Fred Fonseca – Teaching Philosophy

In introductory programming courses students do not have the skills to undertake challenging problems. As a result they frequently drop out, fail, or get by without learning. Students might learn bits and pieces but they are not able to create a functional application.

A common method for teaching programming languages is a sequence of lectures complemented by some practical exercises (labs). The lectures introduce the vocabulary, syntax, and abstract concepts, and the labs require students to apply the lecture material to solve problems. The rationale is, "Until the students master the vocabulary and concepts they cannot program." But exposing students to information before they know why they need it or how they will use it results in student frustration, which in some cases can lead to failure.

I wanted to develop a learning experience that presented students with a project realistic enough to excite their interest, while at the same time sufficiently structured to give them focused opportunities to practice and learn basic skills. I found a good model in the original *The Karate Kid* movie. In that film an eager teenager, Daniel, wants to defend himself against Karate bullies. An aging Karate master makes a deal with the bullies' teacher to lay off Daniel in return for a promise that Daniel will face them in tournament combat in a few weeks. Daniel demands to start right away learning how to punch. The master, instead, assigns him to wax and polish antique automobiles, sand a deck, and paint a fence. After several days, he takes the student to the beach to struggle for balance in surf. Daniel repeatedly asks, "When do I learn to punch?"

The novice's first impulse is to do something. The likely results are mistakes. Deny the impulse and you throttle the desire to learn. Allow a cascade of failures and you destroy confidence in being able to learn. Instructors then have to build on the desire to learn; not dampen students' curiosity or abandon them to flounder. That requires instructors to use something like the Karate Kid model.

The model is: 1) provide examples of and promote a desire to master new knowledge; 2) create exercises that involve what students already know, but demand changes; 3) help students relate what they learn about basics to a realistic project; and 4) give them opportunities to try out new skills by creating applications. The model embeds basic practices within an expert performance.

I opted to implement a two-phased approach to creating a programming course using a similar method: the instructor drives the scaffolding phase of training in basic skills; the students drive the performance phase of applying those skills.

The scaffolding phase includes instructional practices that encourage students to play and hone their skills without fear of errors. This is Karate Kid scaffolding – start with tasks that students can do, but interject novel demands to push them to the next level of understanding. This approach helps reduce the negative consequences of early mistakes while also allowing for learning by trial and error.

The performance phase is the development of the software project itself. In the second half of the semester students face the final project in a do-or-die confrontation, realizing that, "we are going to make it or not." They have to put the pieces together and make them work. Some students are shocked that the "teaching" is finished. There are no more exercises, quizzes, or any other activities besides the work on their own projects. Now they are programmers, the instructor is a senior expert, and the teaching assistants and interns are colleagues.

The transition between phases occurs at different times for different students as they learn to leave their comfort zones and take responsibility for their own learning. Some students demand more time for teamwork to design and execute the project. Other students wait until late in the course to act independently. The two-phased approach of scaffolding and performance allows for these different kinds of transitions between phases; it lets students advance at their own pace while providing positive models for laggards.

At the end of the class the students talk about the elegance and attractiveness of their programs, not about grades. What once seemed impossible becomes a doable task. The last week of the course explodes in celebrations more like those of triumphant athletes than the numb relief of students who have, at long last, finished the course.