◆ SPECIAL EDITION ◆ 2010 TOOLS & SERVICES GUIDE INSIDE ◆

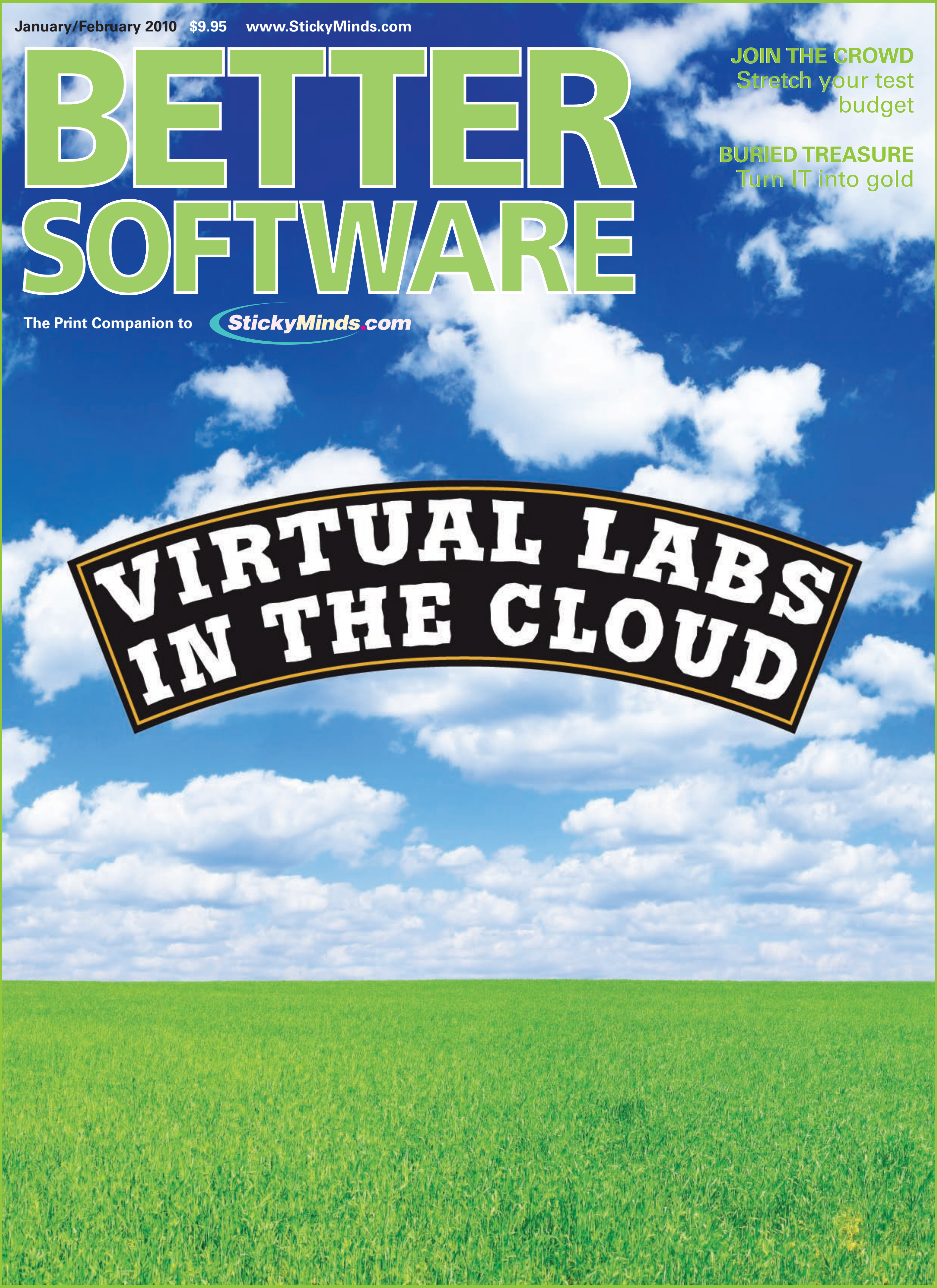
January/February 2010 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

JOIN THE CROWD
Stretch your test budget

BURIED TREASURE
Turn IT into gold



VIRTUAL LABS IN THE CLOUD

◆ SPECIAL EDITION ◆ 2010 TOOLS & SERVICES GUIDE INSIDE ◆

◆ SPECIAL EDITION ◆ 2010 TOOLS & SERVICES GUIDE INSIDE ◆

◆ SPECIAL EDITION ◆ 2010 TOOLS & SERVICES GUIDE INSIDE ◆

Swan Song

by Michael Bolton

From birth, the turkey sees people as kind, considerate caregivers. The farmer feeds the turkey, keeps it warm and dry, and protects it from predators. Every day, the turkey receives more confirmation of the fundamental benevolence of the human species. Then, a few days before Thanksgiving, the turkey gets an unpleasant surprise.

This story, cheerfully cribbed from Bertrand Russell, illustrates the theme of *The Black Swan* [1], an entertaining and insightful book by Nassim Nicholas Taleb. Formerly a trader of stock options and still an occasional adviser to hedge funds, Taleb says that his principal goal in life is not to be a turkey. He proposes that, in a complex and highly uncertain world, we can defend ourselves from rude shocks by being skeptical empiricists and by avoiding several kinds of fallacies and biases. The book reads like a charter for skilled testers.

A Black Swan, Taleb says, is a rare event that is unexpected and inexplicable, has extreme consequences, but, in retrospect, appears to be easy to anticipate and explain. Historically, all swans were known with certainty to be white—until explorers arrived in Australia and found black swans by the thousands. It took only one black swan to disprove the statement that all swans were white. That should remind us that test suites designed to confirm our belief that the program is working should convince us of no such thing. Passing tests are white swans. So, how do we avoid being surprised by the black ones?

When we test, we use models. We model the test space, deciding what is in scope for testing. We map the structure of the program with flowcharts and diagrams. We model coverage in different ways to better discover different kinds of problems. Each model is a representation—a *re-presentation*—of something more complex, so something about the model is very likely to be wrong. Taleb explains where the danger lies: "Models and constructions, those intellectual maps of reality, are not always wrong; they are wrong only in some specific applications. The difficulty is that a) you do not know beforehand (only after the fact) *where* the map will be wrong, and b) the mistakes can lead to severe consequences. These models are like potentially helpful medicines that carry random but very severe side effects." [2]

One antidote is to have many different models and to vary their use. Another antidote is to alternate continuously between focusing and defocusing heuristics. Testing texts often talk about focusing: starting the program from a known, clean state; following straightforward, deterministic use cases; taking precise steps that are traceable to a specific model; being consistent and methodical; producing expected, predicted results; and biasing both our actions and our reporting toward making the test easy to reproduce and easy to debug.

Yet, defocusing has an important role, too. Our real-world clients don't reset their systems after every activity. They aren't usually like us. They present a wide variety of tasks, roles, temperaments, and behaviors. Their complex actions often don't fit our expectations, and they typically don't care how the product fits our specifications or requirements documents. They *do* care about the product's matching their requirements, whether those requirements were documented, implicit, or unanticipated. Often, we don't have sufficient access to the customer to understand what those requirements are from moment to moment. To find the sorts of problems that our users can discover, we often need rich and complex scenarios, long sequences of actions, and tests that are hard to pass rather than easy to justify. As Taleb says, "The last thing that you need to do when you deal with uncertainty is to focus. The 'focus' makes you a sucker; it translates into prediction problems." [3] And, although we can predict confidently that there will be bugs, we can't predict what they'll be. Bugs are by their nature unpredicted and some are unpredictable. If they were predictable, we presumably wouldn't let them happen.

"The turkey's problem is that it has incomplete information about the world, its models aren't sufficient to anticipate the fate that will befall it, and experience gives it all the more reason to feel confident."

We're often advised to model risk in terms of "probability times impact." Yet, our notions of probability, impact, and the function that relates them to risk are also models, so a risk assessment based on them produces an environment favorable to Black Swans. In 1995, Intel used that kind of approach to evaluate the impact of the floating point division bug, both on end-users and on its own business. Because no previous processor erratum had ever caused such a ruckus, Intel's risk model didn't incorporate the emotional and public relations impact of the story. The company was promptly walloped by the Black Swan, a writedown, and a huge drop in its stock value. A caution, though: Thanks to the narrative fallacy—another target in Taleb's sight—it's easy for us to tut-tut in retrospect and say that Intel should have handled the problem differently. We know how the story ends, so we now fool ourselves into believing that we could have done better. Humility is important.

Taleb refers to the probability-times-impact sort of calculation as the "ludic fallacy," the tendency to model reality in terms of games of chance. To illustrate the problem, he introduces two characters, Fat Tony and Dr. John. Each is told that a fair coin has been flipped ninety-nine times and has come up heads each time. Each is asked the odds of tails the next time. Dr. John, a statistician, provides the statistically correct answer: 50 percent, since previous outcomes should have no influence on the next. Fat Tony, a streetwise operator, says to forget the "fair coin" business; the odds of the coin coming up heads again is a near certainty.

Dr. John's problem, says Taleb, is that he's working from a model of reality that is both hypothetical and idealized, and that even with the evidence of a highly improbable event, he's unwilling to explore the world outside his model. The turkey's problem is that it has incomplete information about the world, its models aren't sufficient to anticipate the fate that will befall it, and experience gives it all the more reason to feel confident. This is the problem of induction, believing that the future will be just like the past. The misgivings we feel for the turkey and the smugness we feel over Dr. John should warn us about a confirmatory approach to testing.

We can confirm things about the product using *checks*—individual observations of some aspect of the system, linked to a true-or-false decision rule, that can be done by a machine. Checks, especially in the form of automated unit tests, are very valuable. In support of test-driven development, checks provide rapid feedback to the programmer about the emerging design of the code. Checks aid refactoring by exposing unexpected and undesired changes. Checks expose important problems before they leave the programmer's workspace, and they're fast—enormous numbers of them can be done at machine speed. Yet, we should be wary, because developing and analyzing checks requires real skill in testing and programming. More importantly, confirmatory checks inevitably reflect our models, rather than challenging them. Those aren't new concerns; in 1961, Leeds and Weinberg wrote:

> One of the lessons to be learned ... is that the sheer number of tests performed is of little significance in itself. Too often, the series of tests simply proves how good the computer is at doing the same things with different numbers. As in many instances, we are probably misled here by our experiences with people, whose inherent reliability on repetitive work is at best variable. With a computer program, however, the greater problem is to prove adaptability, something which is not trivial in human functions either. Consequently we must be sure that each test does some work not done by previous tests. To do this, we must struggle to develop a suspicious nature as well as a lively imagination. [4]

So, maybe checking is best left to a machine, rather than to humans acting like one. That frees us to do testing: exploring, discovering, investigating, and learning about the product by using human variability, open expectations, and the capacity to recognize new models and new risks. Remember this: *Explorers* found the black swans. We can't get rid of Black Swans, but maybe we can find some of them and break a few of their eggs.

...

This is my last Test Connection column. I'd like to thank Brian Marick, who almost four years ago gave me the opportunity—along with the freedom and responsibility—to write what I wanted to write about the fascinating subject of software testing. Thank you to the patient editors and staff of *Better Software* magazine. Thanks to James Bach and Jerry Weinberg who provided invaluable reviews of the columns of which I'm most proud. Above all, thanks to Lee Copeland, whose wisdom and decency inspire me deeply. **{end}**

Another of Taleb's ideas in *The Black Swan* is to consider lowering the impact of unexpected risks. How do you do that in your organization?

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

Sticky Notes

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

- References