Teaching Philosophy

I try to make computer science an enjoyable subject, while addressing the challenges, methods, and underlying complexities of problem solving. I like to emphasize the major "nuts and bolts" issues and methods which pervade a subject. I also like to present and deal with the big picture, realizing that sometimes details must be addressed, but am determined not to get bogged down by them unless they are necessary for understanding. Both are important in my view.

In general, I would divide the world of educators, scholars, and researchers into *perfectionists* and *"productionists"*. I am concerned with EDUCATING so that students can learn, understand, solve problems, be motivated and get things done. I know and respect the work that goes into producing perfection. I also know many perfectionists who never finish what they set out to do. Given the choice, I would prefer to be more productive even knowing that my work is not perfect.

My teaching philosophy has been developed from a perspective which was impressed upon me early on by my late father: "There are no poor students, only poor teachers". Although, I can safely say that my experience has not confirmed this point of view, it has served as a starting point and stimulus towards always striving for real excellence.

The young mind of the typical college student is a fragile entity with potential that you can turn on or off. I view my job as the instructor for a computer science course as the primary perpetrator of interest and thinking at the appropriate level for the subject matter and beyond.

For example, at the introductory level, teaching large classes, on the use of *applications and elementary programming* on a personal computer, I feel that a "promotional" attitude is essential. That is, I promote the main features of a programming language or application; addressing the questions: "What is the language/application intended for and what can students satisfactorily get done for their needs?" Essential, specific, syntactic, developmental, and conceptual issues are also discussed. For *introductory*

programming courses **the issues of structured programming and problem solving, combined with those in the previous sentence are considered of critical importance.**

During the past few years I have made the adjustment to a breadth-first approach to teaching the introductory computer science course which combines topics like hardware, software, programming languages, and algorithms.

For intermediate level courses such as ***data structures and algorithms,*** **I focus on more complex programming methodologies, problems, and issues, with consideration for time-space efficiency tradeoffs. By this point students are expected to be more experimental and research-oriented in their thinking. In advanced courses like** ***artificial intelligence*** **(AI) the breadth and diversity of the discipline is stressed; the problems which distinguish AI from other disciplines and other approaches in computer science are considered; the methodologies, tools, and languages employed in AI research are also presented.**

Finally, but not at all in the least, I have become firm in the belief that two educational approaches are excellent ways for students to develop their academic skills at all levels:
1) That students pursue individual case studies in ~~a~~ whatever the area of concentration for a course. Students can study a particular topic for 4-6 weeks and then write a 5 page double-spaced paper (in academic reference style) and make a brief class presentation. For about two months (or longer) students are asked to develop a group project which will generally involve some design, programming, development, and testing, a 20 page group paper, and a class presentation of 20-30 minutes.