See One, Do One, Teach One
The Medical School Model for Software Developers

STAN BLANK, PHD
Five Pack Creative
February 29, 2016

Abstract

The traditional medical education model of “See one, do one, teach one” has much to offer for the efficient training of software developers. In order to fully understand a topic, it is important for the student to watch an expert model the expected behavior required to solve a software issue. The student must then be able to work through a software problem to find a solution based on the expert model. Finally, the student must be able to explain or teach the software procedure or concept to a colleague or team member. The act of teaching requires a deep understanding of the subject and is critical for training valued software developers and developing communication skills among team members.

Traditional Software Training

The great 20th century physicist Richard Feynman was also a noted educator. He, with the help of colleagues who transcribed and edited his lectures, created an inclusive set of lessons for the first two years of physics instruction at Caltech. The resulting texts, The Feynman Lectures on Physics are still in print and are widely regarded as outstanding references for both physics students and their professors [1 p. 52]. Feynman was once asked by a colleague (David Goodstein) to create a lecture on why spin one-half
particles obey Fermi-Dirac statistics. Feynman stated that he would prepare a freshman lecture on the topic, but returned a few days later without success. He stated that he could not prepare a freshman lecture on the topic, which meant that we did not understand it [2, p. 339] [1, p. 52]. There is a lesson here! If we are unable to explain what we know using terms that a beginning student would understand, then we don’t understand the subject material.

Workshops for training software developers generally have some common attributes. Attendees are gathered in a relatively large room filled with laptops and fronted by a projector screen displaying the contents of the speaker’s LCD. The speaker is a recognized expert in a particular software development domain. The students watch the projector screen, listen to the speaker’s explanation, ask some questions (tentative or otherwise) and then attempt to follow some of the speaker’s talking points by coercing their laptops to display (hopefully) something similar to that shown on the speaker’s projector.

If the attendees/students are fortunate, the expert(s) will circulate throughout the room and provide 1-1 support for students who have questions or comments. Unfortunately, too often students are isolated and must rely on hastily taken notes or the questions of others to help solve the contrived software problems presented by the workshop.

There are no panaceas in education and there is no one best way to educate everyone all of the time. It is the experience of this educator that ALL instructional models work some of the time for some of the students. Likewise, even the most time-honored and validated teaching methods will fail, often through circumstances completely beyond the control of the teacher. Traditional workshops do provide valuable instruction which benefits some developers. However, it is a valid point to wonder whether or not we are serving attendees/students/future developers/future employees in the most efficient manner. The question becomes “Can we do better than the traditional workshop model?” The proper goal for any teacher
is to match an instructional model with a specific subject in order to more efficiently teach the intended student audience.

See One, Do One, Teach One

Medical school education has long employed a model of teaching that involves the concept of “See One, Do One, Teach One” (SDT). The SDT model has its roots in experiential learning, specifically problem-based learning [3, p. 361]. Problem-based learning is designed to provide students with real-world experiences requiring observation, reasoning, cooperation and communication skills. The SDT model of instruction begins with an expert, either a professor or skilled practitioner, demonstrating a specific medical or surgical procedure. The students are then expected to practice the demonstrated skill (with supervision). Finally, the students are required to explain or teach the skill they have just learned to another student or group of students. This explanation is an oversimplification, certainly, but the outline is valid. Think of a group of students following and observing an attending physician or resident during daily rounds. Eventually, the students will be expected to perform the same tasks as the attending or resident. Finally, the students become the teachers (attendings or residents) with their own “flock” of medical students in tow.

SDT is still in use in modern medical education and while it has some detractors [4], it is doubtful that wholesale change will eliminate SDT from medical education, particularly in medical procedures and surgery. A fairly recent article from the Georgia State University Law Review suggested that the SDT model might be an appropriate method of instruction for law school education [5]. However, we are here to discuss the training of software developers and not medicine or law! Would SDT be appropriate for teaching students how to become software developers? We think so! The SDT model is perfect for subjects that involve actions such as surgical procedures, courtroom arguments and software development.

The implication in the Feynman anecdote in the first section of this paper is that if you are unable to “teach” an idea concept, then you don’t understand the subject material. Teaching a concept does not have to involve standing up in front a large group and explaining your method. It can be as simple as talking to your lab partner and guiding him through your solution. It can even include talking with yourself if you are developing software.
alone! If you are able to explain your methods, then you understand them. It is our contention that once you explain a few concepts to another person, you are able to learn new concepts more efficiently. There is also an added benefit of enhancing communication between and among potential team members.

So, the model we employ at Five Pack Creative training seminars is the SDT model. We have some of the top iOS developers in the world in our company and these experts enjoy sharing their knowledge with others. This forms the “See One” aspect of Five Pack Creative training. Of course, “One” is not literal in medical training and it isn’t literal at Five Pack Creative. It may take two or three or ten examples from our experts to teach the target concept. Likewise, students may “Do Two” or three or five problem examples in order to grasp the knowledge that leads to a problem solution. Finally, the critical step is the “Teach One” component. We expect students to explain, demonstrate and discuss solutions to the workshop problems with their lab partner(s). While not everyone has the ability (or the nerves!) to stand up in front of a group and teach a challenging programming structure, the goal of our training is to insure that you could do so if you wish.

The goal of our software training is the desire for our students to have a deep “Feynman” understanding of software development. We want our students to be able to explain to others why they chose a particular function in order to add a specific feature to an app and how their choices fit the goals of the project. In addition to programming techniques, you can expect to learn Agile-Scrum, Code Review, Sprint Demo, Test Driven Development and even Paired Programming during our workshop sessions. We want our students to be able to communicate, as simply as possible, their thoughts and programming methods to everyone on the team. We believe that top software developers are excellent communicators and can teach others how to become better programmers. The “See One, Do One, Teach One” method perpetuates the software development field and we at Five Pack Creative are at the forefront of this exciting educational opportunity.

Conclusion

Five Pack Creative training is unique, effective and efficient. Our teaching will cover not only coding, but also tools for managing source code and
work tickets as well as key processes to expect when working on projects. The brand of our training is ALT-U, which is an abbreviation of Alternative University. You will learn differently (‘Think Different’) at Five Pack Creative’s ALT-U! Our intention is to train software developers who could potentially work for our company and we have a vested interest in our training methods. If you are interested in learning more about Five Pack Creative training, you can contact us:

Five Pack Creative
5046 Kiowa Dr
Frisco TX 75034
Ph: (214) 997-4359
info@fivepackcreative.com
http://www.fivepackcreative.com/
http://www.fivepackcreative.com/alt-u/

References


