Teaching Statement

Jeremy Gottlieb

My teaching philosophy centers around the idea that students most effectively learn material when they discover concepts for themselves, rather than simply having them handed to them. This is most effectively accomplished when learning is hands-on, especially in an engineering field like computer science or computer engineering. The basis of these disciplines is the use of technology to solve problems that the world presents. Thus, the job of a good teacher is to provide students with an effective toolbox of problem solving skills, and then point them on the path to developing solutions for themselves. Understanding the specific syntax of Java or C++ matters less than understanding the general programming tools available, and how to use them effectively. The most valuable thing any teacher can impart to their students is not the material in textbooks or lectures, but the ability to develop their own solutions to problems they encounter, and to critically analyze data and evidence to reach their own conclusions about what techniques are most effective for generating those solutions.

Computer science and robotics are disciplines where this type of hands-on, problem solving based approach to learning is particularly appropriate. However, there is still some amount of rote knowledge-transfer that needs to take place. At some point, either through a lecture or a textbook, students need to be shown what the syntax of a for loop is. Unfortunately, time spent in class going through these types of materials is time that cannot be spent focusing on the process of problem solving.

In this vein, when I last taught introductory computer science (Spring 2013) a colleague introduced me to the concept of the inverted (or flipped) classroom. In this paradigm, passive learning activities, such as lectures and reading, are done by students at home, on their own, via videos and on-line workbooks. Class time is then spent on active learning activities such as group problem solving, peer critique, and pair programming. So, for example, a lecture about the syntax of a for loop in C++ would be recorded to a video. This video might be embedded in an interactive workbook that provides students with code examples and quizzes them on various aspects of the syntax. Then, during class time we would first have a small-group activity involving when and how to use a for loop. This would then be followed by a slightly more involved program to be done individually or in pairs that involves the application of a for loop and its integration with previously learned concepts. There are still larger homework projects as well, but much of the process of learning how to problem solve is done in the classroom with more direct interaction between myself and my students, as well as between the students themselves.

When it was presented to me, this division of labor was very intuitive to me, especially as applied to computer science. There is no particular reason that everyone needs to be together when first being presented the syntax of a for loop, but it can be extremely valuable to work with other students, and to have the professor available, when trying to decide whether a for loop is appropriate to the solution of a particular programming problem, and then when trying to implement it for the first time. Having a new set of activities for each class period also give students a tremendous amount of practice at using their newly acquired programming skills to solve particular problems. Students are still given homework they must solve by themselves, but only after having had multiple opportunities to practice skills and get feedback about their implementation. To me, this method represents an excellent mechanism for giving students not just the toolbox they need to use programs

to solve problems, but also for giving them the best opportunity to hone the problem solving skills necessary to best understand when and where to use those tools.

The hands-on style gives students an opportunity to accomplish things they might not in a tradition course setting. At Carthage College I created an introductory robotics course where students built and programmed a robot "from scratch" (using a kit I put together for them). We closed the term with a Grand Challenge style competition in the lobby of our science building. I believe that these are the types of opportunities that students learn the most from, and that get them the most excited about any given topic.

The courses that I have taught include introductory computer science, data structures, artificial intelligence, computer networks, and introductory robotics. As part of these courses or in other contexts, I also have experience with theory of computation, control theory, embedded systems, and machine learning.