The Incremental Developer

A lot has been said in this forum and elsewhere about quality. Don't expect it to die down much—not until things improve, anyway. I find myself in the camp that believes that quality can improve greatly without noticeably affecting schedules and costs. My own experience has taught me that I can improve my effectiveness if I want to, even in the presence of clueless management. If management somehow managed to see beyond the quarterly bottom line (dream with me), such a miracle alone would not suffice to rectify the sad state of software quality. Managers don't do the coding. It's not enough to write software that merely "works" (although that's a good start). That attitude is a chief cause for the software mess we've been discussing this year. It's just as easy to write good software as it is to write bad software, once you know how, and knowing how is chiefly up to the developer.

Nonetheless, what I'm about to say applies to everyone. (Please overlook the fact that it's blatantly obvious on paper—somehow it doesn't find its way into mainstream practice.)

Over 20 years ago I interviewed for a teaching position at a community college in California. In the usual gang interview that such a process entails, one of the panel members asked the quintessential "thought question" for educators: "What, in your view, is more important: mastery of the subject matter, or being an effective teacher?" The question is a little devious because both are absolutely essential to effective teaching. I suppose this is meant to be one of those "let's see how you think on your feet" experiences.

My answer is also devious. I finesse the contrived clash of concerns by asserting that the most important preparation for good teaching is to be a good learner. An effective teacher continually learns about his discipline. He also is wise if he sharpens his ability to communicate, including getting to know each student individually. Both endeavors are rewarding and never boring, especially in our industry sector.

The principle is universal. Those who continue to learn make the greatest contributions and have the most fun. Think about what you learned in school. In my school days Fortran was all the rage, data structures were still evolving apace, and computation theory was still a graduate class. I never had a class in C, C++, or software engineering, but I didn't need one—what I learned in school was how to learn, so I picked up what I needed on my own after I left school. Still do.

Just two weeks ago I heard for the first time a bit of wisdom that says it best:

He who learns from one who is learning drinks from a running stream.

Quality is forever a moving target. Only those on the move can come close to achieving it. Quality software and quality mentoring come from running streams.