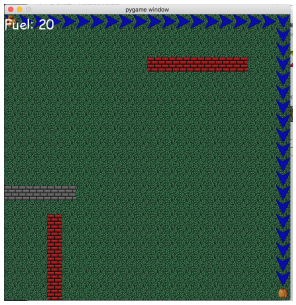
Teaching Statement

Kristopher Micinski (kris@cs.haverford.edu)

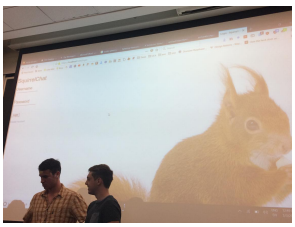
I enjoy teaching because it challenges me to motivate and articulate concepts in simple ways. Over the last few years, I have taught five unique courses in a variety of areas (mainly security and languages) and at every level of the undergraduate curriculum (including the introductory course). To me, effective teaching involves three core components. The first is challenging, relevant projects. Good projects force students to build their own mental model of concepts rather than accepting the one I present in class. Great projects also nurture an appreciation for core concepts by helping students achieve something they're passionate about. The second is effective lectures, largely drawing on in-class examples. I have experimented with a variety of active learning techniques, but in my experience they all approximate the same goal: get students mentally present when talking about hard ideas. Last, as all nontrivial learning involves repeated failure, I build support networks for students so that they may face that failure rapidly and confidently.

What Makes Effective Course Structure? When I design (or redesign) a course, I start by focusing on outcomes. I think carefully about a mix of both high-level and concrete skills I want students to learn, along with explicit non-goals. After that, I topologically order concepts so that as the course becomes more technical, each subsequent topic is clearly motivated by simpler ones. For example, in my accelerated introductory course (CS107 at Haverford) I cover hash tables. To motivate why hash tables are useful, I had students implement dictionaries (in class) using binary trees, before observing how hash tables trade time for space by allocating an array of hash buckets. In my security course (CS311 at Haverford) I took a slightly different approach, alternating each week between attacks and defenses. For example, the first week in class we covered buffer overflow vulnerabilities. The next week we covered address-space layout randomization (ASLR), which makes it much harder to launch an attack on a buffer overflow. We continued with attacks and defenses, becoming more elaborate until reaching the state of the art. Students remarked that this structure kept the course exciting.

What Makes Projects Effective? Through my lectures, I attempt articulate the importance of big ideas in computer science. Good projects force students to plainly see the value of those ideas for themselves. Projects need to be challenging in the right ways, since to be effective they have to challenge students to make some conceptually nontrivial extension to a technical idea. But they also have to be structured to avoid students getting stuck on minutiae.



When I design projects, I aim for a few things. First, I want to make them fun. Students are naturally intimidated by new concepts. A fun project often motivates them to consider the value in the new idea. For example, my introductory data structures course has students implement various components of a tile-based videogame, HaverQuest (example on right). One of my projects has students design a doubly linked list and subsequently use that implementation to build paths through various levels. Next, I leave enough breadcrumbs for students to clearly see why the concepts from the class can be used to solve some concrete problem.



Along with teaching core concepts, I believe projects should also develop students' general maturity in programming and systems. I teach this by having students use production tools like git, making them write unit tests as part of their projects, and providing command-line scripts to perform grading. While I strongly believe this is the right approach, it requires care to ensure I don't wall out students who might get intimidated by these tools because they haven't seen them before. To mitigate these concerns I spend some class time to teach these tools and emphasize to students that this is useful trivia, but not a reflection of students' core abilities.

Last, I often structure projects so that—when done correctly—they can be used in students' portfolios. For example, in my security course (CS311 at Haverford) student groups built a chat server and client with end-to-end encryption and web interface (example shown on right). Several students have told me this project helped them either secure internships or built skills they subsequently used in their jobs.

If you have time, I sincerely hope that you'll spend some time going over some of the projects I have written and judging for yourself.

What Makes Effective Evaluation? When I began teaching the hardest aspect was ensuring that students were evaluated and graded in a fair way. Over the courses I have spent teaching, I have pulled away principles that seem to work well for me. First, I make course grading split roughly between exams and projects. I occasionally assign group projects. When I do, I grade groups on their collaboration abilities by making them write up a collaboration plan (detailing how they will meet to ensure everyone is participating) and stick to it. This has been extremely effective at ensuring all students are able to participate in group work. When appropriate, I include one coding-based exam, in which I test both programming, testing, and debugging abilities.

What Makes Lectures Effective? When I started teaching, I used a fairly traditional lecture style: I would start each class by posing a problem (e.g., how to perform insertions into a collection in $O(log(n))$ time) and evolving a narrative to solve that problem. This style made sense to me, but it wasn't effective. First, I noticed that some students would get hopelessly lost after getting stuck on some nuance I didn't anticipate. Next, most students took significant ramp-up time to context switch into class mode. Last, students often think they understand things in lecture before attempting the projects on their own. To remedy this, I experimented with a variety of active learning techniques to overcome these issues. While I don't practice any particular style of active learning, I rely on the following principles:

- Force students to "check in" early in class. Confronting new ideas takes significant mental energy, and switching into learning mode takes more time than I previously assumed. I do this by asking a warmpup question early in class and having students work in groups to answer.
- Place in-class examples at periodic intervals throughout lectures. My experience is that once students begin to get confused, they quickly check out, sometimes for the rest of lecture. Period in-class examples give students time to check back in and ask questions without the pressure of interrupting a lecture.
- Have students work in small groups or pairs. Being wrong is one of the most important parts of learning. I find that working in groups helps foster an environment where being wrong is beneficial to everyone, rather than a competition to answer my question correctly.
- Instead of punting the hardest concepts to projects, make students confront those concepts during lecture.

My lectures now consist of a mix of slides, board work, and live coding. I have been surprised at how much students prefer slides, so I often make them available to study from. I have also observed that having students watch me write code doesn't seem to work well, so I often provide starer code (via git) and have students collaboratively write code during class. For example, in my accelerated introductory course I demonstrate a persistent version of binary tree insertion and then have students write an imperative version. I am often surprised at the questions students ask as they write code during class—often revealing crucial confusions—and have found this extremely valuable at getting students to engage with technical content.

Building Support Networks for Students Outside of Class I work hard to ensure students feel supported outside of class. This is challenging because students are highly nonhomogenous, both in their background and perceived abilities, but also in their individual approaches to problem solving and willingness to ask for help. In my experience, the most important aspect of connecting with students is being aware of problems early.

I work to enable this communication in a variety of ways. First, I always provide some online help for students. I often use a mix of a forum (such as Piazza) and instant messaging (Slack). I work hard to answer any question extremely promptly (often within minutes) and—while I do take care to ensure students don't rely on this as a crutch—I have found this makes students much more open to coming me with issues.

I provide flexible office hours, often via Skype. This has been crucial for students who work during my normal office hours, but also students at Bryn Mawr College, who are in many of my classes. Beyond this, I check in on student grades frequently, and email students who appear to be getting lost in the course.

Course Areas and Preferences I would feel comfortable teaching throughout the curriculum, but would be specifically excited for courses in security, systems, and programming languages / compilers. I would also be interested in teaching graduate-level courses on advanced security concepts and static analysis (perhaps with a focus on security auditing). I see this as a promising way both to contribute to the department but also recruit PhD students in my area.