Statement of Teaching Philosophy

Matthew Anderson

I have had the pleasure of learning from many excellent teachers. Looking back at the truly exceptional ones, I see that their style, techniques, and personalities span a wide spectrum. However, they have one striking commonality: the passion to instill the joy of learning in their students, to prepare them to excel in their future endeavors. This is the ideal I humbly aspire to. In what follows I try to describe a few personal best practices, informed by my experiences, towards this ideal.

Experience. I have had frequent opportunities to serve as both a teacher and a mentor. At the University of Wisconsin-Madison, I taught three semesters of introductory programming in Java as an instructor, and volunteered as a coach during the intensive training of the department's ACM International Collegiate Programming Contest (ICPC) team for four years, with one year as the head coach. In addition, I assisted in numerous courses, including the undergraduate programming, data structures, numerical methods, and networking courses, and a graduate-level programming languages course. During my undergraduate studies at Carnegie Mellon I assisted in an introductory physics course, and tutored students in upper-level physics and computer science courses.

Since my research focus is complexity theory, I would enjoy teaching theory courses like discrete math, complexity, and algorithms, and relish the chance to develop courses on related special topics, like quantum computation. The variety of my teaching experience has made me comfortable with teaching courses across a wide swathe of the undergraduate computer science curriculum, and I welcome the unique opportunity to learn that teaching provides. I specifically asked to teach introductory-track classes at Wisconsin because of the opportunity to hook students into the amazing field that is computer science!

Create an Engaging Environment. For me, the ideal learning environment is participatory, collegial, and reflective. To facilitate interaction, I keep class time structured, but varied. For example, when I taught introductory programming I divided the class time between lecture, small group activities, and interactive programming sessions. To carry this engagement outside the classroom, I gave out short computer-sciency brain-teasers at the end of class to encourage students to take a moment to revel in the joy of problem solving, to emphasize that computer science is more than just programming, and to give us something to chat about before the beginning of the next class.

I try to be as approachable and accessible as possible—one Thanksgiving Day I had several phone conversations with a student trying to help him get a programming project working. My lecturing style is energetic, but casual and conversational, and I try to inject spontaneous humor when possible and appropriate (the confines of this document may make `this` paradoxically-spontaneous non-spontaneous attempt at Java-reference-reference meta-humor fall flat)[1].

Of course, sometimes miscalculations are made, by students or me; establishing a shared accountability for learning can turn a potentially negative situation into a positive one. I have often used short quizzes or brief homework exercises to give students immediate feedback on their understanding, and to encourage them to stay engaged with the material. I have also used short anonymous surveys so that students can assess my own performance. These frequent reciprocal reflections help to establish joint responsibility for the course: They allow me to adapt difficulty, depth, or delivery of the course in response to students' needs, and allow students to dynamically adjust priorities. For the same reason, I have attended several day-long professional development workshops examining all aspects of teaching, including a Teaching and Learning Symposium at UW-Madison. Hearing the experiences of others and sharing my own perspective has helped refine my teaching methodologies. While I have not had

[1] As a commitment to spontaneity I used the first joke I thought up. Sorry. This was the second.

the curriculum control to experiment with new practices, like MOOCs, augmented reality, or inverted classrooms, I am intrigued by the possibility they present to enhance learning.

Enable Accomplishment and Discovery. In general, I view homework and projects as learning tools rather than a means of evaluation. This allows me to encourage students to collaborate with each other in a dynamic way, which is difficult to achieve in a lecture room, and to make full use of the available resources, much like is expected in the real world. Ideally, the problems are simultaneously interesting, challenging, and instructive, and give students a sense of accomplishment when they complete them. For example, when I taught introductory programming, one of the month-long final projects I designed was a variant of the classic arcade-game Frogger. Since this was a first-semester course, to make the project concurrently accessible and interesting we constructed substantial front- and back-ends. This allowed the students to focus on the core programming task, the overall design, and other creative elements while being able to ignore some low-level details. It also allowed us to greatly automate the grading process. The end result was a program that was complex, graphical, and exercised everything the students had learnt throughout the semester.

I advocate incremental design—designing projects that gradually buildup from simple exercises to complex and challenging problems. The benefits of this approach are many-fold: it allows students to solve more difficult problems than they could have otherwise accomplished on their own, it can be used to guide students to discover new ideas, and it has granularity that provides students feedback about their understanding as well as the ability to select their own level of difficulty. While leading lab sections I would come up with a set of increasingly challenging questions for students who finished the basic lab early, e.g., in a lab focusing on navigating a robot in a discrete rectangular grid, which was able to move in the four cardinal directions, I would ask: Suppose the grid contains no obstacles and the robot starts in the upper-right corner, can you traverse every cell in grid without repeat and return to the starting point? How does this depend on the dimensions of the grid? For some dimensions the traversal is trivial and others impossible; how one might argue the latter provoked much creative discussion then and in the next lab.

Support Learning through Mentoring. I look forward to serving as a mentor and advisor to students at all levels of development: from those just beginning their exploration of the wonderful world of computer science, to those taking their first tentative steps into research, and finally to those working shoulder-to-shoulder with me as a colleague. My goal as a mentor is to support and magnify a student's innate abilities, and to help them realize their true potential. This can range from discussing study skills and organization, to advising them on which courses to take, to helping them through some of life's difficult transitions, or to providing supplementary instruction to enable success in a challenging course. For example, I helped organize social events and reading groups aimed at acclimating new graduate students to the theory group at UW-Madison.

I relish the opportunity to guide students as they begin research. I have several research projects in the preliminary stage that are appropriate for and accessible to an advanced undergraduate. For example, I began a programming project using the high throughput computing system at Wisconsin to search for algebraic structures that facilitate faster matrix multiplication.

The most difficult aspect of being a mentor has been how to encourage a student to continue working towards a challenging goal even though there may be no discernible short-term benefit. For instance, when coaching our ICPC teams it was a challenge to get students to practice during the months leading up to the regional and world final contests. After several years of experimentation, the best strategy seemed to be making each team responsible for leading the discussion on a fraction of the problems. This established a communal responsibility for learning, and, anecdotally, improved both the amount practiced and the level of interaction during our discussion sessions.