Users We Don't Like

by Michael Bolton

GETTY IMAGES

As testers, it's practically inevitable that at some point in our careers we'll show someone a problem we've found and we'll hear, "No user would ever do that!" What that often means is "No user that I've thought of and that I like would do that on purpose." A skilled tester will pose the consequent questions: What kind of user haven't I thought of? What might a user I like do by accident? What kind of users don't I like?

People who we intentionally don't like are called disfavored users (see the StickyNotes for a reference). Black hat hackers are canonical examples of disfavored users—people who might use or exploit our software with malicious intent, whose requests we don't want to honor, and whose intentions we want to stymie in some way. Disfavored users might include pirates, spies, embezzlers, and data or identity thieves—anyone who might intrude, snoop, or steal in a way that compromises privacy, reliability, safety, or security. If things are too easy for disfavored users, there is a risk that they'll exploit vulnerabilities in our software.

Still, some hackers are great black box testers. They often find catastrophic bugs and security holes without access to the inside information that we have. They have strong technical skills, and they're often self-taught. They use lightweight, powerful tools and they learn to become familiar with patterns of weakness. *We don't like hackers' motives, but it might be useful for us to respect their skills, to use their tools, and to look for vulnerabilities as capably as they do.*

While I was teaching the Rapid Software Testing course to a group, I asked "What kinds of users don't we like?" I was fishing for someone to respond, "Hackers!" Instead, the first answer to my question was one I didn't expect: "People who don't read the manual!" A lot of people in the room laughed. I did, too.

Hmm. I remembered my technical support days at Quarterdeck, when I would sometimes take calls from people who seemed unable to read the manual that we supplied. Had they done so, they would have learned about startup switches and program parameters that would have solved their problems. Those people were wasting my time and theirs.

Then I remembered the opinion of an older, wiser support manager, who suggested that the only good parameter was no parameter at all. He believed that software should be clever enough to determine when a given parameter is necessary and simply configure itself without requiring the user to specify anything.

I realized that sometimes it is hard to like people when they're not happy with us—perhaps because we've failed to serve them in some way. *Thinking about people who don't read the manual might remind us to identify and report problems that the software could identify and solve on its own.*

Suddenly, the class was on a roll. Someone suggested that we don't like impatient users. They're not willing to wait for our slow software or to follow our convoluted workflows. By clicking around impatiently before the product is ready, they may expose timing problems and resource conflicts. Besides, hurried or impatient people make mistakes and then blame us for not alerting them to the problem. *Thinking about impatient users might help us look for usability, performance, and timing problems.*

Someone else piped up, "We don't like Luddites (people who insist on using old stuff), because whenever they buy a new version of our product, they complain about performance on their old, slow machines with their outdated operating systems and browsers—even though our product is supposed to support those platforms." If we're supporting the older platforms, we need to include them in our tests. *Thinking about Luddites might remind us to test for a wider diversity of platforms.*

"We don't like our network managers," said one participant, "because they're control freaks. They keep griping about installation problems. Once they've got the software installed, they gripe about configuration and support problems." Support people have important jobs to do. They serve many people, and they must do it quickly. Something that takes an extra moment for a single user might take dozens or hundreds of extra moments for a network support group. *Thinking about network managers might remind us to test for configurability, supportability, and scalability.*

Then things got a little controversial. I suggested that we don't like people who are getting older (as we all are—this is

the year that I got bifocals). Oh, we say we like them, but if we observe the way that we design and test software for them, it is sometimes hard to see respect for their legitimate needs. *Thinking about people who can't see very well might remind us to test our products for accessibility—for example, at a wider variety of screen resolutions.*

Naturally, we don't like developers. They're always assuming that we're supposed to find the bugs, and then they get upset with us when we do. Then they complain that they can't read our bug reports or reproduce the problem on their machines. And on top of that, they don't provide log files or scripting interfaces or build notes or the things that would make the product easier to test. Nonetheless, developers are arguably our most important customers. *Thinking about developers reminds us that our job is to help make them look good and to provide them with excellent feedback quickly, and that to do that effectively we have to be helpful and collaborative. Thinking about developers also reminds us to ask for testability.*

We don't like managers. They never seem to know what they want, they keep changing our priorities, and yet they're always the first to blame us when a bug slips through. But managers have it tough; they have many constituencies to satisfy, they need to know the kind of information about the product that we can provide, and sometimes they don't understand what testing can and cannot do. *Thinking about managers reminds us that we're a service to the project and reminds us to obtain consensus and keep focus on the testing mission.*

Someone suggested, "We don't like marketers and salespeople, because they make all kinds of claims about the product and sometimes those claims turn out to be optimistic or outright bogus. Then the salespeople say that they thought the product could do those things, based on some long-forgotten remark by some developer in some meeting a while ago." *Thinking about the salespeople reminds us to check not only the claims made by designs and specification but also the claims in the marketing materials, in meetings, in email, or in hallway conversation. If the product can't fulfill the claim, then the product—or the claim—must be changed.*

Of course, throughout the exercise, we realized that we didn't really dislike any of the people who we had mentioned. They're all members of our project community, and they all have needs and concerns that might be legitimate. There's the old saying that "the customer is always right." I like Karl Wiegers' claim that the customer isn't always right, but the customer always has a point (see the StickyNotes for a reference).

As we test, we become experts in configuring and operating our products. When we encounter problems, we often learn to work around them while we're waiting for them to be fixed. Those problems might recede into the background and lose prominence for us due to our familiarity and expertise with the product. One way to get better at noticing problems is to consider users we don't like—those we legitimately don't like, and those we profess to like but whose needs aren't apparently being addressed. {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.*

STAR WEST SPEAKER

Sticky Notes

For more on the following topics go to www.StickyMinds.com/bettersoftware.

- Disfavored users
- Karl Wiegers

Which other users don't we like, and why don't we like them? Is it because they represent a threat to our products or because we've disappointed them in some way?

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