March 2008 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

ALL ABOARD
Keep your test coverage on track

THINK GLOBALLY
Tips for building a distributed team

Breaking Ground on SOA

How to Build and Test Your First Web Service

How Much Is Enough?

by Michael Bolton

"I *know* how to spell banana," the little girl said. "I just don't know when to stop."

An *entirely* scripted action is one in which the person performing it makes no decisions of her own; all of the ideas came from some other person, some time in the past. An entirely exploratory task comes from inside the person performing it, in the moment that the task is being performed. No test performed for a client is entirely exploratory: The exploratory tester has a mission that ultimately derives from some goal or need previously expressed by the client. No *good* test—except one performed by an automaton—is entirely scripted, either. Only the worst imaginable human tester will fail to break out of the script, pause, and take note when he recognizes a problem of some kind. Even in a heavily scripted process, testers are usually encouraged to investigate the bugs that they find—and that requires exploratory behavior. Since the bug itself was unpredicted, the decision to stop is unpredictable, too.

The nice thing about an entirely scripted task is that you *know* when to stop. The last note of the musical score, the last line of the play, and the last empty space on the paint-by-number canvas all mark the end of the work. When you're using a highly scripted approach to test, you stop when you've noticed a problem, have more questions, or have an interesting idea.

When you're exploring, knowing when to stop is less obvious. In a jam session, musicians signal each other with eye contact and musical cues so that they can fade out or stop together. Improv actors build unscripted scenes, play them out, and then freeze in position or break character when they discover a plausible conclusion. To someone unskilled in these arts, the factors that go into the decision to stop tend to be unclear or invisible. Even among performers, the less experienced might not be able to articulate the elements of the stopping decision, but the skilled practitioners can.

Exploratory testers design and execute tests in the moment, starting with an open mission and investigating new ideas as they arise. When do we stop? How can we be sure that we were done? And how can we justify our decisions to our clients?

The first step is to recognize that we *can't* be sure that we're done. Any approach to answering the stopping question is necessarily heuristic, based on some fallible, context-dependent method for solving a problem or making a decision. Heuristics serve the purpose of learning and discovery quickly and inexpensively, but not conclusively—rather like indicator lights on your car's dashboard. When any lamp on your dashboard glows red, you should consider stopping until they're all green again—*but maybe it's more important to ignore the lamp and keep going. Consider these heuristics:*

ISTOCKPHOTO

Time's Up!: If we've specified an explicit timebox for a test, we might choose to stop when we've exhausted the time that we allocated to the task. *Might our timebox have been inappropriately arbitrary? Might we want to allocate more time for further investigation?*

Piñata: When the candy starts cascading from a piñata, we typically stop whacking it. If we see something reasonably spectacular when we're testing—a crash or hang, screen or data corruption, behavior that we can't understand or explain—we might believe that we've found a problem that's dramatic enough to justify stopping. *Might there be an interesting piece of candy still stuck in the piñata?*

Dead Horse: When the application is down for the count, there's no point in continuing to test. Anything we see after this is likely to be a side effect of things that we've already discovered, or our system is corrupted and the integrity of the test is compromised, as the developers will quickly point out when we try to report. Moreover, the problems that we've already discovered will lead to bug fixes, and those changes will require more testing later. *Might we see an even more dramatic or damaging problem if we proceed a little further?*

Loss of Charter: Ultimately, we work in the service of a client. If we have any reason to believe that our work no longer has the support of the client, then we've lost the primary motivation for our test. If the client asks us specifically to stop, we're mandated to do so. *Have* *we made our client sufficiently aware of the risks associated with information that we might discover with further testing?*

Mission Accomplished!: We might determine that we've learned the answer to some specific question that we asked as we set out, and we don't expect to learn anything more than what we've already discovered. We suspect that we're done when we have a plausible story for why we think we're done, and we run this story by the client. *Might there be more to the answer than we've already found? Might new, important questions occur to us if we continue to explore?*

I Feel Stuck!: We might choose to stop, at least temporarily, when we feel stuck, severely confused, underequipped, or blocked by some bug. We might feel the need for tools, equipment, or bug fixes to proceed. *Are we really stuck, or could we use other information to proceed? Could we report that we're stuck and why, and then proceed along some parallel or alternative path?*

Flatline: During a stress test, we may see the program become unresponsive by some measure. No matter how much data we force-feed the program, it displays the same response. We might hypothesize that the program has crashed, is throwing data away, or otherwise is failing to handle the load. In any event, the program is flatlining, and it's time to pull the plug on this test. *Might the program be handling the stress and recovering?*

Cost vs. Value: All of the above heuristics are really variations on the idea that we stop when the incremental value that we are obtaining, or that we anticipate, is too low compared to the cost of continuing to test. Every activity takes time. Our current activity might have some value, but is it the most valuable thing that we could be doing right now? *Could we lower the cost of our current activity through automating some aspect of it? Could we increase the value of the activity by using human capabilities of observation, cognition, and inference?*

Testing isn't a race; we're not in defined lanes, and there's no clear finish line. Scripted approaches might tell us to stop too soon or suggest insufficient, unimportant, or distracting observations. Part of the power of exploratory testing comes from its open-ended nature; we're free to keep testing if we think there's value to be found—a bug, a new test idea, or a new aspect of the product to discover. Still, we might be wrong. If we think we're done, a problem might manifest itself moments after we've stopped testing and watching; if we're inclined to go on, we might not add sufficient value. More exploration will almost always yield more information; with increasing experience and skill, we learn to infer heuristically whether that extra information will matter. **{end}**

Are you in control of your own testing process?
How do you decide whether to stop or to keep going?

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

RALLY SOFTWARE
Scaling Software Agility
Over 50% of the world's largest companies use Rally to:
Shorten development cycles
Increase visibility and collaboration
Synchronize global development teams
RALLY
AGILE
Sign up for FREE Community Edition
Rally's award-winning Agile life cycle management tool for a single team!
www.rallydev.com/bsm