Test Automation and Software Development

Amit Srivastava
Test Automation and Software Development

Amit Srivastava

Technology Review #2002-07

Test Automation and Software Development

Amit Srivastava

December 2002
Contents

1 INTRODUCTION ........................................................................................................ 1
  1.1 MANAGEMENT’S ROLE IN TEST AUTOMATION ......................................................... 1
  1.2 PLANNING FOR TEST AUTOMATION ........................................................................... 2
  1.3 PRINCIPLES OF TEST AUTOMATION ......................................................................... 3
    1.3.1 Hierarchy ......................................................................................................... 3
    1.3.2 Atomic Tests .................................................................................................. 3
    1.3.3 Independent Tests ........................................................................................ 3
  1.4 ATTRIBUTES OF TEST SUITES .................................................................................. 4
    1.4.1 Maintainability .............................................................................................. 4
    1.4.2 Reliability ...................................................................................................... 5
    1.4.3 Performance .................................................................................................... 6
    1.4.4 Optimization .................................................................................................. 6
  1.5 LOGGING AUTOMATED TEST RESULTS ................................................................... 6
  1.6 BENEFITS OF TEST AUTOMATION ............................................................................ 7
  1.7 MANUAL V/S AUTOMATED TESTING .......................................................................... 8
  1.8 CONCLUSION ............................................................................................................ 8
1 Introduction

In this era of cut-throat competition, every organization wants to take early movers’ advantage in introducing new products. This has led to drastic reduction in the product development cycle time. Testing, an important phase of the development cycle, has also suffered from this shrinking of cycle time. Testers are usually given lesser and lesser time to complete their testing, which makes them more prone to committing errors.

Test Automation is one of the most effective and popular solutions for meeting aggressive deadlines and delivering a product of quality. This is achieved by a marked reduction in the QA cycle time, without compromising on the test coverage and the ultimate quality of the product.

Though most of the organizations agree to it in principle, many are apprehensive about its outcome because of the initial costs involved.

This paper outlines the changes required by an organization, which is adopting test automation as a means to strengthen its testing process.

1.1 Management’s Role in Test Automation

Top management’s commitment/initiative is required for any software organization to successfully implement the automation of their testing process, because of the huge initial costs involved. The company needs to establish a separate group/department for test development and automation. This is required, because automation of tests requires, in addition to program development skills, knowledge of the testing framework and automation tools. This knowledge may not be available to the groups involved in product/software development.

Management needs to clearly define the test automation group’s responsibilities and their interface with other groups (such as products, etc.) of the organization. An organization might be developing many products, but in order to achieve economies of scale, it may be desirable to keep only one Test Automation group, which caters to the needs of all the product groups.

It is always advisable to keep the test automation team separate from the test execution team, as the two require entirely different skills. Management should ensure that the test developers have good programming skills. They should have a good exposure to software development and testing principles before they take up test automation. Management should not have the misconception, that test automation is a means to make testers develop programming skills. Putting unskilled programmers in test automation delays it, and thus, the organization fails to derive the full benefits of test automation.

Management should ensure that a process is in place for projects to start their test (automation) planning in the early stages of product development, in line with the ‘V’ Principle of Testing (Common Body of Knowledge ‘CBOK’ for CQSA, QAI).
Test development and automation are time-consuming processes and it needs to be ensured that the automated test suites are ready for testing before the first release of the product.

Most of the organizations start their automation late, mostly when they are hit by the high cost of manual regression testing. This not only increases their cost of testing, but also the value from automation is less, in this case.

1.2 Planning for Test Automation

The Test Automation Group needs to do a lot of planning before it starts taking assignments from the software/product development group.

The test automation group needs to decide the test automation tool to be used for automation. This decision should be based on the probable range of products they are going to test. Various kinds of test automation tools are available in the market. The purchase decision may be based on whether the product to be tested has a GUI, Command Line or Web Interface, or whether network-related testing is required, among other matters.
The testing framework also needs to be developed/acquired for test automation. This framework, as far as possible, should be independent of the products to be tested. This means that the same framework, with some addition/modification, should be usable for most of the similar products of the organization.

1.3 Principles of Test Automation

1.3.1 Hierarchy

Instead of just writing tests and clubbing them, based on the functionality they test, the test developer will now have to focus on the test suite hierarchy. A test suite is a collection of test cases with a hierarchy associated with them. This hierarchy assumes great importance in the scenario of automation.

There will be numerous setup steps that would be performed by the tests to reach a desired state. Many tests will have some common setup steps. These steps need to be identified and moved to a higher level to improve the performance of the test suite. This will also ensure that time is not wasted in the execution of tests where the desired condition would not be achieved. For this, the testing framework needs to ensure that the tests below the level where the setup has failed, are not executed.

1.3.2 Atomic Tests

A single test should test only one functionality of a product. This will ensure that the failure of a test does not leave any functionality untested. In cases where more than one functionality is being tested by a single test, and it fails while testing the first functionality, the remaining functionalities will remain untested.

1.3.3 Independent Tests

It has been seen in many test suites that if one test fails, then all the subsequent tests are not executed. This happens because these tests were dependent on each other. This inter-dependence of tests is beneficial if one is planning a manual test execution, as the tester will not have to repeat the steps, thus achieving greater productivity. In an automated test execution scenario, such interdependence will either result in re-running of the tests, or will leave many tests untested. There might be situations where the tester might not even achieve the test completion criteria (in terms of the percentage of test completion). In such situations, the tester will have to resort to manual testing to finish the execution cycle, thus defeating the automation effort. This will become clearer from the example given below:

Assume a test suite has three tests:

1. Create a file.
2. Change the permissions of the file.
3. Delete the file.

In manual execution, one would get higher productivity if the file created in the first test is modified in the second test, and then deleted in the third test. In the automated test suite too, this approach might appear to give a higher performance, but then this perception is dangerous. A failure of one test (which might be due to test script failure,
network failure, etc.) will result in the other tests remaining untested. For completing the testing, the tester will either have to re-run the test suite or might have to manually complete the testing for the tests that remained untested.

Another disadvantage is that in a test suite having inter-dependent tests, the independent re-running of a single test would not be possible, though this is a frequent real-life requirement.

1.4 Attributes of Test Suites

1.4.1 Maintainability

Though maintainability is a quality attribute required in all software, in test automation, maintainability is a core attribute. Without this, all the automation effort would be a waste and the cost of automation might be even greater than the cost of manual testing.

The automated tests will have to be modified for every minor change in the software that changes its functionality or interface. If a test suite is automated in such a manner that all such small software changes affect the suite in a major way, then a lot of effort would go in keeping the test suite up and running.

Maintainability, the most important quality attribute of a test suite, is also the most difficult to achieve. This is especially because the test developers automate a suite keeping the environment strictly under their control. The temptation of this control is so overpowering, that developers end up hard-coding their scripts to such an extent that maintaining them becomes a big task.

Maintainability can be achieved by:

**Extensive use of variables**

Avoid hard coding of values in the test scripts by declaring variables. Once the variables are defined for all the values needed in a test script, one can decide on making them global or local variables depending on their usage.

**Return or Error Message**

The tests abort when they don’t get the expected result. Before every such abortion, tests should log in an error or return message. This message is very important both for the analysis of test results as well as for the maintenance of the test suites, in case the error is in the test suite.

**Modularity**

The infrastructure for the automation of the test suite should be modular in nature. For all main features of the product, common functions should be written, which can then be called in the test scripts for achieving a desired state of the environment.
For example, in the testing of an Operating system, functions can be written for creating files and folders with the desired permissions. The desired permission here will be passed as an argument to the function.

Despite taking all care, there will be some amount of maintenance involved with the automated test suite, because the software product it tests will definitely undergo some changes over a period of time.

1.4.2 Reliability

A software product is as reliable as the test suite that tested it, and therefore, reliability of the product is also dependent on test developers. The important issue that arises is how to measure or control the test suite's reliability.

The test suite developers test their newly automated test suites on a stable system. Most of the time a test that passes on these systems is termed as complete. It is here that they commit the biggest mistake. A test that has passed on the stable system might have the logic written in such a way that it will pass successfully even when the functionality that it is testing it fails in some future revision of the product.

A thorough review (code walkthrough) of the code can bring out these logical errors. This review should first be done by the developer and then by the reviewer. Reviewers should challenge all the verification steps and see how they will behave, in case there is an unexpected result. This process will bring out many surprises and even the developers, who resent having their logic challenged, will start understanding the importance of the review.

The example presented here illustrates this point:

Scenario: A test fails if the value of Field1 is not equal to “abc”, and the value of Field2 is not equal to “xyz”.

The test developer writes the code:

```c
If ((Field1 != 'abc') & (field2 != 'xyz'))
{
The test fails. Log failure message
}
```

In a stable system, the fields will have their expected values and this test will pass as expected by the test developer. The error in the logic is that the test will pass even if the value of only one field is correct, whereas the product requirement was that both the fields should have correct values. The correct logic was to use the ‘OR’ logical operator instead of ‘AND’.
This was a small mistake made by the test developer, probably made in the pressure to meet deadlines. It could have caused havoc in execution of the software, while the test would have cleared the software of any errors.

To avoid such small, but high-risk mistakes, a code walk-through of all the test scripts is a must.

1.4.3 Performance
When should we start automation?

When the cycle time gets reduced and the pressure is increased on the testers, is the best time to start automation.

Performance, an important attribute, has still been rated below maintainability for the reasons provided below.

The cycle time of automated testing will also include the time test developers take to port the automated tests to the latest version of the software product under test. This time is directly proportional to the maintainability of the test suite.

The test developer may be tempted to test more than one feature in a single test or may write complex setups such that the control over the test environment is lost or may have tests that are interdependent. If this happens, it will reduce the reliability and maintainability of a test suite. This is why the first focus should be on maintainability and reliability, rather than on performance.

1.4.4 Optimization

One software defect should be reported by only one test of a test suite. The test suites are not just a random collection of tests. A test suite should be developed in such a manner that the above requirement is met and at the same time, no functionality is left untested.

Achieving this objective will ensure that the test suites do not have redundant tests, which end up testing the same functionality more than once.

1.5 Logging Automated Test Results

A proper logging and archiving of automated test results is more important than automating the tests. An improper Test Result Report can result in the failure of the whole testing process.

The most important point is that the test result should state a test as a failure only if the test fails to do the operation that it states. If the script fails at some other point, that should not be termed as test failure. This ‘other point’ could be a setup or network failure.

If all the test script abortions are termed as failure, then situations may arise where one software defect would result in the failure of numerous tests. This will become clearer from this example of the UNIX system
This test suite will test the file creation feature of the UNIX system, i.e., a user can successfully create a file if he/she has proper privileges. There are other tests for this product, which would require a file to be created for the execution of the test. Here, the creation of the file is a setup step for bringing the system to a desired state required for the tests to be executed.

In the above example, if the file creation feature of UNIX fails, then the first test that was for this feature will fail and would be justified in doing so, as this was the feature it was testing.

The tests that had file creation as a setup step too will abort. The abortion of the script for these tests should not be termed as test failures. Putting them as failed tests will misguide the Managers, as they will assume that the fix of one problem is going to make all their tests pass. They might even go ahead and release the product with one known bug. The truth is that in the tests where setup failed, the feature that they intended to test was never tested. All these tests should be classified as untested. Classifying them as untested will depict a correct picture of Test coverage and Test/software failure.

The result directory for the test suite should also contain the complete result logs for all the tests that failed. These result logs will help in analyzing the failures and debugging the test scripts, if required. To make this analysis easy and smooth, it is important, on the part of the test developer, to give meaningful return and error messages in the tests.

1.6 Benefits of Test Automation

Cost

Automation reduces the cost of testing, as the number of human resources required is much less as compared to manual testing. The cost advantage of automation is gained only when automation begins early in the product development life cycle. This is because automation requires heavy initial costs for building infrastructure and then for automation of test suites. Also, some cost is involved throughout the life of the product for maintenance of the automated test suite. Therefore, a minimum number of testing cycles is required to make automation cost-effective.

Reliability

Test automation reduces the risk of human error involved in manual test execution. In manual testing, the reliability of test results is totally dependent on the tester. There cannot be a process that can guarantee that the testing results reported are 100 percent accurate.

Automation of test cases reduces this risk. The organization can put a process in place, which can ensure that the test scripts written are correct. The reliability of a test script can be increased by allocating good programmers for test development and then by having rigorous code walk-throughs of the test scripts.
Performance

Automated testing reduces the testing cycle time and hence the time-to-market for a product. The automated test suites can run unattended for a long duration of time. Therefore, 24 hours’ utilization of the hardware resources can be achieved without having to call the testers in shifts.

1.7 Manual v/s automated testing

Automated and manual tests are not mutually exclusive. They must coexist to improve the overall testing productivity.

The testing process will be most benefited if one has an optimum mixture of automated and manual tests. The automated tests should usually be those, which cover the most important features of the product and are likely to be executed in all the regressions.

It will never be possible to have all the test suites automated. Some tests cannot be automated because the tool or testing framework does not support automation. For example, with a console-testing tool, the automation of GUI tests will not be possible. There might be other tests, whose automation is not possible because the product under test requires some manual hardware intervention to execute those tests.

Apart from these, there can be tests, whose automation is not very cost-effective. They might require enormous amounts of the developers’ time for automation/maintenance and might test a small, not very important feature of the product under test.

1.8 Conclusion

In conclusion, it can be said that test automation is more of a change in the mindset than anything else. This would mean a change in the mindset of the management, the testers, and the test developers. Test automation requires a top-down approach, i.e., the top management of the organization, will have to start the process—as reaping the benefits of test automation requires a lot of initial investment.

Test automation involves all the phases/activities of software development and thus, the people involved in it must have sufficient exposure to software development. This is the fundamental change required at all levels, to ensure the success of test automation.