May/June 2009 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

STAND AND DELIVER
Panic-free presentations

THE GOOD, THE BAD, AND THE VIRTUAL LAB
Is VLA right for you?

Issues about Metrics about Bugs

by Michael Bolton

ISTOCKPHOTO

In my travels, I've worked with a number of companies that have attempted to assess the quality of their testing—or worse, their testers—using poorly considered metrics. Sometimes the measurement is based on a count of bugs that make their way into the released product (escaped bugs); sometimes the measurement includes another factor, like the number of bugs found before release. To many managers, this kind of measurement has intuitive appeal: If the purpose of testing is to find bugs, then assessing the quality of the testing effort starts with looking at the number of bugs that the testers found or didn't find before the product was released. How could this appealing-sounding metric possibly go wrong?

One answer is found in *reification error* [1]. A bug isn't a concrete thing, like an airplane or an apple. "Bug" is a label for a *construct*, an idea. One idea might be that a feature seems to be missing, another might be that performance is slower than we'd like, and yet another might be that an error message is accurate but unhelpful. A bug could be something missing, or some form of behavior that we don't want to see. To Rapid Testers, a bug is *anything that threatens the value of the product* [2]. Value is multidimensional and subjective. Someone may value a buggy product that she owns over a less buggy product that she considers too expensive. Someone else may reject a product that provides excellent performance, preferring one that is compatible with his other applications. Like "value," "problem," and "acceptability," bugs are not tangible, countable things; they're expressions of part of a relationship between some person and some product [3].

To us testers, often it seems amply clear that some behavior represents a bug. A system crash, an inaccurate calculation, or a mangled record is very likely to be a problem and a threat to the value of the product. A direct violation of a reliable specification is probably a bug, as is a corrupted display or an unexpected, persistent howl from the system's speaker. Yet some evaluations require more subtlety. If the program rejects an input value as "too big" when a reasonable user might disagree, we have reason to suspect a threat to the value of the product—that is, a bug—even if the specification clearly outlines a smaller range of supported values. Is this an intentional limitation or did a business analyst misinterpret or mistype the end-user's requirements? This is why it's so important for testers to change the question "Does this test pass or fail?" to a question that better addresses a possible threat to someone's values: "Is there a problem here?"[4]

One oracle—a heuristic principle or mechanism by which we recognize a problem—might tell us that there is no problem, where another oracle would cause us to perceive a problem immediately. As testers, we may have strong beliefs one way or the other, but it's the project owner who gets to make the decision. That's why our role is not merely to report things that *are* bugs but also to report things that *might be* bugs or that *could be* bugs when viewed through a different set of values.

In addition to reporting bugs, it's also the role of the tester to report on *issues*. Where a bug is something that threatens the value of the product, for Rapid Testers an issue is *something that threatens the value of our testing*. (Some people call this a concern or obstacle. The concept is important, but the label isn't; call it whatever you like.) If we're uncertain about whether something is a bug or not, that's an issue. If we identify a problem with testability—anything that slows down testing or that makes it difficult to determine whether or not there's a problem—we *may* be seeing a bug, but at the least it's an issue. If we lack sufficient equipment, tools, or training to accomplish the mission in the required time, that's an issue. If a product that we're testing has so many problems that investigating and reporting them dominates the time we have available for finding them, that's an issue, too. We'll come back to that point.

When a product is released or deployed, it's because some person—the product owner—has decided that it's ready to go. That decision should be based on another: Does the product owner have sufficient information to make the ship/no-ship decision? That's a judgment call, based upon not only technical information but also upon business imperatives. A tester is unlikely to have authority over the schedule, the budget, staffing, or market or contractual obligations. So, while the tester helps

to *inform* the decision, he shouldn't be *making* the decision unless he is also the product owner.

For the same reason, the tester should be very careful about asserting that the product has been "adequately" or "completely" tested. A clothing salesman should offer excellent service to the customer as long as she's is in the shop, but he can't be held responsible for deciding whether the customer has bought too much, too little, or just enough. If something looks particularly embarrassing or complimentary on the customer, the salesman can assist the customer by pointing it out. If there's some interesting, new piece in the back room, it behooves the salesman to bring it to the customer's attention. But in all cases, the customer—not the salesman—is responsible for deciding what she wants to buy, what services the salesman shall provide, when he has completed his service, and whether the service was adequate.

So it is with a tester and his client. The tester provides information about problems in the product, but those shouldn't be the only items that he brings to the table. The tester may also report on benefits in the product, about comparable products, or about risks. The tester should also be prepared to point out parts of his work that he would recommend covering, but that he hasn't covered. The product owner uses that information, along with all of the other technical and business information, and decides whether she has enough information to issue the order to ship.

This is a good reason to be wary of bug metrics. The project owner is the person who ultimately makes the decision about: what is a bug, whether the known bugs are trivial enough to permit shipment, whether there are sufficiently important open questions such that shipping would be unwise, and whether the business priorities outweigh the technical ones. These decisions will have a profound impact on any attempt to count bugs, either before or after the product is released.

Bugs in the product may inhibit our ability to find bugs. Some problems—blocking bugs—may make it difficult to execute tests by preventing access to parts of the product that require further testing. Other problems—intermittent bugs—may cause test results to be inconsistent, inconclusive, or ambiguous. If there are large numbers of bugs to fix, programmers may provide us with a large number of builds that we must reinstall and reconfigure—or they may provide us with a single build on which we have to do dozens of fix verifications before we can pick up again with other tests. All of these interruptions—setup, configuration, bug investigation, and reporting—take time away from the design and execution of new tests, wherein we obtain more coverage of the product. So, a buggy product gives bugs more time and more places in which to hide, requires more time to test to the same level of coverage, or both. That's an issue—a very common and very serious issue.

If there are many bugs in the product after release, it may well be that the testers have done less than excellent work. Yet there are many other plausible explanations: a very complex product, inadequate programmer testing, an overly aggressive schedule, a rational business determination by product management that the known problems in the product aren't worth fixing, or any or all of the above. Next time, we'll talk about the risk of metrics based on possible motivations for them: inquiry or control? **{end}**

References

[1] Levy, David. *Tools of Critical Thinking: Metathoughts for Psychology.* Waveland Press, 2003.

[2] Bach, James, and Michael Bolton. "Rapid Software Testing." www.satisfice.com/rst.pdf

[3] Bolton, Michael. "It's All Relative" in Fiona Charles, et al. *The Gift of Time.* Dorset House, 2008.

[4] Bach, James, and Mike Kelly. "Is There a Problem Here?" www.michaeldkelly.com/pdfs/IsThereAProblemHere.zip

Who decides what the criteria for a bug are in your organization? What do you do to make sure that your metrics prompt questions and investigations, rather than drive decisions?

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