The Shoulds And Shouldn'ts Of Software Testing


Scott Barber

Lately I've become rather fond of saying that *should* and *shouldn't* are the two most frightening words a software tester hears while testing. They probably aren't far down the list of words that frighten managers during status updates either.

Consider this example. While doing some performance testing for a bank, I noticed what looked like a fascinating pattern in the results with respect to response times. After applying some statistically flimsy grouping (by rounding the response times of the measured objects *down* to the nearest whole second), I confirmed that the pattern was, in fact, quite fascinating.

Reasonable Consistency

Response time results generally follow one of just a few patterns. An individual transaction will typically display response times that are reasonably consistent throughout the test, that increase linearly or geometrically throughout the test, or that increase during peak volumes and return to low-volume response times as the volume decreases. Pretty much any other response time pattern for a particular transaction is a good indication that something worth investigating further is going on. In this case, the response times seemed to oscillate unpredictably in four-second increments.

As I investigated further, I found several items worthy of note.

First, more than a third of the measured objects returned in less than four seconds. As it turned out, these objects were all graphics, style sheets and plain HTML (text) objects. Second, the majority of the remaining objects displayed response times of four seconds—with a few just over five seconds. Third, all of the objects returning in four or more seconds contained data that they retrieved from a remote, hosted database. Last of all, every one of these remaining objects returned in "four-and-a-little" seconds, "eight-and-a-little" seconds and so on.

Pinpointing the Problem

As I explained this behavior to the team (after defending my test, test tool, analysis and competence), the developers finally agreed to instrument their code and systems to time stamp each transaction as it entered and departed their areas of responsibility. Running the test again, all of them were happy to report that their code/system was responding in a quarter of a second or less—all except the middleware developer. He reported that the requests were spending under a tenth of a second in the middleware, but that the time difference between the outbound requests to the remote database and the responses was, in fact, showing the same four-second step pattern.

The overwhelming reaction to those findings by the managers and developers was, "Oh, it must be the database"—not surprising since the main reason for the project in the first place was to ultimately replace the outsourced database. I wasn't as immediately convinced, however, and asked, "Are we sure it's not something in the network? The database is two firewalls, probably upward toward a dozen switches, hubs and proxies, and over a thousand miles away." Once the team finished gawking at me and softly chuckling to one another, their response was, "That shouldn't be the problem; we've never had trouble with the network before."

It took us three weeks to verify conclusively that the database was not the problem, two more weeks to convince the network administrators to trace the packets, one hour to conduct the test, 15 minutes to analyze the results, and less than five minutes to fix the offending setting on the recently upgraded network appliance. So, basically, the word *shouldn't* cost the project five weeks.

Of course, the hardest part for me was biting back the urge to say, "I told you so!"

Sage Advice

In his book "More Secrets of Consulting: the Consultant's Tool Kit" (Dorset House Publishing, 2001), Jerry Weinberg offers memorable rules and principles based on his years of experience in the software industry.

For those of you who may not be aware, Weinberg has authored and coauthored more books than I care to count—virtually all related to his 40 years of experience in the software industry.

While it is true that many of his lessons come from the point of view of a consultant, every one of his books holds valuable information that is applicable to anyone in the software industry (and most any other industry for that matter). So don't think that his work is interesting only to consultants. Weinberg's books and his lessons that others have shared with me have had, and continue to have, a

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

significant influence on how I approach software testing, my writing and my values and principles as a consultant. If you've never read any of his work, I highly recommend it.

If only Weinberg had gotten this book out a year sooner, and we had read and paid attention to Chapter 3, we might have avoided our five-week delay.

The first three key points in Chapter 3 are: "If they're *absolutely sure* it's not there, it's probably there," "Don't bother looking where everyone is pointing," and "Whenever you believe that a subject has nothing for you, it probably has something for you."

A mere two pages later, Weinberg introduces a concept he calls "lullaby words" that he summarizes this way:

"Later, I reflected on the deeper lesson underlying our discovery of all these lullaby words. In effect, the words discourage feedback by putting both the speaker's and the listener's mind to sleep. When feedback is discouraged, the meaning of a statement cannot be clarified. If it's not clarified, the statement can mean almost anything—and that's always the beginning of trouble. If you want to avoid such trouble, start converting those lullaby words to alarm words—words that wake you up to potential misunderstanding, rather than lulling you to sleep. Just do it!"

I'm guessing that you won't be surprised that Weinberg includes *should* as one of his lullaby words; the other six are *just, soon, very, only, anything* and *all*. While Weinberg's insights are both brilliant and useful (as usual), I have to say that I think Weinberg missed a valuable opportunity to write about why people use these lullaby words—or maybe Weinberg and I just have had some very different clients. What I have seen are members of the client's team who use lullaby words very intentionally as a method of all but begging the tester (or consultant) *not* to look too hard in a particular place.

It's a sad but true fact that team members often try to keep the testers off their turf and thus keep them from finding problems that will put their team under the microscope.

Sometimes this effort is an act of self-preservation brought on by organizations or managers who are particularly harsh when critical defects are traced back to an individual; sometimes it's a side effect of nontechnical testers trying to tell developers how to do their jobs; and other times it's simply organizational politics.

'Whenever you believe that a subject has nothing for you, it probably has something for you.'

In this case, politics is exactly what led to a five-week delay that was kicked off by the use of the word *shouldn't*. I later found out, months after phase 1 (and my official involvement with the project) was completed and in production, that there had been a long history of contention between the development teams and the infrastructure teams in this organization, allowing the development teams to basically dictate tasks to the infrastructure team and blame them for any number of defects.

Making matters worse, a few months before I came on board, a new executive vice president who, among other things, made it exceptionally difficult for the development teams to get unscheduled assistance, was placed in charge of the infrastructure team. By all reports, many of the real and perceived infrastructure issues went away very shortly after the change in management, but the balance of power had taken a 180-degree turn.

The gawks, chuckles and "shouldn't" that I received when I asked about the network actually meant, "We really hope it's the database because we don't want to fight with the infrastructure team again...especially after they assured us that the network wouldn't be an issue for this project." Now, if only someone had told me that from the beginning, we could have created a fairly simple test to prove that the problem was between the middleware and the database, as opposed to spending weeks designing, creating, scheduling, executing and analyzing a test to prove the problem was the database, which would have quickly implicated the network by default.

In this particular case, at least, it was much easier to disprove the database theory of poor performance than to prove it.

All in all, after the network fix, the remainder of the performance testing effort went well, and last I heard, the application was still performing well.

Look Where You Least Expect

But that isn't the big lesson. The big lesson that I've had to relearn time and time again since this project, which is the first encounter I had with lullaby words that I can recall, was that no matter the reason behind the use of *should* or *shouldn't*, as a tester, there is only one single response that makes sense every time: "I understand that shouldn't be the problem, but let's test it really quickly just to make sure. If nothing else, it's worth being able to document the test and conclusively rule it out."

So now whenever someone says, "This shouldn't be the problem" or "This should work," what I hear is: "Make sure *this* is something you don't forget to test."