informIT

How Do You Practice Software Testing?

Date: Aug 12, 2005 By Michael Kelly.

Many of us can play a simple tune on the piano. If we want to actually play the piano, though, that takes practice. Mike Kelly shows some simple techniques that can help you to get away from being a "one-tune tester" to developing real testing skills through practice.

Dr. Richard Restak devotes a large section of his book *The New Brain: How the Modern Age Is Rewiring Your Mind* (Rodale Books, 2003) to the topic of practice. When talking about a person's true potential, he cites the research of Florida State University psychologist Anders Ericsson. Ericsson, studying "superior performers" in various fields, has found that the key ingredient to superior performance turns out to be one's willingness to "stretch yourself to the limit and increase your control over your performance."

That means *practice*.

This article provides a list of possible ways in which you might practice your software testing. I've included references or examples where I think they might be helpful. This is by no means an exhaustive list; it's some ways in which I or the people who reviewed this article practice. If you practice in some way that's not on this list, I strongly encourage you to add a comment at the end of this article, sharing that experience and some resources to help others get started.

Focusing Your Practice

Restak writes, "[For superior performers,] the goal isn't just repeating the same thing again and again but achieving higher levels of control over every aspect of their performance. That's why they don't find practice boring. Each practice session, they are working on doing something better than they did the last time."

For example, a musician doesn't play a scale repeatedly just for the sake of playing the scale. When I repeat a scale while playing the guitar, it's not so I'll learn the scale; I know the scale. It's so I can get my *fingers* to know the scale. I want them to move faster and with more confidence. I'm attempting to achieve a higher level of control over my performance. If I can better develop my fingering technique on that scale, I can better control my fingering in other aspects of my playing.

Each time you practice testing, you should be interested in doing some specific thing better. By improving one specific technique at a time, you gradually improve your overall ability over time. In music, I might focus on a specific song for an hour, or a specific type of music (jazz, rock, ska...). The focus of that practice is not the repetition of that specific song or style of music; the focus is on improving a specific aspect of what I'm practicing (speed, technique, experimenting with pedals or amplifiers, and so on). A generic goal of practicing just to "play better" isn't practical. To be more effective in my practice, I need to focus on one specific thing and do that thing better.

Avoiding Automated Performance

Restak also writes, "In order to achieve superior performance in a chosen field, the expert must counteract the natural impulse to gain an automated performance as soon as possible." When I first started learning guitar, I wanted to play songs—specific songs. In addition, I wanted to play them well. I would practice a song here or a song there, but always the same set of songs and always in the same style. Over time, I played those songs rather well. Had I continued down that path, my guess is that I could have mastered those songs—providing an automated performance.

One day I wanted to play something else. I tried and I failed. I couldn't bring my mind and fingers to play a different type of music or a different song. To do so, I would have had to start over from the beginning and repeat the whole process for the new song. I was always attempting to *automate* my performance. But what I needed to learn was *technique*, not automation—so that, over time, it would only take me minutes, not days, to be able to learn a song. I had focused too much on automation and not enough on superior performance.

How much of your testing is like this? How easily do you fall into automated performance with what you do every day? Practice can build new thought patterns—and can also reinforce existing thought patterns. By doing something over and over or repeatedly thinking about something in a specific way, you actually change the way your mind works. Remember that the goal of practice is to stretch yourself and to increase your control over your performance.

Contributing to Open Source Projects

Contributing to an open source tool is one of the best ways to practice because it most accurately simulates a real project. By *contribute*, I mean that you can submit code, test code, write documentation and examples, or just be active on a mailing list and answer questions for people getting started. All of these activities will help some aspect of your testing because all of them expose you to different *real* problems on a software project.

Open source projects are good because you can contribute on your own time—or, if you're one of the lucky, while you're at work (a great example is the work that Atomic Object did with Systir). If you just can't seem to find the time or energy to contribute, you can also practice by downloading and using some open source tools. A good place to find tools to practice with is OpenSourceTesting.org. At the time of writing, they listed 207 tools.

Beta Testing

A favorite strategy of Antony Marcano and Neil McCarthy is to get involved in a beta testing program. If you use commercial tools (IBM Rational, Mercury, Microsoft, etc.), let your local representative know that you would like to be involved in their beta test program. This opportunity gets you using the latest and greatest software and potentially exposes you to new types of bugs (hopefully in a way that improves the way you test *your* products). In addition, you sometimes get to be an active participant in providing feedback that sets the direction for the tool you're testing.

Pair Testing/Programming

Pair testing/programming doesn't need to be limited to code and bugs. Find people you think you can learn from and with, and get working together on problems. Keep in mind the following suggestions:

- If you tend to practice better with a buddy, set up a time at which you can work on pair programming and/or pair testing together. Be sure to practice with *both* a tester and a developer.
- If you do pair programming or testing on a regular basis, try some different pairings.
- If you're a tester, spend some time working with developers to learn how they unit test. Write some unit test code, either for their code or for any test code you may have. Over time, this strategy will allow you to better collaborate when they're planning their developer testing.

A good practice tool that I've used is NLP for testers. NLP is a modeling technique (documented extensively by Alan Richardson) that can be used to question requirements and design to identify ambiguity, errors, and omissions. I've found it helpful to pair with requirements analysts, developers, and testers and apply the NLP Meta-Model to their work products.

Adopt "Parallel Thinking"

Another effective method of practice might be to take a problem from work and attempt to analyze it from different perspectives. For example, if you applied the four schools of testing as identified by Bret Pettichord and evaluated your problem through the prism of each school, you would most likely develop a deeper understanding of the problem. Each school's relative merits might force you to think in unusual ways. How do the core beliefs for each school change the way you view the software and your testing? What types of tests would you perform in that school? How would you interact with other team members? The idea is similar to the process outlined in Edward de Bono's *Six Thinking Hats* (Back Bay Books, 1999), a book on parallel thinking.

Search for Bugs in the Wild

My favorite method of practicing is finding bugs in the wild. Some people have a built-in ability for this approach; for example, read Tim Van Tongeren's story of how a bug harassed him at home by calling him each week. For another example, read about how James Bach tests software while at Borders or his experience testing airport kiosks. One of my richest experiences in usability testing was when I tested a movie ticket kiosk. I'm sure the software you use every day—and I'm not just talking about vendor test tools and Microsoft products—is full of bugs. Challenge yourself to find bugs in other systems you interact with every day. Extra credit if you find a bug in a non-software system.

If you find a bug in the wild, share it with the world. Start a blog on bugs you find and how you found them. I recommend TestingReflections.com as an excellent place to start a blog for free. If you don't have that kind of voice, join a forum or mailing list where you can share what you find. Part of the practice built into finding bugs is reporting them and recognizing *how* you found them. Did you just stumble on the bug or did you apply some skill or technique? Reporting the bug forces you to identify what you learned. Bugs are out there just asking you to find them (in Tim's case, they were looking for *him*).

Learn "Systems Thinking"

If finding bugs in the wild isn't your thing, try practicing *systems thinking*. I encourage you to get a copy of Jerry Weinberg's Quality Software Management Volume 1: Systems Thinking. It's not too heavy a read, and it points to more material on the topic if you really get into it.

Systems are everywhere, just waiting for you to identify them. If you can train your mind to quickly identify different systems and to notice interactions within the systems, you'll be more effective as a tester. At a recent workshop, James Bach related a story of how, while standing on a street corner waiting to meet someone, he spent time identifying the different systems around him. You can practice by identifying systems while waiting in traffic, walking through malls, sitting in restaurants, and certainly at home and in your relationships.

Teaching and Writing

Another possible source of practice is teaching others and writing about your experiences. There are several online places for testers to publish, as well as some good testing magazines (check out print publications Better Software, Software Test & Performance, and Professional Tester). If your group includes junior testers, schedule some time to train them. If you really know something in depth about testing, or have an interesting experience to share, present the topic at a local user group or at a conference. By training others, not only do they benefit, but *you* benefit through shared experience, asking and answering questions, and the reinforcement of the material that comes with speaking about it.

Participating in Conferences and Workshops

For the community-minded, a lot of good practicing can happen at conferences and workshops. A very large software testing community hosts and encourages participation in focused workshops on software testing. If you need to energize your testing and energize your practice, take some time to attend one of these events. Instead of practicing a little bit each day or week, conferences and workshops give you high-intensity practice sessions lasting days, not minutes or hours.

Develop Your Cognition Skills

Whenever I get the chance, I try to work on my cognitive abilities. Cognitive abilities allow us to process the sensory information we collect, giving us the ability to analyze, evaluate, retain information, recall experiences, make comparisons, and determine action. By practicing to increase your cognitive ability, you'll be better prepared for all the other tasks you perform as a tester.

In his article "Inside the Mind of an Exploratory Tester," James Bach provides a great example of a practice exercise to increase rapid cognition skills. Go to a bookstore, pick a computer book at random, flip through it in five minutes or less, close the book, and ask yourself the following questions:

- What does this technology do?
- Why would anyone care?
- How does it work?
- What's an example of it in action?

Repeat the process until you can answer the questions.

Exercises like this are excellent for increasing your ability to find and learn the information you need when faced with new problems.

Finding the Time for Practice

On most projects, it's a heads-down race to the end of the project: too little time, too few people, too much work. After one iteration, or at the end of a project, it's a dash to the next one. It's difficult to find the time to practice when we have so much work to do. After work, life takes over, and other promises and obligations keep you from devoting the time that you know your craft needs.

The trick is to find ways to practice while still getting your work done and while living everyday life. You need to make sure that you aren't just practicing for automated performance, but for specific improvements. Identify where your testing may be weak and think of a series of practice sessions that might help improve that aspect. Do you need to become more technical? Do you need to brush up on your black box testing techniques? Or do you simply need to step back from test management and actually get your hands dirty again? Identify what you want to improve and focus on doing that better.

If you already have a regular practice routine or if you start to develop one as a result of this article, take some time to post a comment at the end of this article and share your experience. If you find some bugs in the wild, post them (or links to them). By learning how others practice, we can better build a practice toolbox, allowing us to tune our practice sessions to better fit our needs.

800 East 96th Street Indianapolis, Indiana 46240