexpected, the routine should recognize the termination and restart the application with clean-up. This prevents cascading effect or reporting wrong defects after a test suite execution. In nutshell, this will ensure that failures in test execution are effectively trapped, interpreted and suite continues to run without failures. Without such an error recovery system, automated test suite runs will never take off and manual presence will become a necessity during test suite execution.

B. SELECTIVE AND SINGLE EXECUTION:

The script should be written in such a manner that they provide the independence of executing individual test cases or at least test cases belonging to same module. This is important when the need is to execute selective test cases for verification of specified bugs during regression.

Also, test cases should be independent on execution of previous test cases in execution. If there will be 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 files should be explanatory enough.

C. TEST EXECUTION ENVIRONMENT:

This would include a continuous build system to launch a test suite at will, on schedule, or even after another event has occurred. With this environment, you can also run different suites of tests, such as smoke tests, functional tests, or browser-based tests.

V. MAINTENANCE AND EXTENSIBILITY:

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

The above mentioned points are the basic key guidelines that, if followed, will help in designing and developing a test automation framework which will be reusable and helpful in a long run.

VI. SOFTWARE TEST AUTOMATION: HOW-MUCH

After deciding on when to start automation and how to design and develop automation framework and automation test suite, the main point that needs to be thought of is how much to be automated. Considering following points for finalizing the exact scope of useful automation:-

A. ARE YOU THINKING OF 100% AUTOMATION?

Please stop dreaming. You cannot 100% automate all your testing work. Certainly you have areas like performance testing, regression testing, and load/stress testing where you can have chance of reaching near to 100% automation. There are areas like User interface, installation, compatibility and recovery where some part of manual verification is always needed.

So a prior thought should be given before deciding on the volume of test cases which needs to be automated. Time line to delivery and complexity will also be major factors in deciding on the number of test cases which needs to be automated. While designing the test cases, each test case should be assigned a priority and the while starting the automation, first focus should be to automate P0 and P1 test cases, so as to ensure all major functionality is covered before the release.

B. FREQUENCY OF TEST RUNS:

Identify application areas and test cases which will need to run at most only once and will not be included as part of regression. Avoid automating such test scenarios or modules of the application.

VII. LIFETIME AND SCOPE OF AUTOMATION SUITE:

Every automation suite should have enough life time that its building cost should be definitely less than that of manual execution cost. This is bit difficult to analyse the effective cost of each automation script suite. Approximately your automation suite should be used or run at least 15 to 20 times for separate builds to have good ROI.

Also the automation suite should take care that it includes at least minimal number of test cases from all major functionalities delivered by the application. So, in minimum all positive test cases should be included as part of automation test suite.

PHASE-WISE AUTOMATION:

Instead of automating the entire test suite, it is always advisable to opt for phase wise automation. Test cases can be picked up for automation in various phases depending on priority and scope of the test cases.

VIII. CONCLUSION:

Test automation is always a great idea, but it fails many a time due to eagerness of achieving fast results. So it's always advisable to spent proper time in planning the when, how and how much phases of automation. Also, there should be a proper mix of manual as well as automated testing for a successful delivery of the application.

REFERENCES:

[1] K. Adamopoulos, M. Harman, and R. M. Hierons. How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In GECCO (2), Volume 3103 of Lecture Notes In Computer Scienc, pages 1338–1349. Springer, 2004.

[2] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. Information and Software Technology, 51(6):957–976, June 2009.
[3] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions on Software Engineering, 36(6):742–762, Nov. 2010.
[4] N. Alshahwan and M. Harman. Automated web application testing using search based software engineering. In Proceedings of Automated Software Engineering, pages 3–12, 2011.
[5] D. Amalfitano, A. R. Fasolino, and P. Tramontana. A GUI crawling-based technique for Android mobile application testing. In Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, pages 252–261, 2011.
[6] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. Mcminn. An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8):1978–2001, Aug. 2013.
[7] S. Anand, M. Naik, M. J. Harrold, and H. Yang. Automated concolic testing of smartphone apps. In Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, pages 59:1–59:11, 2012.
[8] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In Proceedings of the International Conference on Software Engineering, pages 402–411, May 2005.
[9] Mit app inventor. http://appinventor.mit.edu/explore/.
[10] A. Arcuri and L. Briand. Adaptive random testing: An illusion of effectiveness? In Proceedings of the International Symposium on Software Testing and Analysis, pages 265–275, July 2011.