July 2007 $9.95 www.StickyMinds.com

BETTER SOFTWARE

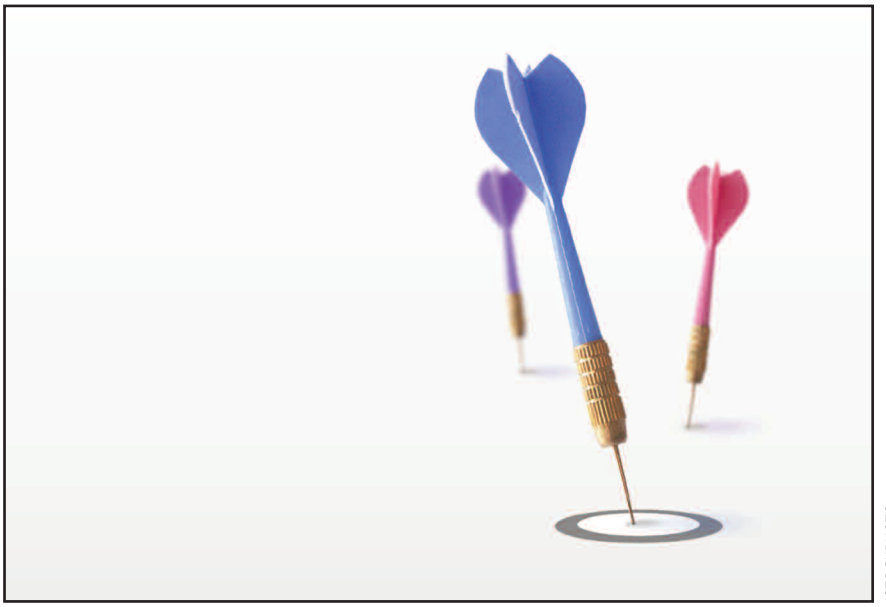
GET TO THE SOURCE
SCM as a testing solution

SAY WHEN
Defining what's "enough"

The Print Companion to StickyMinds.com

Test Design with Risk in Mind

by Michael Bolton



ISTOCKPHOTO

For Rapid Testers, "risk" has at least three interpretations. A risk might be "a bad thing that might happen"—an *error* or *omission*, like a missing requirement or coding mistake. It might be *consequences*—undesirable aftereffects of the first kind of risk; examples include system crashes, loss to the business, or an unflattering article in the business section of the newspaper. Finally, a risk might be a *chance* that we're prepared to take that one of the first two risks won't happen. "We're short on time, and few customers still use Windows 2000, so we'll take the risk of not testing it."

Sometimes in testing we find problems that surprise us—that we didn't anticipate at all. In risk-based testing, we try to anticipate a problem and then test for it. Risk-based test design is based on questions that begin "What if . . . ?" What bad things might happen? What good things might fail to happen?

Rapid Testers model the risk story in four parts. There's a risk when a *victim* may be affected by some *problem* caused by a *vulnerability* in the product that is triggered by some *threat*. By thinking expansively about each part, we reduce the possibility that we'll miss an important risk.

For this discussion, a *victim* is an agent—a person, group, or system that interacts with the system under test—that suffers inconvenience, annoyance, damage, or loss because of a problem. General systems thinking reminds us that there are many possible notions of who a user might be and how certain users might perceive the product or a problem with it. A few years ago, I worked on a banking application designed to be used by bank tellers. Those tellers—the people we thought of as "the users"—were certainly potential victims of software bugs, but if those bugs caused delays for the bank customers waiting in line, then the bank customers were victims, too. If the bank customers lost confidence in the bank or left in droves due to slow service, the bank's shareholders became victims. If coding and testing errors that enabled the bug reflect badly on our reputations, then we are possible victims of a problem. Ultimately, a risk affects somebody—often several somebodies.

For *problem*, I recently found G.F. Smith's definition: "an undesirable situation that is significant and may be solvable by some agent, although probably with some degree of difficulty" (see the StickyNotes for references). One way to model risk is to consider quality criteria. Quality is subjective—"value to some person," as Gerald Weinberg says. A problem threatens that value for some potential victim. If the problem doesn't matter, or seems to affect only unimportant people, the problem will be deemed to be *no problem*. There may be substantial business risk in declaring a group of people to be unimportant. An expert tester not only develops risk ideas and scenarios that reveal the importance of a problem but also searches for people who might have been misclassified as unimportant.

A *vulnerability* is some weakness in a system that could allow a problem to occur. Systems are bound to have flaws. People specify them and people code them. People are fallible and communication between them is more fallible still. A vulnerability might be the result of a programming error, an unexpectedly incompatible library, a specification weakness, or a misunderstanding of a requirement.

Still, a vulnerability doesn't matter unless (or until) it is triggered. A *threat*—some state or transition, whether arrived at deliberately or inadvertently—turns a vulnerability into a problem. A vulnerability could hide forever in an area of the code that is never accessed in testing or production or that requires an improbable set of variables to be in some state, and we'd never know a thing about it. Hackers might trigger a vulnerability deliberately, but well-meaning people could trigger it inadvertently. When they're confused, enterprising users will often say, "What if I try this?" If no one else has tried that, the workaround might work—or trigger a serious vulnerability.

In risk-based testing, we consider variations of victim, problem, vulnerability, and threat and make choices about how—or whether—we're going to test for them. But the combination of variations of all four aspects is almost always intractably large. Which potentional victims matter? For the ones that don't matter, *why* don't they matter? What problems might harm or annoy potential victims?

"Solo efforts are OK, but the process is much richer when other members of the project community are involved."

Which problems might be tolerable, and which ones would be catastrophic? What principles or mechanisms allow us to recognize a vulnerability in the program? How can we trigger vulnerabilities in the test lab?

Try using a half-hour brainstorming session to create a risk list. Solo efforts are OK, but the process is much richer when other members of the project community are involved. It's good to create a risk list early in the project, but since we learn about the system and attendant risks throughout development, it's useful to revisit the risk list later as a group exercise. Whenever we identify new risks, we add them to the list.

Try looking at a structural diagram showing objects, modules, or subsystems and the interrelationships and dependencies between them. Even if you ignore the labels, you can bring important risks to the surface: "What does this line mean? What error checking happens on the way into this box? What happens if the line between these two things gets broken? Does this arrow ever point the other way? Could this bucket overflow?"

Try exploring the product, alone or in pairs, emphasizing "testing to learn" rather than "testing to find bugs." Look for information about how the product works and how it might fail. Michael Kelly has a useful mnemonic guide: "FCC CUTS VIDS" (see the StickyNotes for a link). Look at features, complexity, claims, configuration, user types, testability, scenarios, variability, interoperability, data, and structure. Tour the menu options; perform actions via the mouse or the keyboard; tour the folders in which the application installs itself; and look through the help file, tutorials, and supporting documentation.

The risk list can be used to generate test ideas or to identify ways of addressing vulnerabilities before test design and coding begin, or to guide exploratory test execution, which is typically much faster than writing and executing test scripts. Most importantly, the risk list can help us prioritize risks. What problem could end up on the front page of the *San Jose Mercury News*? Which risks might cause a customer to abandon the product or to talk to friends about the problem? What problems have caused tech support calls? What problems have we already seen but become accustomed to? Quantitative measures of risk might not be useful or accurate; qualitative risk assessments might be sufficient. One heuristic is to watch for the cringe when a risk idea is discussed.

If we've done a good job of modeling risk, we'll probably have more test ideas than we can act upon. That's where the third definition of risk comes in. Some risks are sufficiently low compared to other risks, so we take a chance on not testing for them. That's OK; as Tom DeMarco says, "If a project has no risks, don't do it." {end}

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries. He is a regular contributor to Better Software *magazine. Contact Michael at mb@developsense.com.*

STAR WEST SPEAKER

Sticky Notes

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

- References

In this column, I've focused mostly on product risks. What aspects of the *project* might contribute to our risk model?

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