

**INFO TO GO**

- No matter how long you spend in the testing field, you'll still have more to learn.
- Study public databases, your own code, and supporting technologies to learn more about bugs—how they are born and how they can be overcome.

# The Care and Feeding of Testing Skills

by Elisabeth Hendrickson

JILL GLANCED AROUND THE CONFERENCE ROOM. "Actually," she confessed, "even though I'm here to interview for the test position, I really want a job in development." She gave Thomas a conspiratorial smile. "I've been testing for two years," she explained, "and I think I've pretty much done it all at this point. I'm ready to move on."

Thomas, who had been testing for three times as many years and felt as if he'd just scratched the surface, was taken aback. *Hmmm*, he thought. *Let's see how good she is*. Standing at an empty whiteboard, Thomas sketched a dialog box with four fields and a button. He labeled the fields and then stepped back. "Here's a form from a fictional system. Given what you see, what tests would you want to run?" Thomas handed her the pen in case she wanted to draw her ideas on the board.

"Well," Jill began without moving from her seat, "I'd try some boundary conditions, some negative tests, a use case or two, and I'd be done." She looked very satisfied with her answer.

Thomas smiled. "And you'd be done?"

Jill frowned as she thought for a moment, and then smiled in return. "Yup. I'd be done."

Thomas's smile dissolved. "Jill, most of the candidates we have through here would have filled that whiteboard with tests. You've barely begun to explore the testing problem I posed to you." Thomas paused to let his criticism sink in. "Want to try again?"

Jill came up with a few more tests, but not many. She had no idea what Thomas was looking for, and she didn't particularly appreciate his attitude. *Testing is easy*, she thought. *What's the big deal?*

The big deal isn't how many tests Jill surfaced as much as her attitude toward testing. She thinks she already knows all there is to know.

Thomas, on the other hand, recognizes that professional growth can take a lifetime. He knows that no matter how long he spends in the testing field, he'll still have more to learn. He takes time to nurture his skills, grow his abilities, and feed his mind. As a result, Thomas will continue to progress in his testing career.

Jill's career stalled after just two short years, while Thomas is continuing to grow professionally and personally. How can you assure your continued professional growth? Here are some ideas.

## Avoid Complacency

Jill was quite satisfied with her answer to Thomas's testing problem. She felt that she'd covered everything. But since testing is an infinite problem, no answer would be complete. Jill had fallen into the trap of complacency. Testers like Thomas know that no matter how many clever tests they've already thought up, there are more tests they could be running.

## Study Bugs

Thomas has more than a passing curiosity about bugs. He seeks to understand them: Why did the software behave that way? How can I search out related bugs in other areas? What was the root cause?

It's good if you study the bugs that have already been found in the software you're testing. It's even better if you study other people's bugs, too.

The next time you get together with a group of testers, swap bug stories. Conferences or local quality group meetings provide an ideal forum for a friendly chat about bugs with other testers working on different kinds of software.

Also take a look at public bug databases. Microsoft has an extensive collection of bugs in their knowledge base. The information in the knowledge base is especially useful to anyone working with Microsoft technology. Another source of public bugs is the Mozilla browser project. Reading Bugzilla entries can give you ideas for testing browsers and Web applications, as well as ideas about what can go wrong with any kind of desktop software. (For links to public bug databases, see this issue's StickyNotes at [www.stqemagazine.com](http://www.stqemagazine.com).)

## Make Your Own Bugs

I know of no better way to appreciate where software bugs come from than to write your own code. Whether you're a beginning programmer or an expert, write programs and you'll create bugs. (After all, the difference between a neophyte programmer and an expert is that experts tend to create more complex bugs that are harder to track down.)

When you discover that you've created a bug, study it. Dissect it. Understand its genus and species. If you could create that particular bug, so could another programmer. When you under-

stand what happened in your own code, you'll have a better idea how to find that kind of problem in other people's code.

You'll learn a lot about bugs, no matter what language you choose for writing your programs. However, if you can write programs in the development environment your programmers are using, you'll find the experience even more valuable. You'll gain a greater appreciation for the trials and tribulations your programmers face, as well as an understanding of where bugs might lurk in their code.

### Study Supporting Technologies

All software suffers from *duct tape problems*—incompatibilities that arise as a result of integrating various technologies. The software you're testing is no exception. Does your software rely on the .NET framework? Relational databases on the back end? Web servers? SOAP? XML? What about the operating system? Whatever technology your software uses, you're more likely to find duct tape problems if you understand something about the ins and outs of the underlying third-party technology.

While you can learn about possible duct tape problems by writing programs, you can also learn about them by reading articles and talking with developers and other testers. Seek out information about weaknesses in the integrated technologies: security vulnerabilities, reliability problems, or performance issues. Then try to apply that new knowledge to testing your software. You can do this by finding instances where the inherent weaknesses in the underlying technologies are caus-

ing problems with the software you're testing.

### Learn the Business

Given that there are an infinite number of tests you could run, how do you decide which are the most important? Understanding potential business risks can help you prioritize your testing. What do your customers say about the company? What are your competitors doing? How does your software give the company a competitive advantage? Are there particular types of failures that would be especially devastating? What does the press say about your company or product? What do the industry analysts say?

You can learn more about the business by reading industry journals and your company's annual reports, and by talking with the business folks. If your company holds quarterly or annual meetings to discuss financial results, attend. (Yes, I know these kinds of meetings are often long and boring, but buried in the stultifying speeches is critical information you need.) Invite your local business analyst, marketing representative, manager, or executive to lunch and pick his brain. Find out what challenges the business is facing and how the software will help the company meet those challenges.

Then take the business knowledge you've gathered back to your desk and figure out how to apply it. What risks could you expose through testing that would be most beneficial for the business decision-makers to comprehend? The more you understand about the business context, the more you can ensure you're chasing after the right information.

Since you're reading *STQE*, you already know how important it is to read

about testing. The best testers I know read voraciously about testing and also about a host of other subjects: programming, project management, business strategy, marketing, psychology, history, anthropology—you name it! Testing is a broad, sweeping topic, and ideas can come from anywhere.

### Keep a Journal

As you learn more about testing, use a journal to record your discoveries. I have work journals in my office dating back more than fifteen years. These journals have served to track my thinking at different points in my career, reminding me of lessons learned and inspiring me to try new things. My journals also help me spot patterns in my thinking as well as synthesize new ideas. I use my journals to chronicle my test efforts and clarify my analysis of software.

If you don't already have a journal, start one today. You don't need anything fancy; a cheap spiral-bound notebook or a text file on your computer will do.

Your first entry could be your thoughts on how to continue the care and feeding of your testing skills. **STQE**

---

*Elisabeth Hendrickson (esh@qualitytree.com) is an independent consultant specializing in software quality assurance and management, with fifteen years of experience working with leading software companies. You can read more about her ideas on quality and testing at [www.qualitytree.com](http://www.qualitytree.com).*

**STQE magazine is produced by  
Software Quality Engineering.**