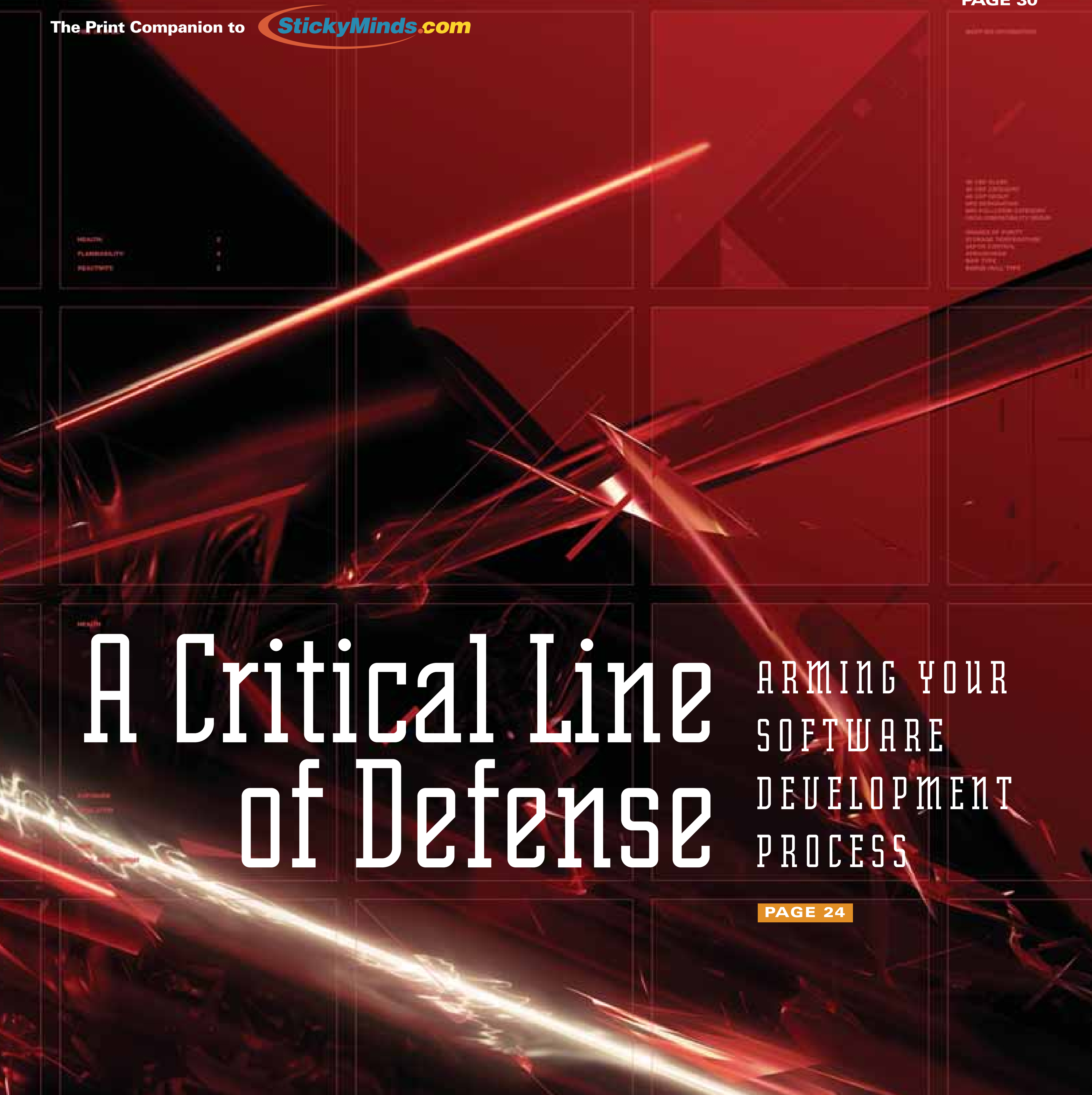
February 2006 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

THE BEST LAID PLANS...
Let principles be your guide
PAGE 6

AGILE SCM?
Oh yeah, it's possible
PAGE 30



A Critical Line of Defense

ARMING YOUR SOFTWARE DEVELOPMENT PROCESS

PAGE 24

Maintaining Your Course

by Michael Bolton

GETTY IMAGES

Maintainability is an attribute that, in my experience, doesn't receive a lot of attention from testers. In general, maintainability is about a product's ability to stay the same or to adapt to change. For Rapid Testers, maintainability is the guideword heuristic that reminds us of the need for retesting and the need to be able to reuse testing ideas.

Things don't stay static in our world. The business environment changes, the product changes as new development happens, the people around the product change—not only the developers and the testers, but also the users, the buyers, and the entire project community. Some of our test strategy should focus on continuity. Parts of the product need to be insulated from the change that's happening around them. We want change to happen consciously, not inadvertently. To a great degree, that's what confirmatory regression tests are for—to make sure that the things that used to work still work. But a complementary part of our testing strategy should be to note the costs, values, and risks of making changes to the product, and to be aware of the impact on the product and on the test effort.

In order to generate ideas that broaden and deepen our test coverage and strategy, Rapid Testers try to extend conventional notions beyond their traditional senses. When people talk to me about maintainability on a project, they refer to things like configuration management, code readability, documentation, or object orientation. But surely there's much more to maintainability of a product than that. We might have the parts and the technical manuals that we need to repair a car, but lack the equipment, the expertise, and the time to do it. What information might we need to maintain some aspect of a software product? What skills, tools, or resources might someone require?

We could think about a product's maintainability in terms of the code, but what other aspects of the product might change? File format, configuration information, platform, user interface, documentation—name any component of any model of the product and that component will stay the same, evolve in some way, or disappear.

Questions about risks associated with change and maintainability help me focus on the areas of the product I need to test and retest:

- In the face of change, what existing behavior must I confirm?
- What must I investigate anew?
- What *matters*?
- What will the needs be—retesting the whole system or targeted retesting?
- What parts of the product might be brittle or hard to test?

Advocates of agile processes tout automated tests to provide rapid feedback when change happens, and indeed automated tests are potentially useful and powerful. Here, it's useful and important to make alliances with developers. Something that is more testable, through scriptable interfaces and logging, is more *re*testable and ultimately more maintainable. However, it could be easy to fall into the trap of believing that unit tests and regression tests will help us spot problems reliably after modifications have been introduced. If we have a big regression suite, part of the maintenance effort should be to revisit the tests regularly and critically. Are they still likely to reveal important problems? Are there new risks that they're missing? Have we selected test tools that will be compatible with the product's target platforms?

It would be nice if we were always working on new projects with new code, but many testers come to projects that are several major versions old—even new software spends a lot of time interacting with very old software. Maintenance may be done in a hurry by people who haven't had the experiences of the original team.

Writing automated unit tests for every change is the prescribed agile approach. Automated test development is a strong asset for maintainability, but it isn't free; writing an automated test can often take a good deal more time than a fast manual test. So for each test idea ask, "Is the risk of some failure sufficiently low that I can perform a very quick, trivial test right now, one that I probably won't need to repeat? Would it be OK simply to add this risk to the bottom of the risk catalog? Or should I write a script that can throw a lot of data at some program, or that can save a lot of time if we end up using it often?" Heavily scripted tests for an application that is in flux will require a lot of maintenance effort. Maintaining test scripts takes time away from testing—actively operating and observing the product.

Maintainability influences the way I learn about the project, which in turn influences the way I think about the product's documentation. In most projects I've worked on, documentation starts with a lot of energy and enthusiasm—and volume—and becomes less and less useful as it falls out of sync with the product. More extensive documentation requires more effort to maintain, so one approach for Rapid Testers is to bias ourselves toward less documentation rather than more. Don't try to write down everything at the beginning, when we know the least about the task at hand. Test, learn, and iterate. If we need more documentation, we can add it where it's needed over the course of the project. Instead of imagining what *might* be done and writing detailed, step-by-step test scripts, think about cataloging risks and test ideas, then let tester skill and creativity, combined with good note taking, record what *was* done. Exploratory testing, if done in an unstructured way, could lead to an unmaintainable test effort; consider Session Based Test Management (see the StickyNotes) as an antidote for information loss.

Rapid Testers shouldn't escape their own scrutiny. I use maintainability to help me think critically about where I'm spending time and the potential return on investment. I try to think about how I take notes on what I'm doing or learning:

- Will a sketch, a few rough notes, or a concept map help me remember and stay organized?
- Am I absorbing and recording information mostly for myself, or will I need to present my work to other people?
- Are their needs served better by conversations supplemented with my notes, or by detailed, formal reports?
- What information do people need immediately?
- What will they need in the longer term?
- What tools am I using for specific tests?
- Will people need to use my scripts again, or are they one-off experiments?
- When I'm testing or developing something, what are the odds that I'm going to deal with this artifact again?
- Is the work that I'm doing right now for me, or is it for someone else?
- How might my approach change if my assumptions aren't borne out?

If you're in a position to comment about the maintainability of the overall product, beware of asserting simply that the product is not maintainable. Maintainability is a compared-to-what question. *All* products are ultimately maintainable, because doing nothing at all is one option; making changes at some cost and at some risk is the alternative. Identify those costs and risks, and compare them with the value that the changes bring. **{end}**

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. Michael is a regular contributor to Better Software *magazine and program chair for the Toronto Association of System and Software Quality. Contact Michael at mb@developsense.com.*

Sticky Notes

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

- Session Based Test Management

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.

A Rapid Testing Exercise:

Try performing an instant analysis. Choose some specific aspect of the product or its universe—consider the project environment, the product elements, and the other quality criteria in the Heuristic Test Strategy Model for ideas. Start by asking, "Suppose that this thing stays the same as things around it change—what do we need to do to ensure that stuff still works?" Then ask, "What if this thing *changes*? What else might be affected by that change? What risks would emerge? What are our oracles—what principles or mechanisms could we use to recognize a problem? How would our coverage models change? What test activities might we add, drop, or refine?"

After you've gone through the exercise yourself, bounce your ideas off a colleague. Identifying potential risks is a skill that improves with practice, but it's also important to remember that risk involves prediction and projection. We will get some things wrong. That's OK. As Niels Bohr said, "Prediction is very difficult, especially about the future." It's nice to be prescient, but we're not omniscient.