Statement of Teaching Philosophy

Joel Ross

As a teacher, my goal is to help students approach computer science as more than just programming, but as the study of processes that people use to solve problems. Learning computer science requires developing and mastering such problem-solving techniques—in particular, the skill of solving problems through constant questioning and testing. Students must repeatedly ask questions such as: "What is the next line of code?" "Will this algorithm help solve my problem?" "Why does the system behave this way?" "How does this technology influence the world around it?" and then seek out the answers in order to construct an understanding of computer systems. **My teaching philosophy centers on helping students to develop a questioning attitude and apply it as an approach to solving problems throughout their careers.** I aim to help students practice asking questions that allow them to learn how to learn.

In introductory programming courses, I model this problem-solving approach though live programming demonstrations, involving students in the questioning process by prompting them: "what is the next step?" Even if they cannot produce the correct answer, just considering the question helps students to connect the material to previous topics and other experiences. For example, a student's suggestion to use a list to solve a problem better suited to a tree led to a discussion that helped solidify the whole class' understanding of both data structures. Students have a chance to practice this questioning process through *pair programming* assignments, further mastering the material by simultaneously teaching someone else. I have watched pairs of students work together to understand and implement a program, practically taking turns explaining how to add the next function and then to fix the next bug. At the end of the course students present "beta" versions of their final projects to one another (as part of a formative assessment), and the common response of "wow, how did you do that?!" shows they have developed a questioning attitude and a desire to continue learning.

In my courses, students also apply a question-and-experiment approach to problem-solving to support their own self-directed learning. My courses include open-ended final projects and assignment 'extension challenges' that guide students to sources of further information on a subject. This allows them to practice applying their problem-solving skills to new topics ranging from techniques for designing software interfaces, to advanced ray-tracing algorithms for generating photorealistic computer graphics. Learning to self-teach is an explicit objective of my Software Engineering course, in which students practice using online resources to master new software libraries. Adopting this objective has challenged me to find the right balance between lecture and outside work—a balance I am working to achieve through additional lab sessions where students can utilize the professor as a backup learning resource. Such self-directed learning requires students engage in critical meta-cognition and regularly evaluate their own understanding. My courses support this reflection via regular writing assignments in which students analyze their own processes in implementing computer

programs: "what has worked and what hasn't—and why?" Moreover, I hope to encourage students to extend this critical reflection to the contexts in which technology is used after it has been developed. I am experimenting with techniques for introducing more human-centered concepts and examples in programming courses, such as by including human users as part of the traditional graphics pipeline.

In order to critically interrogate computational systems, students need to consider systems from perspectives other than their own as developers. One of my goals for performing such reflection is to help students appreciate a wider diversity of viewpoints. I strive to design my courses so that they support such diversity in students. Applying the values of a liberal arts education, I include interdisciplinary content in my programming courses, such as the history of computer technology and programming jargon. My assignments often have students work with media representations (e.g., images and sound) to support engagement and encourage question-driven experimentation, particularly in non-majors who may feel overwhelmed by the abstract symbolism of computer code. I also work to emphasize the contributions of groups traditionally underrepresented in the discipline of computer science. For example, my Capstone in Computer Science course includes analysis of the writings of Ada Lovelace, as well as class discussions on the historic role of women in programming early computers like the ENIAC. I wish to support these multiple, diverse viewpoints to encourage the participation of women and other underrepresented students in computer science, and to support a greater sense of social responsibility in all students.

As developers, we need to be aware of how our systems and technical designs affect those around us. There is purpose and ideology inherent in any act of design, and thus the meanings of computer systems must be interrogated so that technology can be designed effectively and used responsibly. A questioning stance can help students learn to bring a variety of critical perspectives to bear on the position of technology in society—including research topics such as how system designs might favor or disadvantage different social groups, or how technological proliferation may impact the natural environment. In the end, my teaching strategies support a liberal arts education by working to help students develop this critical, questioning stance for finding solutions to problems. Such a stance can enable students to become lifelong learners, prepared to effectively adapt to the wide variety of technical and social contexts they will encounter throughout their future careers.