TAVANT

[Whitepaper]

10 Steps to Simplify Software Test Automation Process

Raghavendra Kulkarni

Abstract

Software quality has a direct impact on cost, service reputation, and revenue. The best way to monitor and manage software quality is to have a testing program in place. While manual testing might be the best option in some cases, in others it can be tedious and time consuming. Moreover, as the complexity of a software application increases, quality becomes more risk prone, and that requires organizations to decipher the optimal testing solution. Software test automation can be the best answer in such cases, as it increases the span and repeatability of testing, and even reduces redundant manual testing.

Automation testing is particularly suitable for projects in which Agile Sprint methodology is followed for developing an application under test.

Introduction

Software test automation is the use of special software test processes to control the execution of tests and the comparison of results with predicted outcomes. This method is useful where repetitive tests are required, but it is difficult to perform them manually. Automation testing is particularly suitable for projects in which Agile Sprint methodology is followed for developing an application under test. That approach can also be used in other automation projects where requirements change frequently and features are under development. To optimize the benefits of automation testing, we need to focus on a small set of requirements and incrementally build automation framework / tests in subsequent iterations. However, care should be taken while developing those framework enhancements / utilities. One should ensure to keep them as generic as possible for allowing wider support, or a larger range of options, during script development. That requires thorough understanding of the framework, automation scripts, and software libraries, which will be used to build the same.

Framework/test enhancements should be kept as generic as possible to allow wider support, or a larger range of options, during script development.

Challenges With Traditional Testing Processes

Manual testing involves:

- Writing the code
- Uploading it
- Building it
- Running the code manually, step by step

  Checking log files, database, external services, values of variable names, output on the screen, etc.

A primary challenge with manual testing:

If something were to go wrong, the whole test process would need to be repeated. That involves a gross overhead in terms of work and ROI.

An example:

- registerDomain
  - Case 1: Register a .com domain with all correct fields
  - Case 2: Register a .com domain with an invalid name server
- More often than not, manual testing ends up as integration tests. The challenge with integration testing is that if just one unit fails, the entire test process breaks down.
- Manual testing requires complex setup processes, i.e. run db queries, changes to backend servers, and the process becomes more complex with incremental work.
- Manual tests do not drive design
- Manual tests are run post-facto, and hence only drive bug-patching
- Manual tests do not provide a safety net
- Manual tests create code clutter
- Most manual tests involve –
  - System.out(s) to check values of variables
  - Useless log file entries in app server, db server, etc.
  - Causing code/log/console clutter
    - if then(s), flag based logging, event-based log entries, etc.

As mentioned earlier, following all these steps can slow down the application considerably.

Software test automation, besides saving time and effort, helps to design, develop, and deliver automated tests to increase efficiency and reduce errors. Since the automation can support all aspects within the lifecycle of testing, it can expedite and streamline testing efforts.

Software test automation can streamline efforts and reduce errors while supporting all aspects within the lifecycle of testing.

Criteria for automated testing to perform effectively:

- Run on multiple computers
- Run on different types of operating systems's, platforms, and programming languages
- It should be independent of network protocols
- It should, ideally, also support integration of multiple testing tools.

10 Steps to Simplify Software Test Automation Process

1 Requirements/Stories

A review of requirements/stories can provide insight into the overall automation strategy, i.e. evaluating tools and techniques. The next step will be to review available automation tools for the application/feature, and choose the one which best meets requirements.

2 Automation criteria

Establish standard automation criteria for requirements/stories. Those criteria should determine whether the requirements are ready for automation. It can help in avoiding issues arising from frequently changing requirements, or in the absence of detailed test cases.

3 Estimation

Review detailed test cases and determine automatable scenarios out of them:

- Identify automatable test cases
- Classify test cases based on complexity (high/medium/low)
- Estimate the effort required for each test case (the higher its complexity, the greater the effort required)
- Estimate effort for test data preparation, test script execution/testing

While automating web applications using Selenium API/library, programming language capabilities along with automation tool features can be leveraged to achieve automation.

4 Test script design

While designing automation scripts/libraries, it is necessary to understand the capabilities and limitations of tools used for automation. For example, while automating web applications using Selenium API/library, programming language capabilities along with automation tool features can be leveraged to achieve automation. Further, to automate Omniture (webpage metrics tracking) using a given browser, Selenium/Web Driver can be used to request a page and download network traffic in .har (http archive) format. We can then use Java libraries for parsing the .har file to get specific Omniture parameters and corresponding values for verification.

5 De-coupling test script and test data:

Separation of test data and object identifiers from test script/logic is a critical part of the design process. Such type of design provides for ease of maintenance of automation artefacts. Whenever there is a change in application GUI, only the files storing object identifiers need to be changed without modifying scripts. Similarly, change in application behavior can be incorporated in the automation framework/suite by changing the script logic.

Therefore, having separate test data can make testing easier for non-automation engineers, as there is no need to know the underlying details of automation scripts/logic to run tests with different test data sets.

Separated test data can make testing easier for non-automation engineers, as they will not need to know the underlying details of automation scripts/logic to run tests with different test data sets.

6 Implementation

During scripting, the following guidelines should be followed:

- Ensure that automation script implementation follows standard coding guidelines of the programming language to make scripts effective & readable. This includes, but is not limited to, adding documentation to the code, formatting, etc.
- Prepare the test data in a particular format (string, text file, xml file, image file, etc.) for a given script.
- Identify elements to be interacted with in the application under test.
- Follow a design pattern such as Page Objects for web application automation. This is even more significant in the agile process, where the system under test is evolving or changing frequently. Using Selenium/Web Driver as API for automation and Page Factory along with page objects is recommended for better organization of scripts and usage.
- Keep the script generic and testable for future updates while developing automation libraries.
- Execute scripts in target environments (as specified in the project plan).

7 Execution

In a continuous development process, where automation scripts play an important role in certifying builds for quality, it is recommended to follow a continuous script execution process. This can be achieved using a Continuous Integration server, such as Jenkins, to run scheduled executions on predefined test environments or nightly development builds to produce automation test results. The continuous execution mechanism provides a regular feedback loop for uncovering defects in the system under test, the script, or the data in use. By following an iteration based model, where we begin with a set of core requirements, build necessary features in the framework (if not already present), and add utilities, the automation script development process becomes simplified.

In a continuous development process, where automation scripts play an important role in certifying builds for quality, it is recommended to follow a continuous script execution process using a Continuous Integration server.

8 Automation test results saving/tracking/management

Modes of capturing test results through xml, html, logs, etc. may be easy or available as part of the test framework, but the files lack the ability to provide fail counts, insights into historical test executions and customizable metrics.

Capturing and presenting automation test results are as important as creating tests. There are several common modes of capturing test results such as xml, html, logs, etc. Those types of files are easy to implement or available as part of the test framework. Although the files provide for easy review of current/recent test results, they lack the ability to provide insights into historical test executions and customizable metrics. They cannot track execution details such as fail count, based on ongoing or future executions, on a given configuration (browser/OS/test environment). One solution is to integrate with a database management system such as MySQL. It is also advisable to enhance the framework to integrate the database for storage mechanism for all automation test artefacts including test cases, test data, etc. Having additional automation test configuration details like test environment, multi-browser support, and test data for data-driven tests, provides for better control over test execution.

9 Automation metrics

Accumulated test results can be used for analysis and generating metrics, which can help fine tune the test data and test cases.

Stored test results accumulated over a period of time can be used for analysis and generating metrics. Such metrics can help in fine tuning test data and test cases. Additionally, it can help in improving quality of the application under test and the quality of test process, as the tests are run on a scheduled basis. Having database integration for capturing test results helps to get real-time test execution updates into the database, thereby providing results until the execution is completed. If the execution is aborted before it completes, execution data is available up to the point the test is run. Such execution information stored in a database can also be used to automatically mark a test as flaky if it fails often. Such classification of tests can help an automation tester to analyze/debug and fix tests, test data, or the environment configuration. Another advantage of classifying tests as flaky is that such tests can be excluded from regular execution until they are fixed.

10 Automation testing for release support

Jenkins configuration is necessary as it helps in detecting critical defects considerably early. It can be used during regression testing and in the pre-production validation phase.

Jenkins configuration is necessary for optimum results. It is important to have the main job configured to run sanity tests covering all high-priority tests on selected environments. Then, secondary jobs will kick in to run more tests. Such a setup helps in detecting critical defects considerably early. It can be used during regression testing and in the pre-production validation phase.

Summary

Automated tests can be performed frequently and rapidly. They are beneficial for projects with longer maintenance lives. In testing, the ability to react with ease to an evolving or changing system, and requirements, is extremely important. Automation testing can be the success factor for a project if there is a proper process in place, and that process should be simple, practical and adaptable to changing requirements. The 10 steps documented here are lessons learnt from multiple automation project implementations at Tavant Technologies, the challenges encountered and subsequent solutions, and the best practices implemented. The guidelines (including step 6) can be used in any automation project with minimum or no modification.

About Tavant

Headquartered in Santa Clara, California, Tavant is a digital products and platforms company that provides impactful results to its customers across North America, Europe, and Asia-Pacific. Founded in 2000, the company employs over 2500 people and is a recognized top employer. Tavant is creating an AI-powered intelligent enterprise by reimagining customer experiences, driving operational efficiencies, and improving collaboration. Find Tavant on LinkedIn and Twitter.

About the Author

Raghavendra Kulkarni
Senior Lead, QA

Raghavendra is involved in Web Application Automation Testing solutions for automation projects, primarily using FIRE Framework and other open source test tools. He also provides support for Automation POC using FIRE and develops enhancements to the FIRE Framework.