Investigation vs. Validation

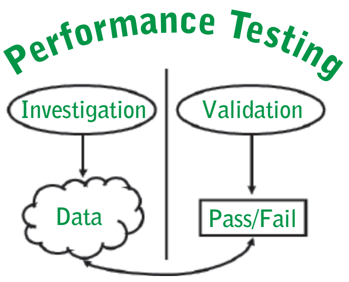
Scott Barber

Have you ever had this experience? You're explaining something that you've gone over a million times before. Suddenly, you stop in the middle of explanation one million and one and say, "That's it! Why didn't I think of that years ago?"

This happened to me just the other week while I was working to help a client improve an approach to performance testing. It was almost as if I was listening to someone else speaking for a moment, as I heard my own words replay in my head:

"We know that there is an issue on the app server that they are working on now. That pretty much means that testing requirements compliance would be pointless, but we still have to see what the new Web server hardware has done for us, right? So let's forget the requirements for now and whip up some scripts to investigate how the…."

I didn't even get to finish the thought, because Tom was asking "Investigate how the *what?*" and Kevin was looking at me as if I'd grown a second head. So I did what any good tester would do. "Hold on a second," I said, and moved briskly to the whiteboard to quickly sketch a picture that looked something like the figure shown here.



I'm sure some of you are thinking "OK, what's the big deal?" Neither investigation nor validation is a revolutionary concept for software testers. In fact, the Association for Software Testing (www.associationforsoftwaretesting.org) specifically refers to software testing as "a technical investigation done to expose quality-related information about the product under test." And one can hardly read an article about software testing that doesn't discuss "validation" in one way or another.

What struck me in that moment was not the fact that most performance testing projects necessitate both investigation and validation; it was the relationship between investigation and validation during performance testing that became suddenly clear. For years I've been trying to explain to people that the relationship between investigation and validation in performance testing is fundamentally different from the relationship between investigation and validation in functional testing. But while I understood the distinction clearly in my head, it never seemed to come across very well verbally.

Before I make my case about how these relationships differ, I should clarify my working definitions of "validation" and "investigation" for the purposes of this discussion. Whether a project is agile, waterfall or somewhere in between, at some point it becomes important to determine whether or not the software does what it was intended to do in the first place. In other words, you have to test it.

Of course, if you follow a waterfall model, a V-model or some similar model, this happens near the end of the project and takes the form of executing lots of well-planned individual tests. Generally, each one of these tests will have been designed to determine whether or not one specific, predefined requirement has been met. If the test passes, the implementation of that requirement is said to be "validated."

If you take a more agile approach, you may instead be executing tests to determine whether or not the concept sketched on the bar napkin, now laminated and tacked to the wall in the lead developer's cube, has been implemented in accordance with the vision of the original artist. Although the criteria for determining whether one of these tests passes or fails are not nearly as well defined as the ones we discussed above, a passing test is nevertheless said to have "validated" the implementation of the feature or features being tested—so pretty much any way you look at it, "validation testing" can be thought of as an activity that compares the version of the software being tested to the expectations that have been set or presumed for that product.

That takes care of validation, but what about investigation?

Presumptions of Innocence

Let's start with a dictionary definition of investigation: "a detailed inquiry or systematic examination." The first and most obvious difference between our working definition of validation and the dictionary definition of investigation is that we have stated that validation requires the existence of expectations about the outcome of the testing, while the definitions of investigation make no reference to either outcomes or expectations. This distinction is why we talk about "investigating a crime scene" rather than "validating a crime scene"; validating a crime scene would violate the presumption of "innocent until proven guilty" by implying that the crime scene was being exam-

Scott Barber is the CTO at PerfTestPlus Inc. His specialty is context-driven performance testing and analysis for distributed multiuser systems. Contact him at sbarber@perftestplus.com.

ined with a particular expectation as to what the collected data will mean.

The most well-known testing method that can be classified as "investigation" is exploratory testing (ET), which can be defined as simultaneous learning, test design and test execution. In a paper titled "Exploratory Testing Explained," James Bach writes: "An exploratory test session often begins with a charter, which states the mission and perhaps some of the tactics to be used." We can substitute for "mission" the phrase "reason for doing the investigation" without significantly changing the meaning of Bach's statement. If we then substitute "A crime scene investigation" for "An exploratory test session," we come up with "A crime scene investigation often begins with a charter, which states the reason for doing the investigation and perhaps some of the tactics to be used."

Other than the fact that I doubt crime scene investigators often refer to their instructions as a charter, I don't see any conceptual inaccuracies with the analogy, so let's agree on "investigation" being an activity based on collecting information about the version of the software being tested that may have value in determining or improving the quality of the product.

So what is it that makes the relationship between investigation and validation in performance testing fundamentally different from their relationship in functional testing?

In my experience, two factors stand out as causing this relationship to be different. The first is that typically, some manner of requirement or expectation has been established prior to the start of functional testing, even when that testing is exploratory in nature, and in last month's column I pointed out that performance requirements are rarely well defined, testable and/or in fact required for an application to go live. What this means is that, with rare exceptions, performance testing is by nature investigative due to the lack of predefined requirements or quantifiable expectations.

The second factor differentiating these activities is the frequency with which a performance test uncovers a single issue that makes any additional validation testing wasteful until that issue is resolved. In contrast to functional testing, where it is fairly rare for a single test failure to essentially disable continued validation testing of the entire system, it is almost the norm for a single performance issue to lead to a pause, or even a halt, in validation testing.

When taken together, these two factors clearly imply that the overwhelming majority of performance tests should be classified as "investigation," whether they are intended to be or not. Yet the general perception among many individuals and organizations seems to be that "Just like functional testing, performance testing is mostly validation."

The majority of performance tests should be classified as 'investigation,' whether they are intended to be or not.

Take a moment and think about the ramifications of this disconnect. How would you plan for a "mostly validation" performance testing effort? When would you conduct which types of tests? What types of defects would be uncovered by those tests? How would the tests be designed? What skills would you look for in your lead tester?

Think, too, about the chaos that ensues when a major project enters what is planned to be performance validation two weeks before go-live, and the first test uncovers the fact that at a 10-user load, the system response time increases by two orders of magnitude, meaning that a page that returned in 1 second with one user on the system returns in 100 seconds with 10 users on the system—on a system intended to support 2,500 simultaneous users!

And if you think that doesn't happen, guess again: That is *exactly* what happened to me the first time I came on board a project to do performance testing at the end of development rather than at the beginning. It took eight days to find and fix the underlying issue, leaving four business days to complete the performance validation. As you can imagine, the product did not go live on the advertised date.

Now think about how you would answer each of those questions if you imagined instead a mostly *investigation* performance testing effort. I suspect that your answers will be significantly different. Think about the projects you have worked on: How would those projects have been different if the project planners had planned to conduct performance *investigation* from the beginning? If they had planned to determine the actual capacity of the hardware selected for Web servers, planned to determine the actual available network bandwidth, and planned to shake out configuration errors in the load balancers *when they first became available?*

The chaos on the project I described above would have been avoided if there had been a plan (or a charter) in place to investigate the performance of the login functionality as soon as it became available. One test. One script. One tester. Four hours, tops, and the debilitating issue would have been detected, resolved and forgotten before anyone had even published a go-live date.

Simple, huh? With or without the drawing on the whiteboard, the entire concept that I have struggled to make managers and executives understand for years comes down to these six words: "*Investigate* performance early; *validate* performance last."