

Why Tests Don't Pass

Douglas Hoffman
Doug.Hoffman@acm.org

Abstract

Most testers think of tests passing or failing. Either they found a bug or they didn't. Unfortunately, experience shows us repeatedly that passing a test doesn't really mean there is no bug. It is quite possible for a test to surface an error but it not be detected at the time. It is also possible for bugs to exist in the feature being tested in spite of the test of that capability. Passing really only means that we didn't notice anything interesting.

Likewise, failing a test is no guarantee that a bug is present. There could be a bug in the test itself, a configuration problem, corrupted data, or a host of other explainable reasons that do not mean that there is anything wrong with the software being tested. Failing really only means that something that was noticed warrants further investigation.

The paper explains the ideas further, explores some of the implications, and suggests some ways to benefit from this new way of thinking about test outcomes. The paper concludes with examination of how to use this viewpoint to better prepare tests and report results.

Biography

Douglas Hoffman has over 30 years experience as a consultant, manager, and engineer in the computer and software industries based on a solid foundation in computer science and electrical engineering. He provides organizational assessments, strategic quality planning, and test planning services. His recent technical work has focused on test oracles and advanced automation architectures. He is an ASQ Fellow, member of ACM and IEEE, holds ASQ Certificates in Software Quality Engineering and Manager of Quality/Organization Excellence, and has been a registered ISO Lead Auditor. He holds credentials for teaching Computer Science at the college level and has done so at the University of San Francisco, UC Santa Cruz Extension, and Howard University.

Douglas is a Past Chair of the American Society for Quality (ASQ) Silicon Valley Section and the Santa Clara Valley Software Quality Association (SSQA), a task group of the ASQ. He is a member of the Board of Directors in the Association for Software Testing (AST) and committee member for the Pacific Northwest Software Quality Conference (PNSQC). He is also a regular speaker at software quality conferences including STPCon, STAR, PNSQC, Quality Week, and others.

Copyright © 2009 Software Quality Methods, LLC.

Published at the Pacific Northwest Software Quality Conference 2009

Why Tests Don't Pass¹

Douglas Hoffman

Summary

When we execute a test the results come down to either finding a bug (the software failed the test) or not finding a bug (the software passed the test). We either have something to investigate/log into the bug database or not. Unfortunately, experience repeatedly shows us that passing a test doesn't really mean that there are no bugs present. It is possible for the test to miss the bug. It is also quite possible not to notice an error even though a test surfaces it. Passing a test really means that we didn't detect anything out of the ordinary.

Likewise, failing a test is no guarantee of the presence of bugs. There could be a bug in the test itself, a configuration problem, corrupted data, or a host of other explainable reasons that are not due to anything wrong with the software being tested. Failing really only means that something that was noticed warrants further investigation and possible reporting.

This paper explores these ideas and some of the implications further and suggests some ways to benefit from this new way of thinking about passing and failing. The talk also examines how to use this viewpoint to better prepare tests and report results.

Why Tests Don't Pass (or Fail)

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.

A test *Oracle* is the principle or mechanism by which we decide whether the SUT behavior is good or bad.

We make observations about the general behavior of the system and specific behaviors related to the exercise when we run a test. These observations are evaluated through one or more oracles. General behavior may include such things as memory leaks, display behavior, network access, etc. that are not specifically being tested but could manifest errors from bugs present in the SUT. The specific behavior is directly related to the test exercise and the expected software behaviors

¹ Extending work first presented at the Conference for the Association for Software Testing (CAST) 2009

from the exercise. 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) that the software is buggy. Further action is indicated when the test results flag us to an error or we observe something unexpected during the running of the test. After the initial failure indication, further investigation of the unusual condition is necessary to determine whether or not there is an underlying bug. When the test is complete and we've done the initial analysis, we know whether we will pursue any bugs. In 20/20 hindsight we may know whether or not a bug exists.

A test passes when there are no indications of any problem. In this case, there is nothing to do except record that the test was run.

Either way, though, there may or may not be an error in the SUT. After we have a failure, investigation may show a reportable bug. The investigation may also show that an 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 and some of their sources are explored further below.)

Table 1 illustrates the two possible outcomes and four possible states after running a test. There are two possible situations regarding bugs in the SUT: either there are no bugs in what we are testing, or there are bugs an excellent test should expose. When we run a test, our initial analysis indicates either pass or fail. There are four combinations of the two variables. When a test passes we are unaware of any need for further investigation, regardless of whether or not a bug is actually present. Doing nothing more is the correct action if there are no errors present. However, we will also take no action if there are errors present but unnoticed (a Type I error). This case I call a "Silent Miss" because we do not know about any error from this test and therefore will quietly move on.

Real Situation	No Bug in SUT	Bug in SUT
Test Result		
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 bugs in what we are testing, or there are. 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 a bug is present. A "False Alarm" occurs when a test indicates a failure but there is no bug in the SUT (a Type II error). In this case we investigate because of the test failure, and through the investigation we discover that there is nothing apparently wrong with the SUT. Reasons for the failure 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). Normal because, by knowing the cause we see that the results are what we expect should happen. Having a false alarm is not good. The time spent running the test, analyzing the results, and investigating the suspected failure tells us nearly nothing about the quality of the SUT. The time invested results in almost no new information about the quality of 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 good 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.

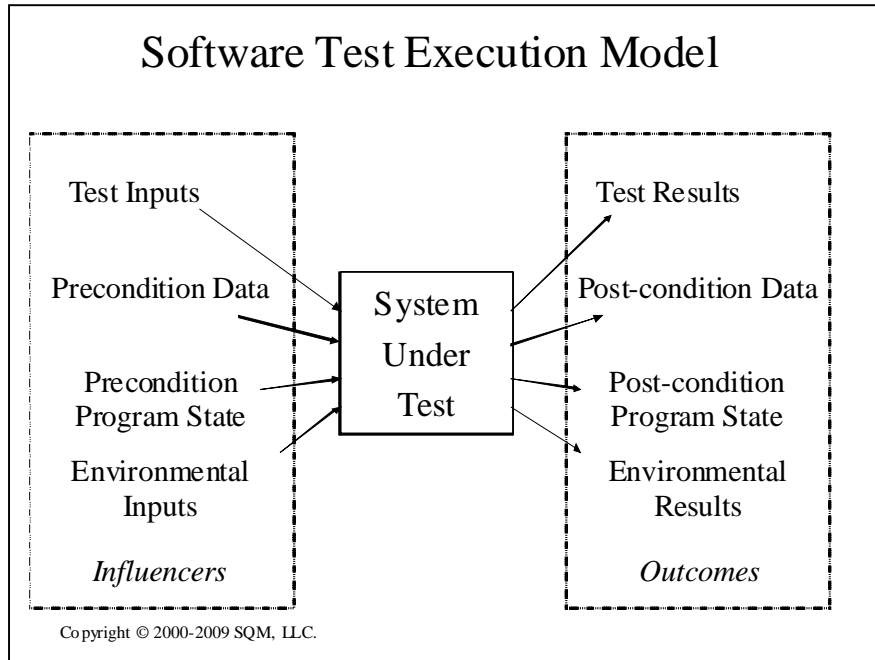


Figure 1

The four areas that influence the SUT behavior (Influencers) are test inputs, precondition data, precondition program state, and the system 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. (This is what most testers think of as the test.)
Precondition/Post-condition Data	Files, databases, and interim values used or set by the SUT. (This includes <i>any</i> data that <i>could</i> 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.

Terms	Definition
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 when bugs are present, and therefore the scope is infinite.)
Outcomes	Elements that are affected by the SUT behavior. This includes the test results, post-condition data, post-condition program state, and the environment. (Note that outcomes include all elements possibly affected by the behavior of the SUT when bugs are present, 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, but this is limited to specific elements required by the test.) 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, decide not to check, or miss 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 these questions may improve the quality of tests.

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

The behavior of the SUT is controlled by a multitude of outside influences, many of which we take for granted. To the degree possible, we should be aware of the potential for external

influences on the SUT, both those required for the test to run and those outside of the ones specifically required by the test. This becomes particularly relevant for automated tests, but applies to manual tests as well. We need to decide whether to control them, monitor them, or ignore them for the purposes of running the test exercise.

The question also comes up as to how we can monitor or control them if we want to. System resources may become unavailable after the start of testing, data values changed, etc. These things may be outside of our control. Monitoring the values may be possible, but expensive. But, even if we decide not to monitor or control, we are much more capable of investigating and isolating bugs if we are aware of the potential for external factors to influence SUT behavior.

Are we checking the most important outcomes? (What outcomes do we know we are not checking?)

How do we know the expected outcomes?

Likewise, the possible outcomes from running the test are infinite in the face of potential SUT bugs and influencers. Being aware of potential outcomes, however, does not address the problem of having test oracles to predict or analyze them. Thinking about the potential outcomes gives us a much better chance to monitor and verify them and thus detect more potential SUT bugs. Being aware of possible outcomes also provides us with more information when investigating and fault isolating when unexpected behavior is detected.

What gives us confidence that the test isn't missing real bugs?

Lastly, we should ask ourselves if we have done everything practical in the test to maximize the chance that the exercise might surface and detect a bug if it's present. Besides answering the question of whether or not this is the best exercise, we need to evaluate what we should know, record, and control about influencers and what outcomes we should evaluate.

Implications

There is a possibility for bugs 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, or we don't notice a failure the test exposed. The bug may be detected later through some other process, but the test results indicated that no further action is required. This means that we should be skeptical when a test indicates a "pass." It's always possible in spite of a pass indication that there is still a bug in the SUT in the area under test.

There is a possibility that there are no bugs even though a test fails. A false alarm could be due to many causes; one or more of the influencers cause unanticipated but normal outcomes (for the SUT, given the values or events of the influencers). 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 won't know which tests missed a bug in the SUT. But, we have no reason to do so. Because we don't really know which passes are real (and are unlikely to investigate to figure out which are), 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 results were consistent with the SUT behaving as expected. There may still be problems in the area of the SUT that test focuses on. Whether the problems were surfaced by the exercise or not, pass means we didn't note anything out of the ordinary, not that the SUT works.

Failing tests provide reasons to investigate the SUT because the test turned up something unexpected (or an expected indicator of a bug). Through the investigation, we will decide whether there is a bug present or if this is a false alarm. Finding a bug is a good thing - that's 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 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 initially. "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.