Exploratory Test Automation

Douglas Hoffman
Software Quality Methods, LLC.
24646 Heather Heights Place
Saratoga, CA 95070 USA
doug.hoffman@acm.org

Summary

Cem Kaner and I started teaching techniques for automated ET in the late 1990's, calling them "high volume test automation." These techniques go after bugs that are virtually impossible to expose or isolate in manual testing. One of the challenges in teaching automated ET is the extent to which sophisticated testing relies on a deeper knowledge of the application under test. As I've worked with investment models over the past two years, I've realized that this is a type of application that can probably capture the interest of most of the people in our community and thus serve as a good foundation for explaining where I think the next generation of testing should be headed.

This paper explores some of the non-regression automation techniques that explore the SUT. The talk concludes with examination of how to use this viewpoint to better prepare tests and report results.

Exploratory Test Automation

First, some working definitions will help clarify what I mean in this paper:

> A ***Test*** is an exercise designed to detect bugs. It is generally composed of set-up, execute, clean-up, and initial analysis of results.
> A test ***Passes*** when the software works as expected. Here, nothing was noted during test execution or initial analysis.
> A test ***Fails*** when it indicates there is an error in the software under test (SUT). Here, something was noted either during test execution or in the initial analysis of the outcomes.

When we run a test there are observations made about the general behavior of the system and specific behaviors related to the test itself. The general behavior may include such things as memory leaks, display behavior, network access, etc. that are not specifically being tested but could manifest errors if bugs are present. The specific behavior is directly related to the test exercise and the expected results. The results are the elements explicitly checked as part of the initial test analysis.

Test Result Possibilities

There are two outcome alternatives when we run a test: we observe behaviors that, according to our oracles, indicate (or don't indicate) the software is buggy. Further action is indicated when the test results flag us to an error or we observe something unusual during the running of the test. The conclusion that further work is required is what I call failing a test. Determining whether or not there is an underlying bug requires further investigation of the unusual condition after the initial failure indication. We know when the test is complete whether we will pursue any bugs. In 20/20 hindsight we [may] know whether or not a bug exists. Only time will tell.

A test passes when there are no indications of any problem. There is nothing to do other than record that the test was run.

Either way, though, there may or may not be an error. When we have a failure, investigation may show a reportable [suspected] error. The investigation may also show that the error indication is due to one of many possible other causes. We don't automatically report all failing tests specifically because of these false alarms. (False alarms their sources are explored further below.)

Table 1 illustrates the four possible states after running a test. There are two possible states regarding bugs in the SUT: either there are no errors in what we are testing, or there are errors an excellent test should expose. A test can indicate either pass or fail. There are four combinations of the two. When a test passes we are unaware of any need for further investigation. This is the correct action if there are no errors present. However, we will also take no action if there are errors present but unnoticed (Type I error). This latter case I call a "Silent Miss" because we do not know about any error from this test and therefore will quietly ignore it.

Situation / Test Result	No Bug in SUT	Bug in SUT
As Expected (Pass)	Correct Pass	Silent Miss
Abnormal (Fail)	False Alarm	Caught It!

Table 1

A failure tells us that further investigation is required. There are the same two possible situations regarding bugs in the SUT: either there are no errors in what we are testing, or there are errors. We investigate when a failure occurs, so we will find out which is the case. It's a good thing if we find a bug. That's the purpose of having the test and it has correctly identified that the bug is present. A "false alarm" occurs when a test indicates a failure but there is no bug in the SUT (Type II error). Here, we are led to investigate because of the test failure and through the investigation we discover that there is nothing apparently wrong with the SUT. The other cause could be an error in the test or some condition that caused unexpected but normal behavior from the SUT. Unexpected because something happened that was different from the anticipated behavior of a bug-free SUT (bug free relative to what the test is looking for). Having a false alarm is not a good thing. The time spent running the test, analyzing the results, and investigating the suspected failure tells us nearly nothing about the quality of the SUT. All that invested time results in no information about the SUT.

A Model for Software Test Execution

It is important for a tester to realize that the four situations exist. There are many reasons for silent misses and false alarms. The model of test execution below helps explain why we can have undetected errors in the face of tests and why some errors are intermittent or not reproducible.

The software test execution model includes the elements that impact SUT behavior (such as data and system environment) and helps articulate what to look at to verify test results. (See Figure 1.) Understanding and identifying the elements leads to more creative test design and better validation of outcomes.

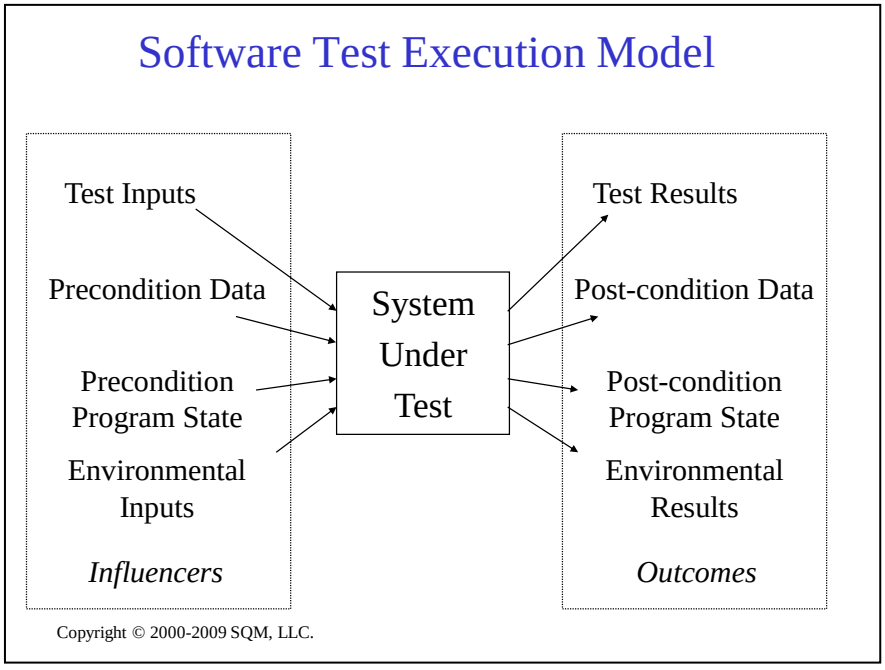

Figure 1

The four areas that influence the SUT behavior (Influencers) are test inputs, precondition data, precondition program state, and the environment. There are corresponding areas where the SUT may have outcomes that should be verified (Outcomes). The areas are described briefly in Table 2.

Terms	Definition
Test Inputs/Results	The test exercise and expected results from proper behavior of the SUT
Precondition/Post-condition Data	Files, databases, and interim values used or set by the SUT. (This includes any data that could be accessed by the SUT when a defect is present.)
Precondition/Post-condition Program State	The internal program state variables. Two examples are an error dialog and an input screen. (Many of the program state variables are internal to the SUT and cannot be easily set or detected.)
Environment	System factors such as hardware components, memory size, network characteristics, operating system, other running programs, user permissions, etc.
Influencers	Elements that affect the SUT behavior. This includes the test inputs, precondition data, precondition program state, and the environment. Note that influencers include all elements that possibly affect the behavior of the SUT, and therefore the scope is infinite.
Outcomes	Elements that are affected by the SUT behavior. This includes the test results, postcondition data, postcondition program state, and the environment. Note that outcomes include all elements possibly affected by errors in the SUT, and therefore the scope is infinite.

Table 2

The influencers include an infinite number of possibilities in the three areas we don't generally control: precondition data, precondition program state, and environment. What we control are the test inputs. Often we validate or control some of the input data, program state, and environment. We cannot control all the factors in any of the three domains, especially in light of possible SUT errors that may reference influencers we never considered (e.g., opening the wrong file). When we cannot duplicate an error we did not monitor or control a critical influencer.

Likewise, there is an infinite space of possible outcomes in the data, program state, and environment we don't normally consider, decided not to check, or missed in our analysis and test design. Again, this is especially true in light of possible SUT errors that may affect outcomes the SUT's intended behavior would not touch (e.g., writing to the wrong file). We will not detect an error if we don't check the outcomes where the error manifests itself.

Questions to Ask Ourselves

There are a number of questions we can ask that will improve our tests based upon the software test execution model. Answering the questions may improve the quality of tests.

Are we controlling or monitoring the best influencers?
Are we checking the most important outcomes?
What values/conditions should influence the SUT? How do we set/cause them?
How do we know the expected outcomes?
What influencers are we not controlling or monitoring?
What outcomes do we know we are not checking?
What gives us confidence that the test isn't missing bugs?

Implications

There is possibility for errors whether a test indicates passing or a failing. A silent miss occurs when we aren't checking the affected outcomes, the test has a bug, we don't notice a failure the test exposed or we have chosen to overlook it. The error may be detected later through some other process, but the test indicated that no further action is required based on the test results. This means that we should be skeptical when a test indicates a "pass." It's always possible that there is still a bug in the SUT in the area under test in spite of a pass indication.

A false alarm could be due to many causes; one of the influencers causes unanticipated outcomes (but normal for the SUT given the value or event of the influencer). At the end of our investigation we conclude that there is nothing wrong with the SUT. This means that we should be skeptical when a test indicates a "fail." It is always possible that there is no bug in the SUT in spite of a failure indication.

Pass/Fail metrics don't really give us interesting information until the cause of every pass and fail is understood. Unless we validate the pass indications, we don't know which tests missed a bug in the SUT. Because we don't really know which passes are real (and are unlikely to investigate to figure out which are real), any measure of the count of passes misrepresents the state of the SUT with regard to the testing. Likewise, the count of failures is only meaningful after thorough investigation and identification of which failures are due to SUT errors and which not.

Skepticism is healthy when reporting test results. As much as we'd like to have definitive answers and absolutes, the results from our testing are inconclusive, especially before failures have been completely investigated. Initial test reports should be qualified with the fact that the numbers are subject to change as more information is gathered about failing tests. Just as the government regularly revises past economic indicators when new information is available, so should we treat passes and failures as estimates until we have gathered all the evidence we expect to glean from the tests.

Conclusions

There are lots of reasons for "passing" or "failing" a test other than bugs in the SUT. Simply stated, "pass" doesn't mean the SUT passes and "fail" doesn't mean the SUT fails. They are indications of whether or not further investigation is necessary. Tests that pass provide no reason to investigate; the outcomes were consistent with the SUT behaving as expected. There may be problems in the area of the SUT that test focuses on. Whether the problems were surfaced by the exercise or not (but not noticed), pass means we didn't note anything out of the ordinary, not that the SUT works.

Tests that fail provide reason to investigate the SUT because the test turned up something unexpected (or an expected indicator of a bug). Through the investigation we decide whether there is a bug present or if this is a false alarm. Finding a bug is a good thing, that is the purpose of testing, after all. In the case of a false alarm, we have learned nothing new about the SUT from the test. The problems surfaced by a fail don't always come from SUT errors, so a failing a test doesn't mean that there is a bug present.

We aren't checking all of the results, so we know we may miss some errors. We don't know the outcomes from arbitrary errors, so we can't know all the right results to check.

Pass/Fail metrics don't really give us interesting information. "Passes" include an unknown number of actual SUT bugs. "Fails" overstate the number of SUT bugs until all the failures are isolated and causes identified.

By asking questions about the influencers and outcomes we can create better tests and do better testing. Knowing more about them can lead to better reproducibility, catching more bugs, and improved fault isolation.