When in Doubt, Reframe

by Michael Bolton

GETTY IMAGES

Several years ago, I was hired to test a product that had been released a year earlier. The product itself wasn't expected to change, but the development organization was preparing a major upgrade to the platform on which the product ran. I was assigned to test a specific module that did data conversion. My assignment was to update some test data, revamp the test documentation, and retest the system for important problems based on test ideas from the previous year's effort. That mission was specific and focused, but in the course of getting ready for some on-charter testing, I did a little experimenting. I entered the digit 9 into every space in every field of an important data input screen and pressed Enter. The program promptly crashed, leaving only an ugly Java stack exception on the screen. I showed it to Alan, the development lead for the project. His response surprised me. First he said, "You're supposed to be testing the conversion module!" Then he added, "Besides, no user would ever do that."

I paused for a moment. It was certainly true that I was supposed to be testing the conversion module, but was it really the case that *no* user would *ever* do that? I had just done it, so what Alan was saying was clearly irrational. Or was it? I remembered a thread from Jerry Weinberg's SHAPE Forum, in which Jerry had said, "Software maintainers definitely have to learn to deal with 'irrationality,' and the first step in that learning is to stop calling it irrationality. Call it 'rational from the point of view of another set of values.'" Treating Alan's reaction as irrational wasn't going to help matters. I needed to re-evaluate and reframe his response.

The first step was to examine my charter. When Alan said, "You're supposed to be testing the conversion module," was he concerned that I was spending too much time on weird problems and not enough time on problems that were more obvious, more common, or—more importantly—within my charter? That's a potentially serious occupational hazard for testers. It's easy to misinterpret our mission as "Find as many problems as you can as quickly as possible," when the mission is really "Find the most important problems as quickly as possible." I had just started on the project and had yet to establish credibility with Alan. Whatever my response, I needed to make sure he knew that I understood the mission.

Could he have been pointing out the risk of wasting time duplicating the work of other testers? That would have been a rational concern if the test had taken a long time to prepare or execute, or if there had been a similar report anywhere in the bug-tracking database. However, the test had taken only a few seconds to design and run, and there was no evidence that this bug had ever been reported. I had heard that the poor product quality had surprised the development team on the first go-around. Using a heavily scripted process, the contractor test team had found 500 bugs, most of which had been fixed before shipping. But according to the test lead, 1,500 new problems had come in from the field in the first three months after release. Now, merely counting bugs or bug reports requires skepticism—a simple count ignores a lot of important information, such as the severity, complexity, and frequency of each problem—but the essence of the story was that the contractors had missed a significant number of problems.

How likely was it that a real user would trigger the problem in production? The probability of a normal user intentionally entering exactly that data in normal circumstances was practically nil. So I had to consider and address the possibility that I was overestimating the likelihood of the bug.

Had I been submitting a bug report formally, it would have been my responsibility to investigate the problem and to provide Alan with as compelling a scenario as I could, but at this point I hadn't done that. Instead, I was mentioning the problem in passing—talking with a live oracle to help me understand more about the bug and its significance.

Did the crash present a real risk? A crash is evidence that there is some unhandled condition in the program, which puts it into an unpredicted and therefore

unpredictable state. This program interacted with many other programs, and without further testing I couldn't be sure that they would be ready to handle the crash gracefully. Moreover, my test had exposed a problem by a fairly arcane scenario, but perhaps there was a less extreme and more typical circumstance that would trigger it. When we are imagining what users might do, we need to consider extreme behavior, even if it's uncommon. Perhaps a normal user might trigger that condition by accident. Perhaps a malicious user might exploit it to try to crash the program. Perhaps a curious person might attempt to "test" the program while in production.

Perhaps we should reframe the stock phrase "No user would ever do that" to, as James Bach says, "No user I can think of, who I like, would do that on purpose—and if he did, I'd hope that the system wouldn't let him." Perhaps Alan was thinking, "I sure didn't design it to handle that, and the spec didn't have anything to say about this situation, so I assume no one else thought of it either. And if we start thinking that way, there are a lot of other things that we'll have to think about, too."

"No user would ever do that" might be missing the words "as far as I know." That appeared to be the case in this organization; developers were rarely in contact with the users of the program, insulated by several layers of help desk, business analysts, and middle managers. "No user would ever do that" might also be missing the words "after we've trained him not to do that." Indeed, several bugs on this project were deferred to be addressed as "training issues."

I also had to consider that Alan's reaction had nothing to do with me personally, but with the bad news or its surprising nature. A bug means extra work for someone, and the initial reaction to a report may have a strong emotional component. The missing words could be a note of irritation ("We've got real work to do here!") or fear ("There might be a ton of bugs just like this in this program and even more that are only a little like this. Don't pull on that thread; our whole program could unravel").

I took a breath and chose my words carefully. "I know that I'm supposed to be testing the conversion module. In the early stages of a project like this one, I'm learning about the product. While doing that, I tend to do some quick sanity checks on the stability of the program and the work of the other testers. It doesn't take much time. Plus, there's the possibility that I might find something important at a very low cost, and it helps me to be sure that things are stable enough to be tested. I'm concerned that if I'm finding a crash at this early stage with a pretty simple test, there are other big problems still to be found. I will focus on the conversion module, but if I find more problems like this one—even if they're not in the conversion module—do you want to hear about them?"

Alan considered this for a moment and said, "Well . . . I guess." As time went on, we found and fixed a lot of bugs.

One key and often underemphasized testing skill is listening for what our clients might be saying aside from their specific words. Another is in recognizing and developing consensus on what the mission is. A third is in digging up buried assumptions. All of these skills benefit from expansive views of what might be possible. Whether in a conversation or a specification, the words provide clues—but they might not be the whole story. **{end}**

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries as part of James Bach's Rapid Software Testing course. Michael is also program chair for the Toronto Association of System and Software Quality. He is a regular contributor to Better Software *magazine. Contact Michael at mb@developsense.com.*

BETTER SOFTWARE CONFERENCE & EXPO SPEAKER

Is there a sentence that you're used to hearing on development projects—a sentence for which you've developed a translation?

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