Teaching versus training

Gary Pollice

March 11, 2003

Software development is a knowledge-based profession. We value the knowledge we acquire as well as how we apply it to deliver value to our customers. Producing software products requires harnessing employees' collective knowledge, and companies that survive in the marketplace view that knowledge as a critical, valuable resource that must be grown and nurtured.

In the academic world, our job is to build a knowledge foundation that will serve students for a lifetime. This foundation must be broad, based upon fundamental principles. And we must enable students to go forth and apply those principles in a variety of ways. Then, it is the employer's
job to supplement our students' knowledge with training for a business context and update their knowledge to keep pace with advances in technology.

So how do we define the fundamental differences between teaching and training?

First, let's look at definitions for these activities in the Merriam-Webster Online Dictionary. 1

*Teach* has many alternate definitions, including:

- To cause to know something
- To guide the studies of
- To impart the knowledge of
- To instruct by precept, example, or experience

Definitions for *train* are:

- To form by instruction, discipline, or drill
- To make prepared for a test of skill

Note that training focuses on skill; the definitions imply a narrower focus than teaching and possibly a shorter timeframe. We might associate training with the notion of exercises that we repeat until we "get" the skills we are trying to acquire - until they become almost second nature.

The definitions for teaching, in contrast, imply deeper knowledge and a longer timeframe. We often hear the term "lifelong learning," but I can't recall ever hearing about "lifelong training."

**Different goals**

Do teaching and training have different goals? First, let's examine this question from two viewpoints: The people receiving the teaching or training, whom we will call students; and that of the organization to whom the students report. For full-time college students, that organization is usually the "Mom and Dad Company." For professionals in the workforce, the organization is usually their employer.

**College students' goals**

What motivates students? As a teacher, I'd like to say that students come to my class with an intense desire to learn. But I'm not that naive - rarely does a student have such a desire. Most college students have three primary goals with respect to their courses:

1. Getting good grades. Like it or not, in academia, grades are how you are measured. They're not always fair. Some people don't test well and struggle to get good grades. But most instructors assign grades to indicate how well students have grasped the course material in a course.
2. Qualifying for continuing financial aid. With the cost of college education today, financial aid plays an important role for many students. Good grades are a requirement for maintaining...
financial aid, typically along with other requirements, such as working on campus for a certain number of hours.

3. Finding employment after graduation. This is, by far, the primary motivation for most students. Many of my computer science students are worried about job prospects when they leave school. They know the market for software engineers is not as robust as it was when they began their course of studies. So they work hard to develop enough skills and basic knowledge to make themselves attractive to potential employers.

Professionals' goals
What about professionals? What is their motivation for additional learning? Surely not grades. Professional courses, seminars, and workshops are typically not graded. You pay your money and walk away with as much knowledge as you can absorb. Nevertheless, I think professionals have three goals that map closely to those of college students.

1. Get a good annual review. Professionals do receive "grades," just not in courses. Instead, they are graded on their job performance. Taking courses might help them get better reviews - first, because it demonstrates their personal motivation and a desire to help the organization, and second, because it gives them skills to do their work more effectively.

2. Keep your job and get a raise. Just as students must work hard to maintain their financial aid, professionals must perform at a certain level in order to keep their jobs and get salary increases. During the last few years, many people have lost their jobs, both in the software industry and others. This sense of insecurity is something we would never have imagined ten years ago.

3. Learn what is necessary to get a better job. Almost everyone looks toward the next job they want when they consider voluntary training - whether it is a promotion in their current company or a new, exciting position in a different organization.

Balancing training and teaching

Everything we do in our lives requires a balance between two things or more - a compromise. This column talks about the balance between theory and practice, so it is appropriate to consider that balance with respect to training and teaching. In my January 2004 column, I explained how I am seeking the right blend of theory and practice in my software development courses. My primary job is to prepare students for the rest of their working lives and help them succeed in their careers. So I first have to ensure that they learn fundamental principles and then teach them to use specific tools and techniques as time allows.

A sample development project

Things are a little different in a work environment. Let's say you are managing a new project to develop an enterprise-wide system to manage inventory in multiple, worldwide locations. This will involve Internet technology, database systems, and so on. You add two new members to the project team. Sarah is a new graduate with a computer science degree. She has learned software engineering principles as well as some specific technologies, such as Java programming. Mike has been employed with your organization for a while; he has attended in-house training
courses in programming as well as on other topics that apply to his job. Mike started his career in the military, which he joined right after high school; he never attended college. Both of these people have valuable skills. If you could combine their two skill sets, you would have an ideal team member who could step in and hit the ground running.

However, as both of these people are new to the project, they need some orientation. Sarah will have the challenge of learning to understand not only the system itself, but also the culture within your organization and the specific ways people conduct business. And Mike will be challenged as well; your project has decided to adopt tools and processes from IBM Rational Suite®, which is new to your organization. You plan to follow a customized version of IBM Rational Unified Process®, use Rational ClearCase® for configuration management, Rational RequisitePro® for requirements management, and so on. The team might use additional tools as appropriate, some of which you will develop in house.

**Applying learning and training**

Now let’s see how their respective backgrounds affect the way Sarah and Mike handle learning challenges. First, in order to learn about the system under construction, both of them look at the Software Architecture Document (SAD). This describes some architecturally significant use cases, UML diagrams, plans for development, testing, and so on.

Mike is familiar with the format: It’s the company standard he’s been trained on. He quickly grasps the project’s scope and potential business impact. His internal training courses and experience within the company have prepared him to understand both the jargon and company-specific business issues. Sarah, in contrast, struggles a little over the terminology, application domain, and acronyms; but she quickly grasps the overall architecture. She understands multi-tier architectures and the technologies used to represent this one. She also understands why certain areas have been identified as architecturally significant and begins to see how the designer might apply some common design patterns to simplify the architecture and make it more robust.

You have assigned Mike and Sarah to the same small team responsible for implementing part of the business logic in the system’s middle tier. The team has already decided that they can break up the use cases into user stories, which will help them schedule work and track progress. They have also decided to use JUnit for unit testing as they work. You allow them to hire an instructor to provide a day of training for the team on how to create user stories and apply JUnit.

After the training, both Sarah and Mike understand the basic concepts and applications for user stories. When it comes to JUnit, Mike understands the mechanism, but he has trouble deciding what to test. Sarah remembers the principles of testing she learned in her software engineering class and begins to write positive and negative tests, boundary condition tests, and so on. She also takes time to help Mike develop good tests. They meet for several sessions of pair-programming for the tests. Sarah learns more about the application domain from Mike, and he picks up new techniques and basic principles from Sarah. She suggests to you that using a code coverage tool in conjunction with JUnit would provide rapid feedback about the completeness of the unit tests. When you give her the go-ahead, Sarah reads through the IBM
Rational PureCoverage manual and then writes a short document explaining how to use the tool to check the team's Java code. She holds a brown-bag lunch to answer questions about Rational PureCoverage, and then the team starts using the tool to improve their testing.

As the project proceeds, the requirements change, and the team has to adapt the code they've already created to changes in the architecture. Luckily, the architecture has been well constructed, so there are clean interfaces for the major modules. However, some of the changes threaten certain interfaces and could have ripple effects through the subsystem the team is building. Sarah thinks about the design patterns she learned in one of her classes and realizes that, instead of refactoring the existing code, the team could make the code much more robust by adding adapters to the code base and applying common patterns, such as decorator, factory method, and so on. Even though she learned these design pattern implementations in C++, she understands the principles behind them and can easily adapt them for Java.

Sarah turns to Mike again for help with the mechanics of setting up her environment to take advantage of the advanced features in IBM Rational ClearCase, because he's had extensive training on this tool. Once she is adept at using these features, Sarah becomes much more productive. She also realizes that the team might be able to leverage Rational ClearCase to achieve continuous integration, a goal discussed in her software engineering class. She reads up on how to script tools for Rational ClearCase and then writes a couple of Perl scripts to make integration tests run continually as the integration builds proceed. When they adopt this change in process, team members find that they can identify integration problems more quickly and fix them more easily.

Mike, meanwhile, uses his knowledge of the organization and how it does business to identify several requirements that simply do not reflect how business is actually conducted. He also knows who to talk to in order to adjust the requirements, so you appoint him the domain expert for the team.

You pat yourself on the back for bringing two high-powered people like Mike and Sarah on the team. Each one knows how to apply what he or she has learned and also how to leverage one another's strengths, which are complementary. But in the long haul, who is most likely to get ahead? I'd put my money on Sarah.

Sarah's grounding in fundamental principles allows her to quickly find solutions to common problems and think strategically about measures that will increase both coding efficiency and system quality. Mike's practical knowledge has given him a deep understanding of his work environment and business processes that he can use to identify problems and quickly get them resolved. Sarah will acquire these same capabilities as time goes on - that is the nature of on-the-job training.

But how could Mike acquire theoretical knowledge like Sarah's? Many people do it by going back to college, but some cannot make that kind of commitment.
Create an ongoing learning environment

Fortunately, there are some healthy organizations that not only offer training but also support opportunities for employees to acquire knowledge about theories and principles relating to their work. By giving people time and funding to pursue knowledge through outside sources, such organizations can give their "Sarahs" the domain knowledge that their "Mikes" have, and also help their "Mikes" learn what their "Sarahs" learned in academia. In other words, they create a learning environment that supports both the short-term and long-term goals of the organization - and the employee.

In a business setting, managers can act as "teachers" who guide learning for their employees. Managers can also help team members plot their careers, based on what each person wants to learn. And good managers can match the employee's desires with the company's needs. Many successful organizations recognize that this form of teaching (or mentoring) is critical for business success. They make it a condition for managerial promotions and pay increases. But they also give employees time to learn and opportunities to use the knowledge they acquire.

On the academic side, we will continue to educate students about basic principles, sprinkled with a little training; we'll give them some practice and a lot of theory. When our graduates join your organization, we hope you will continue to expose them to theory rather than just providing practice in the form of training. You can take advantage of the resources at learning institutions in your area and provide programs that keep employees up to date on issues and ideas about software development. Invest in the future. Training helps people today; teaching guarantees them - and your organization - a bright tomorrow.