Bridging the Gap:

Software Engineering Education and Training

I seem to be a perfect fit for the BackTalk column in this issue. Way back in 1977, I was a (oh so) young Sgt. in the AF, assigned to Keesler AFB. The section I was assigned to teach was the "Intro to Computer Processing" course. We used a Hughes 407L (feel free to Google it – it was old even then). For the next three years, I taught "technical training." I returned to Keesler again in 1983 and taught another three years – this time teaching Ada in the STARS program (Software Technology for Adaptable, Reliable Systems). We were supposed to be conducting "training" – but I am afraid that we ventured into the educational arena.

What's the difference? It's a HUGE difference. Education involves teaching theory and history. It's all about improving your knowledge and making you more intelligent. A crucial difference is that education is not about a job skill. It's often said that an educated person is more employable, but education is not about just getting a job. Training, on the other hand, is about a "skill." Its sole purpose is to transfer practical information and skills to make you more employable. A skill teaches you repeatable tasks that you master to learn your craft. Education is about thinking.

Back in 1983, I found it impossible to "train" software engineers. I suppose there are some skills appropriate to software engineering, but coding in Ada (or any language) is a tiny, tiny part of "software engineering." Coding might be easy. Software Engineering is hard. It's more about education – the theory.

In 1986, I was lucky enough to get an assignment at the USAF academy. I officially moved from "training" to "education" - different type of students, different goals. While I expected my students to be able to code, I was more concerned that they appreciated the differences between a binary tree, a 2-3 tree, a red-black tree, ….. you get the idea. Since 1986, I have been involved in education almost continually (except for a 12-year break as a consultant. Which is where I myself learned a lot. But I still taught college part-time.)

I am currently teaching college again – and don't ever plan to quit. I seem to be a good professor – my students appear not to dislike me too much, and based on tests and projects, my students seem to learn a bit, too.

But what do they learn? That's the dilemma. We follow an ABET-approved curriculum, and we collect enough metrics to ensure that we are meeting our outcomes and objectives. We have about 43 semester hours of computer-science related material plus the required fine arts, natural and physical sciences, math, English, political science, writing and

speaking skills, etc. A typical, well-rounded college education. Not training, education.

Education. That's what a college/university does. I like to think I produce Computer Scientists and Information Technologists that rank right up there with the best of them. So, after four years of nurturing critical thinking skills in such areas as data structures, operating systems, software engineering, information security, discrete math and analysis of algorithms, we've done what we are supposed to do in terms of education. Our students proudly walk across the stage, wave their diplomas to adoring and proud family, and…well, now it's YOUR job to train them. In the Air Force, we called it "On the Job Training" (OJT). They need LOTS of OJT.

Here's why – most students consider 1,000 lines of code a "large" program. They write a program, run it under a single set of test conditions one time, and receive a grade. No realistic configuration management is needed, nor risk management either. They have seldom reused code, nor had to worry much about interfacing with legacy systems. They typically get all of their requirements on a single sheet of paper (sometimes it's actually two-sided!) They probably know Java and C++. Never had serious user interaction, other than an occasional interaction with a professor. I'm not saying this is bad – let's face it, it's about all you can do during a four-year college career. We do the education (and we do it well). You take our educated graduates, add some training, and make productive developers out of them.

So – how are you doing with my bright young crop of educated computer scientists after you hire them? How do you facilitate converting their knowledge and intelligence into usable software development skills? Do you give them a mentor? I mean, not just assign them to a supervisor – but really assign them a effective mentor? One who still feels the excitement and joy of developing software? One who know some of the latest tools and techniques? Do they have time in their schedule to talk with their mentor weekly (daily would be better)? Does the mentor have good people skills? Are there weekly or monthly "brown bags" for them to learn new skills (or sharper the ones they already have)?

How about their working environment? Do they work as part of a team? Experience has shown again and again that working with a peer in developing software is one of the best ways to bring new developers "up to speed." Except for rare group projects – this is not a skill or environment that they have learned during school. In fact, more colleges and universities discourage group work – it's much harder to assign a grade unless each student shows me how well they individually have mastered the theory.

Do you have a way to help them deal with not only the frustration of incomplete requirements and users who don't even seem to know what they want? Trust me – I stay in contact with a lot of my former students –incomplete requirement issues seem to bother many of them a lot.

I hope I speak for the educators out there – we're doing the best we can. We are working to educate our students. Once they graduate – it's your job start their training and expand their job skills. Determine what you want your developers to be capable of – and see what education they are bringing to the job. These new developers want to bridge the gap between their education and the job skills you require – it's up to you to facilitate their training. Different people will have different backgrounds. Design your mentorship/ training programs accordingly – one size does not fit all.

David A. Cook
Stephen F. Austin State University
cookda@sfasu.edu