Don't Use Bug Counts to Measure Testers

by Cem Kaner

Should we measure the quality of testers—their productivity, efficiency and skill—by counting how many bugs they find?

Suppose that mid-project we compared the bug counts per day from exploratory testing with those obtained from regression testing (reuse of test cases that the program has previously passed) of the same area of the program. We might discover that exploratory testing yielded a much higher bug-finding rate, and this might affect our choice of testing strategies.

Leading books on software measurement suggest that we compute "average reported defects per working day" (Grady & Caswell, *Software Metrics: Establishing a Company-Wide Program*, p. 227) and "tester efficiency" as "number of faults found per KLOC" (Fenton and Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, p. 36). Should we apply these measures to individuals?

If (as I believe) the primary function of the test group is to find bugs, and the primary work product of the individual tester is the bug report, then isn't it obvious that we should compare the quality (efficiency, productivity, skill, whatever) of testers by counting their bug reports?

Obvious or not, there are two problems:

1. Bug counts are poor measures of individual performance.
2. The side effects of using bug counts as a measure are serious.

Here are two examples of disconnection between bug counts and value obtained from a tester.

In the first case, management was overeager to ship an unreliable product. I assigned one particularly talented tester to hunt for showstoppers—defects that would block shipment. He reported as few as four bugs per month, compared to rates of perhaps 25–150 from the other eleven testers. We held this product for six months past its scheduled ship date by finding at least one showstopper per week. Most weeks, one of those showstoppers (often the only one) came from our hunter. Was he the most productive member of our group or the least productive?

In the second case, a project team identified one area (graphic filters) of a new version of a popular product as high-risk. If the product shipped with the problems that we feared, we would be drowned in phone calls. The two testers assigned to this area found surprisingly few bugs. The project team (including me) worried that they might not have gotten a complete handle on the risks. I asked a senior third tester, who knew a lot about graphic filters, to assess the testing and the area. Over five weeks, he found about five bugs. He also reported that the first two testers had done good work. Partially based on this reassurance, we shipped the product a day ahead of schedule, sold a huge number of copies, but got few complaints.

In both of these cases, testers with very low bug counts did fine work and played a more significant role in the project than other testers who found many more bugs. Bug counts would have seriously mismeasured these testers.

What Do Bug Counts Measure?

"Measurement" is most often defined as assigning numbers to some attribute of something according to a rule. Unfortunately, we may not have a way of measuring an attribute (such as tester skill) and so we use something easier (like bug count). This alternate measure is called a *surrogate measure*.

The challenge of using a surrogate measure is that you need a theory of measurement that tells you (among other things) the relationship between the surrogate and the underlying attribute, the confounding relationships between the surrogate and other attributes or factors, and the predictable side effects of using this measure. Without

(Continued from page 80)

that theory, you have no justification for saying that a number is a measure of tester goodness (or of code complexity, or of maintainability, or of reliability, etc.).

So what is the relationship between bug count and tester quality? Surely, it is subtle and influenced by a wide range of other factors (such as the reliability of the code being tested and the difficulty of testing this part of the product). Without understanding that relationship, how can we say we know anything about the quality of the tester from the size of the bug count?

And as to side effects, here are just a few of the things that can and probably will go wrong if we measure individuals by counting their bug reports.

- People are good at tailoring their behavior to things they are measured against. (Check out Weinberg and Schulman's classic paper, "Goals and Performance in Computer Programming," *Human Factors,* 1974, 16(1), 70-77.) If you ask a tester for more bugs, you'll probably get more bugs. The additional bugs might be minor, or similar to already reported bugs, or design quibbles. But the bug count will go up.
- People know that other people tailor their behavior. Put a tester under incentive to report more bugs, and programmers will become more skeptical of the value of the reports they receive. Does this tester believe in this bug, they ask, or is she just inflating her bug count? Bug counting creates political problems (especially if you also count bugs per programmer).
- You can make a tester look good or bad just by choosing what type of testing she should do (regression testing often yields fewer bugs than exploratory testing) or what area she should test (fewer bugs to find in less buggy code). If raises and promotions are influenced by bug counts, project assignments will often be seen as unfair.
- Bug counting creates incentives for superficial testing (test cases that are quick and easy to create). Bug counts punish testers who take the time to look for harder-to-find but more important bugs.
- Such a system also penalizes testers who support other testers. It takes time to coach another tester or to help him build a tool that will make him more effective. The tester who does this has less time to find bugs.
- Time spent on any process (such as documenting test cases) that doesn't lead to more bugs faster is time that counts against the tester.

Several measurement advocates warn against measurement of attributes of individuals (e.g., Grady and Caswell in *Software Metrics: Establishing a Company-Wide Program*) unless the measurement is being done for the benefit of the individual (for genuine coaching or for personal discovery of trends) and is otherwise kept private (e.g., DeMarco, 1995, "Mad About Measurement" in *Why Does Software Cost So Much?*).

But We Have to Measure Something, Don't We?

When I advise clients against measuring testers by counting bugs, one response is that they need to measure their staff in some way. If bug count isn't the right measure, they ask, what is?

I don't have a silver bullet for personnel measurement. When I compare the quality of testers, I spend a lot of time looking at the quality of their work. I read bug reports. I talk with them. I talk with people that they work with. I pay attention to promises they make, and whether they keep them. These don't lend themselves to quick and easy number crunching, although you can (perhaps with difficulty) do comparative ranking of testers based on this detailed qualitative look.

If you really need a simple number to use to rank your testers, use a random number generator. It is fairer than bug counting, it probably creates less political infighting, and it might be more accurate. STQE

Cem Kaner, Ph.D., J.D., is the senior author of Testing Computer Software *and of* Bad Software: What to Do When Software Fails. *He consults and teaches courses on software testing and practices law, focusing on the law of software quality. Contact him at* kaner@kaner.com, www.kaner.com, *or* www.badsoftware.com.