Teaching the Tricks of the Trade

Jon Bentley

AT&T Bell Laboratories
Murray Hill, NJ 07974

Introduction

Here's a trick of the medical trade useful for anyone who donates blood. Before sticking the big needle in your arm, the nurse first pricks your finger for a few drops of blood. Some thoughtless nurses jab the pad of the index finger, which is the most sensitive spot of the most used finger. It is better to poke a less sensitive part (on the side, halfway down from nail to pad) of a less commonly used finger (the ring finger). Insist on it the next time you donate blood.

Medicine has a firm basis in science: chemistry gives great insight into pharmacology and biology is essential for surgery. Techniques of project management are also obvious: surgical teams allow a dozen hands to work under one mind, and medical records are often exemplary documentation. But science and management aren't enough; excellent practitioners must also master some tricks of the trade.

The same holds true in our field. All software engineers must be familiar with relevant subdisciplines of computer science and with techniques of project management. But there is more than science and management inside the heads of excellent software engineers. They know the tricks of their trade.

By the term "Tricks of the Trade" I explicitly exclude subjects that might fall under the titles of computer science, project management, or programming techniques. Rather, I wish to emphasize what might be called "common-sense engineering". I have written about such topics in several of my "Programming Pearls" columns in Communications of the ACM.† This paper is about teaching these tricks in software engineering classes. We'll begin by studying one trick in detail, and then survey several others.

† Columns that are specifically devoted to tricks include Cracking the Oyster (August 1983), The Back of the Envelope (March 1984), Graphic Output (June 1984), Tricks of the Trade (February 1985), Bumper-Sticker Computer Science (September 1985), Cutting the Gordian Knot (February 1986), and The Envelope is Back (March 1986). Tricks are used in many of the other columns; see the index entry "engineering techniques" in the collections Bentley [1986, 1988].
Debugging

After we observe weird behavior in a software system, how do we identify the culprit that is causing the problem? That skill is often crucial in building and maintaining a software system, yet is rarely taught in books or classes. It deserves to be taught, but how should we teach it?

Stories can help a lot. In the late 1950's, Vic Vyssotsky was called to repair a program that was raising havoc at Bell Labs. On typical runs, the program worked correctly about half the time. But whenever the programmers came into the machine room, the program always worked — and because they could never see it fail, they couldn’t make any progress in fixing it.

That was the only clue Vyssotsky had: the program worked whenever the programmers were present. Something apparently knew that the programmers were in the room. What could it be? His colleagues generated all sorts of wild theories: did the excess weight on the machine room floor cause it to sag and stretch a cable? Such conjectures are hard to test. Vyssotsky agreed that something did know that the programmers were present, and set out to find an elegant answer. Think hard about the problem for a minute: what was in the machine room that just had to know that the programmers were also there?

People, that’s what. The computer operators could see that the programmers were in the room. With that insight, Vyssotsky was able to observe their behavior unobtrusively. When programmers were present, the operators laboriously followed official policy: mount each tape, check that it is aligned, and carefully hit the rewind button to position it properly. When they weren’t being observed, though, the overworked operators would sometimes hurriedly slap the tape on the drive and let the program do the job with a software rewind command. And that led Vyssotsky to the bug in the program: the rewind instruction in that particular program had been erased (by a distant and particularly nasty bug).

There’s a moral in the story. The expert debugger never forgets that there has to be a logical explanation, no matter how mysterious the behavior may seem at the time. Vyssotsky observed that something had to know that the programmers were in the room, and he conjectured the simplest possible something (which turned out to be a someone) who could know. When your software exhibits weird behavior, think hard to find a simple explanation.

There are lots of great debugging stories. When I asked Bob Martin of Bell Communications Research about the neatest bug he had ever tracked down, he responded immediately with the story of “the program that worked once twice”. He watched a huge system undergoing its first field tests. When the system was booted, it correctly handled one type of query and then reported a string of errors. When the system was rebooted (for an unrelated reason) Martin observed the same behavior again: the first answer was correct, and all subsequent answers gave errors. The program worked once, twice. This was enough of a clue for Martin to tell the programmers where to find the bug: the re-initialization code was