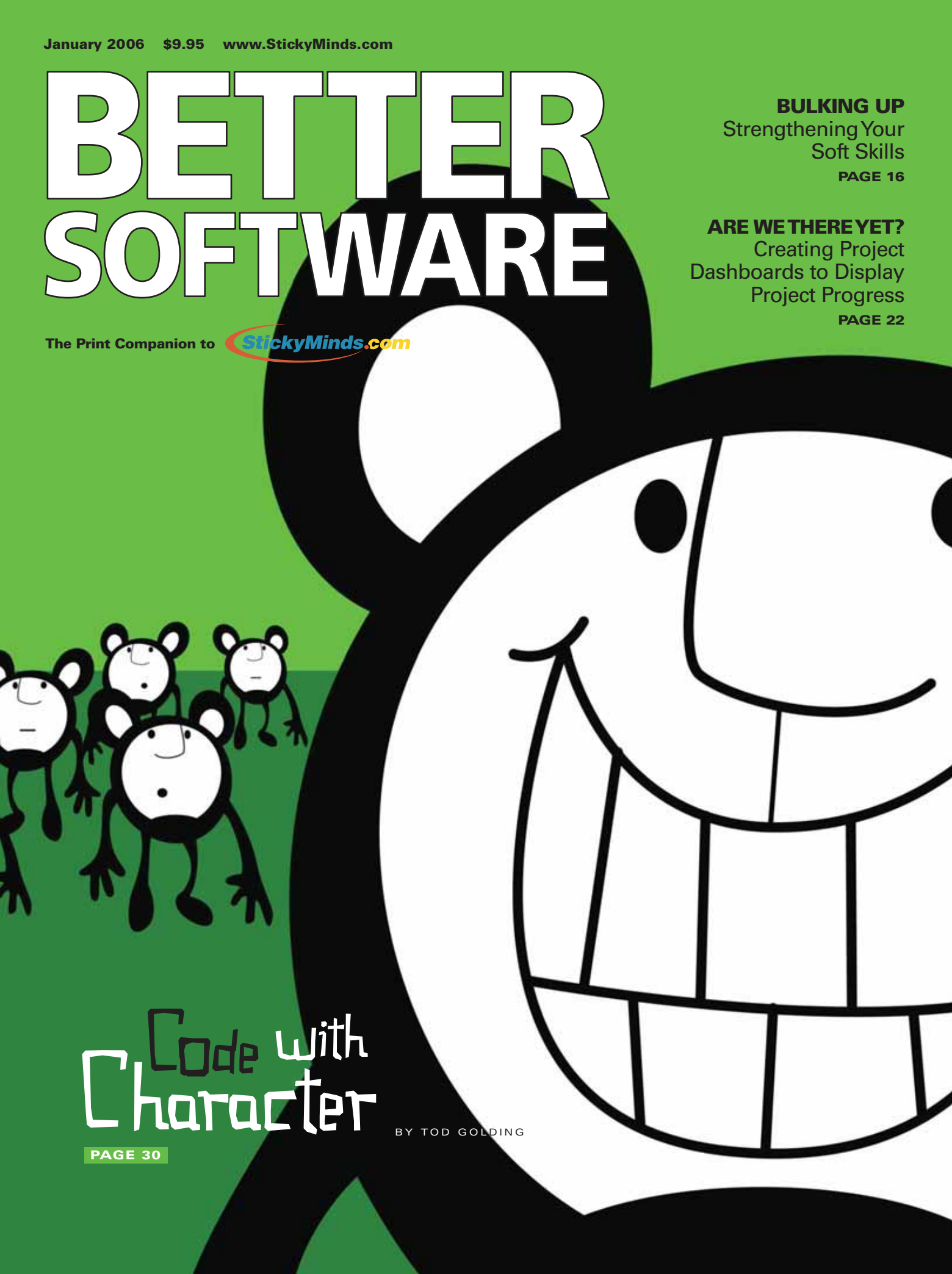
January 2006 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

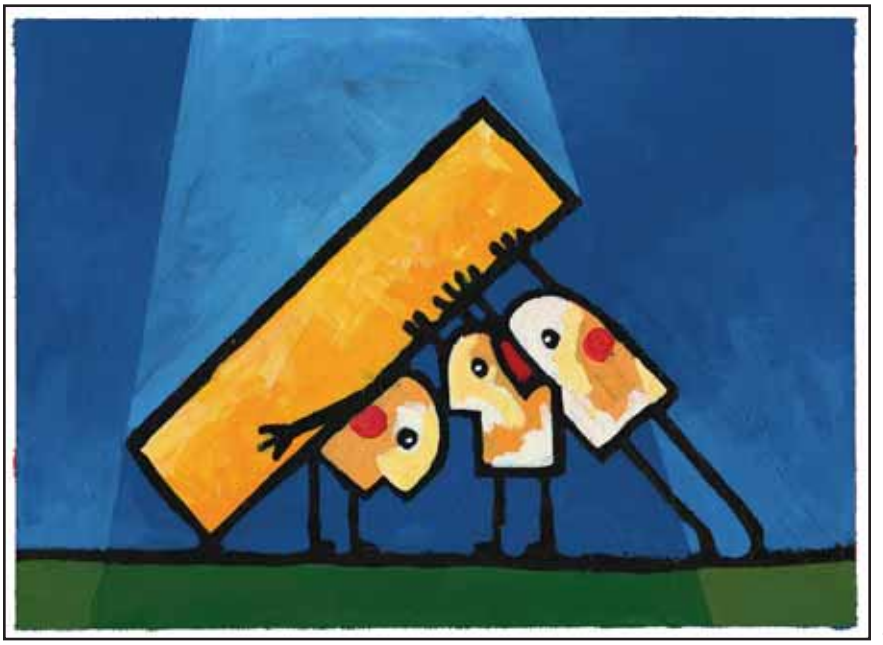


Code with Character

BY TOD GOLDING

Support for Testing, Testing for Support

by Michael Bolton



Getty Images

In the last issue, I introduced the first part of the Quality Criteria dimension of James Bach's Heuristic Test Strategy Model. People often refer to these quality attributes as "the -ilities," properties of the product that customers might find desirable: capability, reliability (which under the HTSM includes security), usability, scalability, performance, installability, and compatibility.

The second part of the Quality Criteria list focuses on development, or producer-facing attributes. The HTSM identifies supportability, testability, maintenance, portability, and localizability. The first two of these attributes are so important that I'm going to dedicate this entire column to them.

Rapid Testers use Jerry Weinberg's definition of quality: "Quality is value to some person." The end-user is just one member of the project community whose values matter. We also attempt to produce value and reduce cost for the organization that is developing the software. We think about supportability and testability to remind us to look for problems that have a real impact on support people and the testers themselves; they are also customers of the testing effort. Both groups can and should ask for supportability and testability.

In the early '90s, I worked for a company called Quarterdeck. Its flagship products were DESQview and QEMM-386, multitasking and memory-management utilities for DOS running on Intel-based personal computers. The PC environment in those days was a mishmash of mostly compatible hardware, but because our products worked so closely to the metal, they were more vulnerable to compatibility problems than most other products.

I worked in technical support, then testing, and later in program management. Those departments were closely linked—everyone used the products in his daily work, so everyone tested to some degree. Ever since then I've been aware that good technical support people are natural allies for testers and can be highly valuable to the testing effort. They're experts with the product, they're direct conduits to the customers, and they're keenly aware of the kinds of problems that make telephones ring and support forums choke. As testers, we want to prevent those things. We reduce cost and add value when we find bugs or anything else that would cause problems in the field or extra work for the support staff.

Cultivate relationships with your support people. They can help you understand what supportability means in your context and what's important to customers. Support people may identify risks that you may not have considered, and they can help estimate the cost and impact of a bug. Most of them will be delighted to help you find problems and will advocate specific bug fixes. Support people may have access to tools, tricks, or tips that testers can use. Swap useful documents, diagrams, or scripts. Make sure that the support people have your number so they can call you directly if they see a problem in the product. Some support staff may aspire to become testers, in which case you have a farm team in-house.

A support person will be the first to tell you that coherent and consistent error messages make a big difference to a program's supportability. Testers should try to trigger all kinds of exceptional conditions while testing. We do that primarily to expose risk; unforeseen conditions that the program doesn't handle put the program in an unpredictable state. But even if we don't find bugs, we look closely at each error message and other feedback supplied by the program. Does the message clearly and accurately describe what's going on? Does it help the end-user solve the problem—or if that's not feasible, would the message assist support staff or developers? Does the error message uniquely identify the point of failure in the program? If the problem is a missing file or resource, does the error message *specifically* identify what's missing? A support person will identify a problematic error message for you right away ("A .DLL could not be found."—OK, but *which* .DLL?).

Many of the attributes that add to a product's supportability also add to its testability. By testability, Rapid Testers primarily mean *visibility* and *controllability*. We evaluate the program based on the means it provides for service people to support it, testers to test it, and developers to debug it. Does the program produce log files? Can we configure logging to provide varying levels of detail? Are the logs consistently structured? Can they be scanned quickly and easily, either by a human or by a program written in a scripting language? Is each event that is recorded in the file precisely time stamped and well structured so we can write scripts that parse, filter, and summarize? Log files are a feature of the product—are we testing and evaluating them, or are we taking them for granted and thereby possibly missing bugs? What other reports does the program produce? Is there information in them that might help in the testing or support effort? What information is missing that we would like to be able to see? Can we query the program on the fly?

We'd like to be able to automate certain functions of the program or our interaction with it, so we ask for controllability—scriptable interfaces, typically using languages like Perl, Python, or Ruby, to reduce the need for expensive front-end tools. When the program provides a means to control it, we can use automation to operate the program, to set up data and manipulate it, to probe the state of the product, or to install and configure it. Programs that can be controlled remotely might add to testability and to supportability if they can reveal useful information. The emphasis is on doing things efficiently—getting the machine to do the work—which leaves us more time for critical thinking, observation, and evaluation.

Note that when Rapid Testers talk about testability, they're referring to visibility and controllability of a program. In some places, "testability" has another meaning; it is sometimes equated with "falsifiability," or "decidability," which is the ability to make a true-or-false or yes-or-no statement about the program. That's a valuable notion, but there are two pitfalls to avoid. The first is that a falsifiable statement may not tell the whole story of an observation we could make. The statement might be too vague to be testable. For example, "The application shall exhibit responsiveness." Conversely, it might be precise in a way that may not really matter to anyone. For example, if an application, as specified, must return a result within one second, give or take five milliseconds, does that difference matter? It might, but if not, we could be tempted to create overly precise tests that distract us from more important things in the mission.

The second pitfall is that jargon words like "testability," "stress testing," or "functional testing" may take on different meanings in different organizations. Someone who claims that a program is testable (meaning falsifiable) may not understand when we assert that the program is not testable (meaning visible or controllable). We can choose to use whatever words are culturally feasible to ask for visibility and controllability, as long as we clearly express that we need them. Moreover, if we consider both possible meanings of testability, we spark our imaginations to create more diverse tests.

To get visibility and controllability, we may need to recruit developers to our cause, but there's something in it for them, too. The developers themselves benefit because a more testable program is almost always easier to debug. They may appreciate that, when a program is more testable, we testers need less time to achieve the same amount of test coverage, or we can achieve more coverage in the same amount of time. Either way, we have a better shot at discovering some problem that threatens the value of the product.

Testability fosters collaboration. At Quarterdeck, memory management and multitasking were tricky to understand and diagnose. To resolve problems and test the product, the development, support, and testing teams needed access to information about the system, DESQview, and QEMM, so the developers wrote a program called Manifest and included it in the package. They continued to refine the product based on ideas from support staff, testers, and customers.

Manifest added a lot of value by making invisible things visible. It allowed our support staff and testers to be more productive by allowing them to troubleshoot problems quickly. Manifest had easily understandable maps of memory that showed which program or device was using which addresses, so finding and resolving conflicts was a breeze. Manifest collected all of the relevant system information in simple tables that were easy to navigate, clearly presented, informative, and able to be printed, mailed, or faxed. Some customers and vendors came to use Manifest as a general troubleshooting tool. For other customers, the DESQview and QEMM packages were more valuable than their competitors, at least partly because they were better tested and more supportable. Did that make a difference? Well, for some time, QEMM was consistently the best-selling PC software package in the world. Testability and supportability count. **{end}**

To get visibility and controllability, we may need to recruit developers to our cause.

Michael Bolton lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries as part of James Bach's Rapid Software Testing course. He is program chair for the Toronto Association of System and Software Quality and is a regular columnist for Better Software *magazine. You can contact Michael at mb@developsense.com.*

Don't Stop Now!

Log on to **StickyMinds.com** and join Michael Bolton and your peers in a conversation about this topic. At the end of the digital column, add your views or just read what others have to say.