December 2008 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

PUT ON YOUR THINKING CAP
A fresh look at test improvement

2008 SALARY SURVEY RESULTS INSIDE!

What's A Manager TO Do?

FINDING YOUR PLACE ON A SELF-ORGANIZING TEAM

A Map by Any Other Name

by Michael Bolton

ISTOCKPHOTO

Through most of the '90s, I worked for Quarterdeck, a company that made memory management software for PCs. Memory management tools were important in those days because programs had been developed for the MS-DOS operating system, which was in turn developed for a processor that provided only one megabyte of address space. When more powerful processors appeared on the scene, they provided access to vastly more memory, but DOS programs couldn't get at extra memory without some fairly sophisticated trickery. One of the approaches was based on *mapping*—making physical memory from far outside DOS's address space appear in a window that was inside DOS's address space.

For a long time, I found the concept difficult to understand. I knew something about mapping in cartography, and I knew that a mathematical function is sometimes referred to as a mapping, but I'd never heard the word used to describe making something appear somewhere else. Things got easier to understand when I considered mapping more generally and realized that maps are links to an idea and a representation—literally, a "re-presentation" of the idea. So a mapping, in a general sense, can take the form of charts, graphs, drawings, or diagrams but might also appear as tables or lists.

When we talk about test coverage, we might be talking about covering a map that represents the product or some aspect of it—a functional diagram or a process workflow. But we might also decide to cover a map—or more accurately a mapping—that presents testing ideas in a non-graphical form. Here are a few examples:

Use a requirements document as your map. Many test groups translate requirements documents from one form ("The input field shall accept up to twenty characters") into another ("Verify that the input field accepts up to twenty characters"). This is not only a waste of time but also, potentially, a directive toward weak testing. Instead of rewriting the document, try testing directly from it. Identify and prioritize statements in the document, using them to trigger test ideas. Checkmark and annotate requirements statements as they're tested, or describe tests in some kind of summary form, pointing to data tables or output files rather than reproducing them. Use the annotated document to guide reviews of testing sessions between a tester and a test manager or project lead. Combine the discussion and the annotated requirements document to check whether test coverage is satisfactory.

Requirements documents and requirements tools are intended to capture and specify someone's intentions for some aspect of the product. These specifications often focus on functional attributes and sometimes pay less attention to the parafunctional (some say non-functional) aspects. While it's likely that much has been learned or changed since the requirements were first identified, in my experience it's somewhat less likely that the document or tool has been updated to reflect the new information. Requirements-based test planning may guide the tester toward a heavily confirmatory approach, rather than a more investigative one. So don't let your test coverage stop here; diversify and use other approaches, too.

Create a map directly from the user interface. While interacting with the product, develop a mind map or a list in hierarchical outline form of the menu and submenu options, dialogs, wizards, buttons, context menus, and other interface options that the product appears to provide. Annotate or detail your map with descriptions of how you tested each item. This approach details coverage for the options that are apparently available to the end-user, but it may be weak in terms of low-level functionality, data integrity, long-term reliability, and so on. It also fails to account for functions that may be available or necessary but not immediately visible. Diversifying your coverage ideas will mitigate the risk of missing something important in the UI. When I miss something using this approach, I ask myself whether I need to explore more methodically or whether features are buried where end-users might not find them, either.

Map the risks. Use review and brainstorming to identify important plausible risks in the product and optionally list specific test ideas. Then perform tests

designed to expose the problems that you've anticipated, checking off existing risks or tests, while keeping your eyes and mind open to new risk ideas. This approach can help drive coverage toward problems that matter. Our own hypotheses about risk are valuable, but we're all likely to be limited and constrained by our biases, so work with other people and use bug taxonomies or cheat sheets (see below) to generate fresh ideas. Ongoing testing may suggest that we're well-defended against certain risks and highly exposed to others, so revisit, review, and revise the risk list frequently to identify what's been covered and what hasn't.

Completely cover some defined corner of the domain space. Doug Hoffman's story of the integer square root function on the MASPAR processor is a case in point. He considered a number of coverage models to reduce the number of tests to run, and then it occurred to him: Why not try all 4,294,967,296 possible integer inputs? Using automation, he prepared a test that covered the entire input domain for that particular function in a few minutes. This wasn't complete test coverage for the whole processor, nor even for all of the possible risks for that function (like stress or flow or performance problems), but he did cover the entire map of its input values.

Map operational models, use cases, or tasks. Use cases or business process workflows can be useful in identifying places where we need to test. Whether you're provided with a list or develop one yourself, you can devise tests to cover the list. On the other hand, it's important to question use cases. I've seen a lot that are very tidy and heavily idealized, but I've never seen one that describes how people actually work in practice. Things are rarely as messy in a use case as they are in the real world.

Use a set of heuristic guidewords or test ideas. James Bach's Heuristic Test Strategy Model, Elisabeth Hendrickson's Test Heuristics Cheat Sheet, and Michael Hunter's You Are Not Done Yet models are all excellent checklists for guiding exploration of some aspect of some model of the program. Use these or develop your own models. Vary the product elements you look at, the quality criteria you look for, and the test techniques you perform. Plot tests against coverage ideas as you …

Work from a test matrix. Prepare a spreadsheet. Scribble a list of test ideas down the y-axis and list aspects of some test coverage model—product elements, quality criteria, platforms, test techniques—across the x-axis. As an experiment, create multiple sheets using the same tests, but with a different coverage model on each sheet, and observe how a single test can provide coverage in a number of different dimensions. For a given coverage model, denser coverage of the matrix suggests (but does not prove) deeper coverage of the particular set of ideas on that table.

Quantitative measures of coverage can be troublesome because they are so easily subject to reification error—treating conceptual things like test cases or requirements statements as though they were units instead of containers for ideas. When we apply models, though, we begin to enter a qualitative world. Qualitative evaluation of coverage can be troublesome, too, because quality—"value to some person," in Jerry Weinberg's definition—is subjective, indefinite, and uncountable. But test completeness is always a subjective and arbitrary concept. Any map tells you something you might want to know, but no map can tell you everything. So we need to develop diversified sets of ideas about coverage and how we map it. Then we check, explore, and compare them to confirm what we believe we know, to guide discovery of what we don't, and to help tell the story of where we've been and what we've done. **{end}**

What do your maps look like? How do you describe coverage to clients and your project community?

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