I developed my pedagogy from experiences guiding students through coding modules, delivering lectures, attending teaching seminars, reading about learning, and mentoring students. **My teaching style uses participatory lectures with guided discovery and constructionist problem-solving assignments.** My objective is to prepare students to succeed post-graduation, and four primary goals guide my approach:

**1. Cultivate student motivation.** Intrinsic motivation can create life-long learners who test better, stay in school, and are successful after graduation [3]. Some methods I use to encourage motivation are to dispel misconceptions [4] and to give meaning to efforts [11]. In addition to common misconceptions, computing majors face additional misconceptions, such as: only men can do it, only geniuses can do it, the math is too hard, if I do it well then I'll be a social outcast (nerd, geek), programming is tedious non-creative work. For example, Mackenzie Morgan, an International Affairs major taking my computer organization lab, was considering switching majors to CS, but she was terrified of math. I explained how math in CS differs from high school math, and that I felt she would do well at it. **Next semester she told me that I convinced her to become a CS major, but what I really did was dispel her misconceptions and ease her fears.** Mackenzie graduated with a BS in CS and now works in industry.

Naïve approaches to give meaning to students, such as telling them how important learning is for their future, can be counter-productive. One approach that works is to have students explain how a course may be relevant to their lives [7]. Another approach that has worked well for me is to describe anecdotes of former students, such as Irie Grant, who came to my office one day to tell me that he was able to "knock out the interviews and land the position at Salesforce" because they asked him to explain how threads and synchronization work—I like to tell this story on the second day of an OS class, just before we introduce the process and threading model.

**2. Improve teamwork with active learning.** To improve teamwork skills needed in the working world, my classes include team projects, **which also increase student interest, thus motivation [9].** The team projects I design give students breadth starting from a problem and designing, implementing, testing, documenting, and presenting a solution. Students learn about teamwork, team roles, conflict resolution, and version control [10]. When possible, I prefer to form student teams by mixing students with different backgrounds and strengths, which the SCALE-UP [1] project has shown provides measurable benefits in both small and large classes to student understanding, attitude, problem solving, and success, especially for underrepresented groups. Non-CS students have told me that working with CS majors provided new insights into software design. I try to mix better- and worse-performing students, preferring to mix majors in cross-listed courses. One non-CS student wrote these teams "worked wonders at keeping anyone from falling behind on the assignments." Mixing students in teams works well, though some care must be taken for grading and conflict resolution. Open-source projects are another great way for students to get excited about software and team-based development. I have experimented with teaching open-source concepts that apply my domain knowledge with RTEMS to benefit students. I have guided students in learning research methods and software engineering processes including version control, code review—which provides many educational benefits [6]–and collaborative software (*e.g.*, instant messaging, wikis, Google Docs, GitHub). Students have responded positively to my use of collaborative tools in class, for example one student wrote she liked that I make myself available "throughout the final hours before a lab assignment is due, allowing students the chance to work through last-minute problems that might otherwise have defeated them."

**3. Foster technical writing skills.** I believe strongly in a "writing in the disciplines (WID)" approach for improving students' communication skills [8]. In all of my courses, I try to include the following WID coursework components: a research-oriented team homework assignment with an oral and written report; a summary and evaluation of a talk by a guest lecturer; and a term project with a design document, oral presentation, and written report. I also encourage student blogs—similar to development process journals [5]—for writing about project design and implementation challenges. **I have found student-developer blogs are useful to me for gauging understanding and progress, and useful to students for self-reflection and report-writing.** I plan to create and improve WID assignments to teach students how to communicate technical content effectively.

**4. Nurture students as research-scholars.** Project-oriented courses are ideal for teaching students about research methods, including how to ask good questions and use the scientific method. **I encourage questions by acknowledging the value of any that are asked**, and try to follow-up with more details by e-mail or class discussion board. Although students benefit from learning the basic skills of research, those skills are not teachable until students possess deep background knowledge [2]. Thus, I incorporate research methods primarily in upper-division and graduate coursework. When evaluating student work, I critique methods, analyses, and presentation of results more than artifacts (code). **By providing students with the framework to conduct research, I give students the experience and skills needed to thrive in challenging jobs.** For example, a student in my OS lab, Dan Fego, turned his team's OS term project into a senior design capstone project. The team's project took filesystem snapshots and supported roll-back of some file attributes. Dan re-designed and extended the project to include support for more file attributes and file contents. He later told me the experience—especially kernel hacking—was instrumental in helping land his first job.

**Summary** The above principles serve to focus and guide my teaching. Student successes like Mackenzie's, Dan's, and Irie's motivate me to become a better teacher, which I do by seeking advice, reading about what works (and what does not), and evaluating my teaching efficacy through self-examination, informal conversations with students, and anonymous feedback.

## Course Interests

With my background in systems, I am capable and interested in teaching courses with strong design and programming components. I have taught architecture and OS courses with both CS and CE students, so I am well-equipped to teach courses cross-listed or overlapping with computer engineering. I am interested in teaching introductory courses focusing on programming skills, software design, and computer organization, as well as developing and teaching upper-division and graduate courses in computer architecture, operating systems, real-time and embedded systems, and security. I am also interested in developing two courses not normally in the curriculum, one that explores the field of hardware cybersecurity, and one that delves into cyber-physical system security.

[1] R. J. Beichner, J. M. Saul, D. S. Abbott, J. Morse, D. Deardorff, R. J. Allain, S. W. Bonham, M. H. Dancy, and J. S. Risley. The student-centered activities for large enrollment undergraduate programs (SCALE-UP) project. *Research-based reform of university physics*, 2007.

[2] Daniel T. Willingham. Why don't students like school: Because the mind is not designed for thinking. *American Educator*, 4(13), 2009.

[3] E. L. Deci, R. J. Vallerand, L. G. Pelletier, and R. M. Ryan. Motivation and education: The self-determination perspective. *Educational Psychologist*, 26(3-4):325–346, June 1991.

[4] C. Demetriou. Potential applications of social norms theory to academic advising. *NACADA Journal*, 25(2):49–56, Sept. 2005.

[5] P. Drexel and R. Andrews. Writing in computer science courses: An e-mail dialog. *Plymouth State College Journal on Writing Across the Curriculum*, 9:60–68, 1998.

[6] P. Guo. Refining students' coding and reviewing skills. *Commun. ACM*, 57(9):10–11, Sept. 2014.

[7] C. S. Hulleman, O. Godes, B. L. Hendricks, and J. M. Harackiewicz. Enhancing interest and performance with a utility value intervention. *Journal of Educational Psychology*, 102(4):880–895, 2010.

[8] D. G. Kay. Computer scientists can teach writing: An upper division course for computer science majors. In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 117–120, New York, NY, USA, 1998. ACM.

[9] S. Ludi, S. Natarajan, and T. Reichlmayr. An introductory software engineering course that facilitates active learning. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 302–306, New York, NY, USA, 2005. ACM.

[10] M. J. Lutz, J. F. Naveda, and J. R. Vallino. Undergraduate software engineering. *Communications of the ACM*, 57(8):52–58, Aug. 2014.

[11] D. S. Yeager, M. D. Henderson, D. Paunesku, G. M. Walton, S. D'Mello, B. J. Spitzer, and A. L. Duckworth. Boring but important: A self-transcendent purpose for learning fosters academic self-regulation. *Journal of Personality and Social Psychology*, 107(4):559–580, Oct. 2014.