What Counts?

by Michael Bolton

ISTOCKPHOTO

A while ago, I was asked to test a network packet sniffer. The system displayed a pie chart indicating the relative number of packets associated with each IP address that had been accessed during the past five minutes. I generated some traffic and observed that for five of the sixteen IP addresses that I had tried, the display was incorrect—the pie chart overwrote the legend identifying the IP address. Was that one test resulting in one failure or sixteen tests with five failures?

I noted that the relative proportions of each slice of the pie seemed to be correct. Then I added one more IP address and noticed the colors of the pie sections changed (they were supposed to stay the same). Now, was that one test or two? If one, then the single test failed. If two, did the first test pass and the second test fail? I noticed that the pie chart appeared to be a circle. When I resized the window by shrinking the y-axis, the circle got smaller. When I restored the window to its previous size, the circle returned to its original size. When I resized the window by shrinking the x-axis, the circle got smaller. When I resized it by expanding the x-axis, the circle again returned to its original size. Was that one test (*"resize the window"*), two tests (*x, y*), or four tests?

While I was investigating the color-changing bug, I checked the specification and found that the pie chart window was supposed to be a fixed size and not resizable at all. This new information made me realize that my earlier sizing tests had uncovered a bug, which changed my perspective on the tests that I had already performed. Was that one failing test (*one bug*) or four failing tests (*four bugs*)? To make sure I had really observed a problem, I tried resizing the window again. I wondered if this new test should be counted along with the original test (*tests?*).

A couple of years ago, I wrote an automated test for an application that checked transaction types. A four-digit field became the index into a table of transactions. About 3,000 slots in this table were populated and valid; the other slots were not. An attempt to use a populated entry was considered valid while an attempt to use an unpopulated entry was invalid. I wrote a quick bit of script to test all 10,000 possible values. One of the populated entries was inappropriately rejected. Did I do 10,000 tests (*"test each value"*), two tests (*"test all valid values"; "test all invalid values"*), or did I do one test (*"test all values"*)? Note that, in order to run this test (*tests?*), I had to populate eight other fields with plausible data for each transaction, and creating the values in some of those fields was challenging. Not only that, but they were to some degree randomized, so I was testing certain combinations. Should I have multiplied 10,000 by eight for a total of 80,000 tests? I also observed the time it took to complete transactions (turnaround time ranged from just under half a second to 1.5 seconds). Was that observation a test? Should I have added another 10,000 test cases to my count?

I'm raising these questions not because I'm some wise guy who enjoys teasing people with obscure corner cases of philosophy and logic. These aren't corner cases. Deciding what to count and how to count it are the kinds of decisions that we make every day. But more importantly, I'm raising these questions because counting tests (and requirements, bugs, and other measures derived from these counts) is an endemic means of deception in the testing business. Some well-known testing experts promote this form of deception; testers then practice it, and project communities have learned to ask for it.

The problem here is *reification*, the act of giving construct attributes to things that are merely concepts. Test cases, bugs, and requirements are expressions of ideas. If we were evaluating the work of a film critic, we would sound foolish if we were to ask how many observations she made, how many problems she had found in the film, or how many paragraphs she had written. I observe that in the testing business, we are infected with counting

"I observe that in the testing business, we are infected with counting disease—we are constantly counting test cases, requirements, lines of code, and bugs. "

disease—we are constantly counting test cases, requirements, lines of code, and bugs. Yet surely one bug can be dramatically different from the next by all kinds of dimensions—the impact on a given user, the proportion of the user base it affects, the extent to which it blocks our ability to see other bugs, the time it takes to pinpoint it, and the effort required to fix it. In a similar way, the value of two tests can differ dramatically. If this test compares an output value with some reference, and that test measures the response time for a Web transaction that involves dozens of interactions with a browser, networks, and databases, should we consider them equivalent and count them as the same?

Counting tests is like counting vehicles or cargo—where a vehicle might be a bus, freight train, tricycle, or space shuttle, and cargo might be passengers, wheat, televisions, or nuclear waste. The problem is compounded by dividing these counts by other counts to create ratios, as though cargo per vehicle were a meaningful ratio unless we knew much more about the cargo and about the vehicle—to the point where the ratio might be the least interesting thing about it.

Another fallacy with counting tests is the assumption that test results are binary—pass or fail. But that is not the question that we really want to ask as testers. That is important, but if we pay too much attention to that question, or consider that question only, we're likely to miss problems. Instead, ask a more fundamental question: *Is there a problem here?*

When I'm performing tests—configuring, operating, observing, and evaluating an application—I'm making dozens of observations every second. Some of them are important, some trivial (*"That's a button; now it's highlighted; there's a tooltip; that word is spelled correctly…"*) Sometimes I'm not even terribly conscious of making them until, all of a sudden, something catches my attention. (*"Hey, the tooltip didn't go away."*) In the course of preparing for a test of some condition, I often observe a failure of some other condition for which I hadn't intended to test. (*"The screen isn't redrawing properly."*) Often, my observation appears to be inconsistent with some expectation that I didn't even realize I had until the inconsistency appeared. When I take a car for a fifteen-minute test drive, I make evaluations of all kinds of stuff, paying real attention to what impresses me or bugs me. Those evaluations are the result of hundreds of conscious and subconscious tests. How do I enumerate those tests—and would it be meaningful for me to do so?

At the very least, we should not deceive ourselves by counting test cases. If your management asks you to do so, you may choose to provide them with the misleading information for which they are asking. It is up to you, but I guarantee that your management does not understand what "742 test cases" actually means. I don't know what it means, and I'm an expert in this stuff.

In a relatively obscure Monty Python sketch, an interviewer asks a seasoned but clearly incompetent actor about the hardest role in the theatre. The veteran responds that the answer must be Hamlet—Hamlet has 8,262 words. Othello is hard, too—but it has 941 fewer words than Hamlet. The interviewer learns quickly: "How many words did you have to say as King Lear at the Aldwych in '52?"—but the actor cautions him, "Well, now, I don't want to give you the impression that it's simply the number of words. Getting them in the right order is *just* as important."

Testers do the same thing when they count test cases and requirements, or when they compare the number of automated test cases with the number of "manual" ones—with the implication that tests are performed by machines or hands, rather than by minds. Worse, just as the doddering actor did, testers train their managers, project communities—even themselves—to accept those deceptive and seductive numbers. **{end}**

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries. He is co-author, with James Bach, of Rapid Software Testing *and a regular contributor to* Better Software *magazine. Contact Michael at mb@developsense.com.*

Do numbers really speak for themselves, or do they need a credible story to give them life and meaning?

Follow the link on the StickyMinds.com homepage to join the conversation.

Log on to www.StickyMinds.com to read and comment on all of the Code Craft, Test Connection, and Management Chronicles columns.