Test Automation Architectures: Planning for Test Automation

Douglas Hoffman
Software Quality Methods, LLC.
24646 Heather Heights Place
Saratoga, California 95070-9710
Phone 408-741-4830
Fax 408-867-4550
doug.hoffman@acm.org

Keywords: Automated Testing, Test Oracles, Automation Architecture, Test Models

Abstract

Designing a practical test automation architecture provides a solid foundation for a successful automation effort. This paper describes key elements of automated testing that need to be considered, models for testing that can be used for designing a test automation architecture, and considerations for successfully combining the elements to form an automated test environment.

The paper first develops a general framework for discussion of software testing and test automation. This includes a definition of test automation, a model for software tests, and a discussion of test oracles. The remainder of the paper focuses on using the framework to plan for a test automation architecture that addresses the requirements for the specific software under test (SUT).

Introduction

Many managers, especially those outside of software quality, have a simplistic view of test automation. Test Automation is more than a set of tests run to generate apparent results. It includes designing testware, implementing automated test cases, and monitoring and interpreting a broad range of results. Automation by simply running test cases without human interaction doesn't provide interesting test exercises. We must know how the SUT reacts before the exercise can become a useful test. In fact, automating the running of tests generates data much more quickly than manual testing, and therefore there is more data to sift through before knowing how the SUT responded during the tests. This requires more time from testers and can result in less effective tests. Elements besides the SUT output can also be directly effected by the SUT, and these need to be evaluated in automated tests as well.

Prerequisites for effective test automation include several factors beyond having test cases and automated mechanisms for running them. Organization for the test cases is needed so we can select and understand what test run, When problems are encountered we need to be able to run selected subsets or start in the middle of the tests. Automated test execution is what most testers think of when thinking about test automation. But, test results need to be captured and expected outcomes compared. To automate capture and comparison the data has to be machine readable, which is somewhat difficult for many classes of results including GUI displays, screen navigation, and program states. Automated comparison of results is highly dependent on the form of information being compared. Filters often need to be made to compensate for expected differences.

Test automation can only be successful when we keep in mind that testware in general, and particularly automated testware, is easily made obsolete by changes in the SUT and environment. The architecture must take into consideration that changes will occur and be designed to minimize the impact of such changes.

A Definition of Automated Software Tests

Manual testing can be described as a situation where a person initiates each test, interacts with it, and interprets, analyzes, and reports the results. Software testing is automated when there is a mechanism for tester-free running of test cases. I generally call test cases automated when all of the following elements are present. If one or more elements are absent I consider the tests semi-automated. (Which is often the most cost-effective.)

- Ability to run two or more specified test cases
- Ability to run a subset of all the automated test cases
- No intervention is needed after launching the tests
- Automatically sets-up and/or records the relevant test environment parameters
- Runs the test cases
- Captures the relevant results
- Compares actual with expected results and flags differences
- Analyzes and reports pass/fail for each test case and for the test run

Key Factors in Automated Testing

The first step in planning for test automation is to identify and understand some key factors about the SUT: Identify what software is to be tested, its specific components and features we want to test, and the environment surrounding the SUT. These factors are critical to the automation architecture. Additionally, understand the existing and available testware elements and tools for testing and test automation in the SUT’s environment.

Although sometimes obvious, it is often enlightening to formally describe what is it we want to test and distinguish it from all other elements in the system. It is also important to identify what things we think are outside the scope of our automation or we don’t intend to test. Several related applications and utilities with different interfaces may comprise the SUT, possibly even running in different environments. Early decisions on which components to include, which to exclude, and which features are most important can put definitive boundaries on the automation tasks and substantially reduce their complexity.

After the SUT elements are identified the environment and interfaces must be considered. For large or complex SUTs there are often multiple environments to consider and the core programs to be tested may have several GUI and API interfaces. The immediate technical environment is important to automation because it provides the facilities and constraints on the most practical approaches. Multi-tasking, process communications, pipes, standard comparison utilities, etc., mold the mechanics of the test automation. Tools and utilities may be unique to one environment or they may be available across platforms. Many tools and utilities that are available in multiple environments have incompatible interfaces, which can make porting of automated test cases as difficult as using different tools in the different environments.

The form of the data for input and results capture is also important. Inputs and results may be keystrokes, data communication messages, pictures, sounds, digital device inputs or outputs, display information, etc. Comparison of binary data is different from characters, and sometimes the data has both syntactic and semantic components that must be dealt with. (e.g., In a data base search, the order of record retrieval may vary but the same records should match the search criteria and the contents of each record should be the same. Another program's output may be postscript listing, which can change substantially without changing the final printed form.) Utilities may run in different operating modes or systems from the core application. Creating a unified test automation architecture is difficult when the SUT needs to be tested in multiple incompatible systems environments. Depending on the requirements in a given situation, a unified architecture may be most effective for the automated testing system or it may be advantageous to employ multiple tools and mechanisms for the different environments.

It is not necessary to have one automation architecture that covers all the components. Testing can succeed when the tester and automation tools perform a useful test and draw proper conclusions based on the results. Automation is valuable to augment the tester by performing tasks that are tedious or impossible for a human or are more cost effective to automate. Different types of tests are run at different times and not all related tests have to be run at one time. There is little advantage in a unified automation architecture when there are substantially different products or environments and very little common testware across them. Two simple automation engines are often easier to create and support than one overly complex one.

The scope of the planned automation tasks also depends on the existing and available testware elements and tools. The testware elements include all of the software, documentation, test cases, data, programs, and associated procedures needed for all the test activities. The tools include operating system utilities, SCM, test selection and control programs, comparison routines, etc., that are employed to do the testing and automation. Although availability and current use of automated tools does not necessitate that automation be based on them, it is common for an automation architecture to have a requirement to include existing testware. (But, if the existing testware and tools were really effective we would not need to develop a new automation architecture.) Whether to constrain possible architectures by demanding inclusion of existing testware should be carefully considered, as it is always possible to continue to use existing automated tests as they are without compromising more effective automation architectures. Frequently the most cost-effective way of dealing with legacy tests is by gradually phasing them out as they require updating.

Two areas of test automation require special attention. Results capture can be a huge task when you realize the world of possible results from running software. The actual results of running a test are much more than viewing information on the screen. System environment variables, memory and file contents, program status, messages, etc., may all be impacted by the correct running of a program. Since we are looking for errors, we must include all the things that the program could possibly impact – a much larger set of things we need to check as ‘results.’

Results interpretation is a second area that can spell trouble for an automation effort. We need to identify those things we are going to verify as ‘the results’ and we need to have an oracle to tell us what those results should be.[1,2] For manual testing we look at the displays, investigate internal variables and program states, and satisfy ourselves that the test passed or failed. For automated testing we must program the automation tools to perform the same task. Manually, the tester decides the sequence of tests and investigations, and it varies based upon the data they find. There may be one test, but there may be several ‘correct’ results from it, and there are a large number of ways an error would manifest itself. In an automated test environment we must plan the verification of results so that we know when the test passed and when it doesn’t. That means identifying all important indications of errors and systematically checking them after every test.

A Model of Testing

Software testing involves more than feeding inputs to a program and observing results. Software today also has states and interacts with stored data and the computer environment. Figure 1 models the inputs and results for some software. Such a model is important in test automation because it provides categories to identify the inputs and results that must be monitored and manipulated during automated testing. For even a simple automated test that feeds inputs to the SUT, the automated test should verify the expected direct results and the postcondition program state to be sure the SUT did the right thing and ended in the correct program state. If the program isn’t supposed to change the system environment or any data sets, then some verification should be performed to confirm the correct environment and data values after the test. (These are particularly difficult to do in many systems and extraordinarily important. Manual testers see these types of problems as “program hangs after completion,” “user no longer has correct permissions,” or “data corruption from unknown source” errors. When we automate tests we must ensure that such problems are detected of we risk incorrectly passing software that obviously doesn’t work.

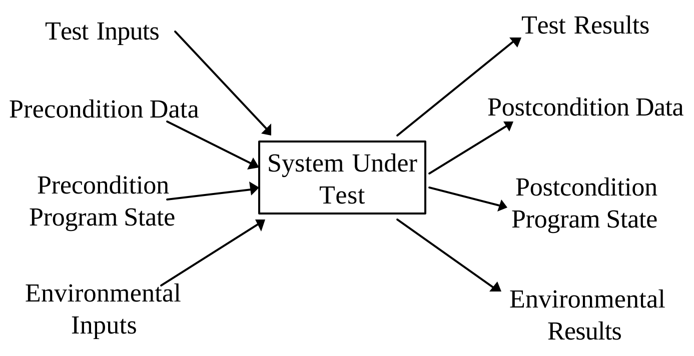

Figure 1: Expanded Testing Model

Software Test Oracles

Test oracles are the mechanisms for generation of expected results so we have something to compare with the SUT responses. For manual testing the oracle is most often the human running the tests. With automation of tests it becomes necessary to automatically generate the expected results in a computer compatible form, and then have the computer system compare actual with expected results[3].

A tester can interpret system behaviors such as GUI navigation and can also perform diverse functions like arithmetic computations, string concatenation, or data base interrogations. Humans know that many factors such as dates and times change and can factor that in when generating or comparing results. The human oracle is so capable that they are often unaware that they are continually modeling program behavior and comparing results.

However, human oracles are not perfect for test automation and some kind of computer based oracle has to be used to fully automate software testing. A human is sometimes slower than computers for generation of expected results. The computer can flash screens faster than a human can capture the images, and volumes of data can be generated much faster than a human can interpret and compare it. Humans are also limited in our view of program behavior. We can’t observe system internal data or program states, sometimes lose concentration and therefore miss things, and are easily “trained” to overlook errors by seeing a repeated pattern even when it isn’t there.

Architecting Test Automation

To this point in this paper the factors and models have been generic; they apply for any SUT. But, it is not true that any single automation mechanism today fits all situations for SUT. The SUT itself plays a major role in the architecture of the test engine. Assuming that good test cases exist, automation of SUT testing involves specific input values being fed and corresponding results checked through the SUT's interfaces. The process I use for deciding upon the architecture for test automation is derived from models and analysis from the key factors. Models for the specific SUT are used to identify interfaces with the environment and to split the testing into smaller components. Once the SUT and environments are understood, the best places and mechanisms for automation can be selected. From this information an architecture can be articulated that provides the best solution to the automation requirements.

Understanding the SUT for automation requires more than evaluating the inputs, processes, and outputs. Each of the eight input and output sections in the Testing Model must be identified in the SUT. Software can effect data values in memory, stored in databases, and program internal states. There are also side effects to program execution such as resource utilization (disk space, printers, system variables, system environment). Programs can interfere with one another, consume all of a resource, change the system state, etc. Manual testing factors in most of these effects because a human observes more than just the SUT. Test automation needs to account for these results from running the SUT. This is most easily done by modeling the SUT and identifying the various interface points. Figure 2 is an illustration of a simple model for some SUT. A model such as this for the specific SUT for which the automated test environment is intended is used for several things.

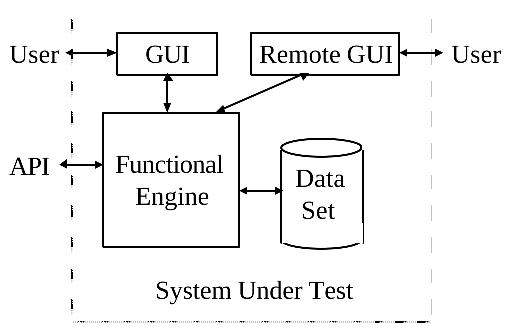

Figure 2: A Model For SUT

Given the model, we might split the testing problem into several smaller problems. The majority of functions performed by a program might be tested through a public or private API. Automating tests run through a programming interface is usually easier and more reliable than through a GUI. A GUI front-end can then be tested independently of the program functions. This means that functional testing is done with programs that can directly feed and retrieve values through the API. GUI testing becomes an exercise to see that values typed into fields are fed into the back-end correctly, that values passed from the back-end are displayed correctly, and display windows are traversed based on the rules for GUI navigation. Figure 3 illustrates a splitting of the testing problem for the SUT modeled in Figure 2.

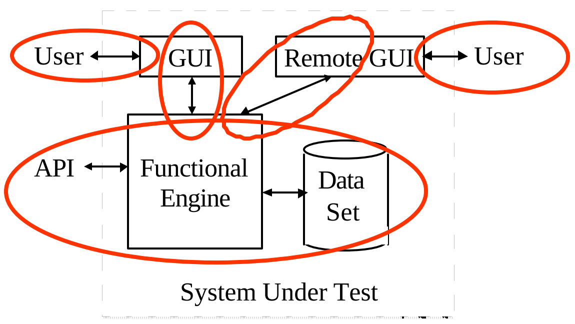

Figure 3: Splitting the Testing Problem

An automation architecture needs to include the SUT, test running, monitoring tools, program inputs, program outputs, stored data, program state, environment characteristics, results capture, results comparison, outcome analysis, and results reporting. Each factor is interconnected, but all are necessary for test automation to be effective.

Automated Test Architecture Descriptions

We have now collected the information needed to describe the architecture. This is the step in the process where "a miracle occurs." Alternatives are weighed based on the requirements and available information. Factors such as the availability and accessibility of inputs and results, the availability or ease of creation of tools, stability of the SUT, availability and practicality of oracles, testing priorities, and resource availability must be weighed with the technical requirements. The result should be a practical, cost effective automation architecture.

There are two parts to the description of the automation architecture. The first is a structural description that shows the elements and connections between them. Like a data flow diagram, information can be displayed and its control and movement depicted. An example of a structure diagram is shown in Figure 4. The second part of the description is the sequence or flow of events. What event starts what process, and when. This second part explains the process for the automated test sequence. A typical sequence is shown below. Not all steps need to be automated. There is always a cost tradeoff to be considered. Each step must be done, but the cost of automating them may be prohibitive for some of the activities.

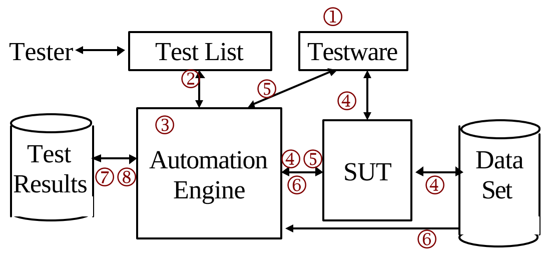

Figure 4: An Automated Process Model

Flow of events

1. Testware version control and configuration management
2. Selecting the subset of test cases to run
3. Set-up and/or record environmental variables
4. Run the test exercises
5. Monitor test activities
6. Capture relevant results
7. Compare actual with expected results
8. Report analysis of pass/fail

Note that the two descriptive elements may be combined into a powerful description of the automation environment. By labeling the structure diagram with the event numbers we can depict which elements perform what tasks in time sequence.

Conclusion

Test automation is much more than computers launching test programs. An automation architecture includes many factors that need to be understood and addressed before automating testing. It begins with understanding test requirements, the SUT, and the test environment. Using modeling we can understand and analyze the testing and automation problems. This information can then be applied to describe a test automation environment using structural diagrams and event sequences.

[1] Douglas Hoffman, "A Taxonomy for Test Oracles," Proceedings of 11th International Quality Week, May, 1998
[2] Douglas Hoffman, "Heuristic Test Oracles," Software Test & Quality Engineering, V1, I2, March/April 1999
[3] Cem Kaner and Douglas Hoffman, "Thoughts on Oracles and Software Test Automation," Proceedings of 12th International Quality Week, May, 1999