Teaching Statement

Adam Blank

Because course policies can have such a profound effect on student learning, I always begin by ensuring my course policies are thought out and justified. I believe that every policy must have a rationale—and that these policies must respect students as people with their own goals. I frame the styles I use to teach with the learning outcomes of my courses. I try to help students feel comfortable and motivated, so that they are more likely to actively participate.

Over the past N years, I have reflected on common misconceptions and difficulties that students have when learning data structures, algorithms, discrete mathematics, and computability theory. I've immersed myself in the practice and research of teaching. But most importantly, I've helped shape how my students view computer science, and I've learned just as much from them in the process.

Learning Outcomes

Before designing or teaching, I've found it absolutely imperative to think hard about what outcomes I am fostering in my role. Like many other scientific disciplines, the best way to learn computer science is by understanding the *why* and the *how*, in addition to the *what*.

As a computer scientist, one of the most fundamental tools I have is **abstraction**—that is, separating the underlying ideas from specific examples and instances. I try to help my students learn to use abstractions both by presenting them myself and guiding students to the general ideas. For example, some introductory data structures courses teach the idea of a *tree rotation*. Sometimes, this idea is presented as a chart of the "four cases", and students tend to memorize them. When I've taught tree rotations in the past, I've used an analogy in which students can re-derive the "four cases", by pretending to lift up the node of the tree being rotated and imagining the effect of gravity.

As I teach a variety of lessons in a course, I take care to make **connections** to (1) the background the students already have and (2) previous and future topics. When I teach introductory discrete math courses *for computer science*, I take the *for* very seriously. Because programming courses are usually a pre-requisite, I can take advantage by drawing many connections between the math we are doing and programming.

One major misconception when students first see quantified statements is always variable *scoping*. Students will often write arguments like: "We know that $\forall x.\ x > 5$; so, x cannot be 5." Luckily, they already understand the scoping of functions; so, I can draw a connection to a piece of (Python) code:

```
1 def f(x):
2    return x > 5
3 print x != 5
```

Students know that the `print` statement doesn't make any sense, and this analogy helps them see that the argument they made doesn't either.

Over time, it's become clear to me that the learning outcomes of the courses I'm involved in are rarely limited to the topic of the course. Courses have **hidden agendas**, and I always try to understand what they are and deliberately align my teaching style with them. Some re-occurring favorites include: learning to learn, learning to work in a team, learning to deal with frustration, and learning time management.

15-251, a second semester CS core course at CMU, nominally surveys discrete math and computability theory. This course is traditionally very difficult. The assignments are extremely hard, and the quick pace makes students feel like they are always behind. As I TAed this course, it became clear to me that

none of the various topics were the main learning outcome. By assigning such difficult problems, 15-251 forces students to work together on the homework. For many students, these problem sets are the very first time they are forced to leave a problem unsolved. To support these hidden outcomes, I helped students find groups for the homework, held extra "conceptual office hours" where I gave students a second chance to see the material, and made individual meetings with students whenever they asked. I also *allowed them to struggle* at times rather than immediately providing help.

Teaching Style

In addition to ensuring I have a good understanding of the learning outcomes before teaching, I ask myself several questions to motivate *how* I teach. I continually re-visit these questions as there are always more nuances to find.

Who are my students?

When deciding how to present material and which techniques to apply, I first do some research into the students themselves. What is their background? Are they more like high schoolers? Or college seniors? Do they like Pokémon? Or sports?

Once I have a preliminary understanding of who my students are, I make **connections** between the goals and expectations of the course, and their background and needs. For instance, I designed the course 15-131 (on basic UNIX tools, debugging, and bash scripting) to help the freshmen at CMU learn how to use the tools they were already expected to know by their second and third CS courses.

Unfortunately, it's very difficult to predict the background of students perfectly. I handle this issue by frequently asking the students for feedback on the course, my teaching techniques, and assessments. If I find a mismatch, I make drastic changes as soon as possible to make **connections** between the students and the pedagogy, if necessary. This takes on many forms: adding late days, adjusting pieces of projects, re-ordering topics to find an extra 20 minutes to give to something difficult, etc.

When I teach large courses, I often end up with large (15+ people) course staffs. I apply these same methods in rough situations with TAs as well. In the 400+ person CS 2 at UW, there are routinely more than 20 TAs. We ask a "poll" question at the beginning of every meeting, and I often use this information to make changes within 20 minutes of the poll as necessary.

What does education research suggest?

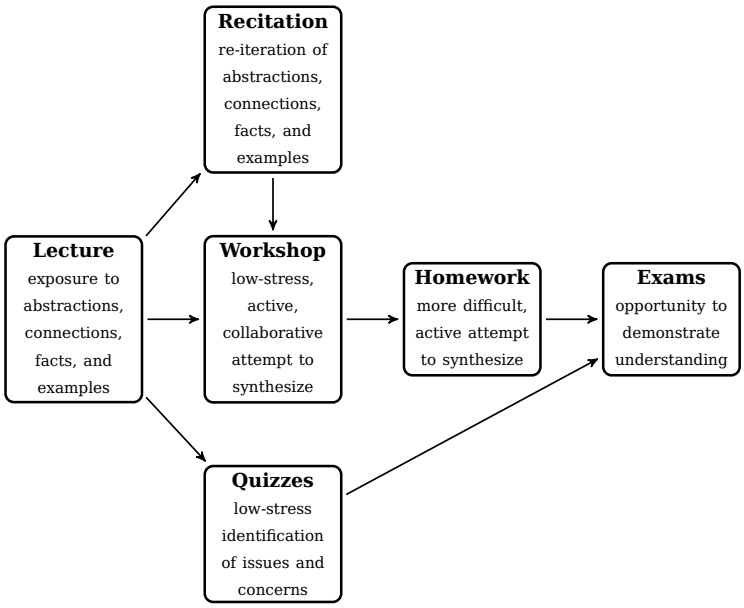
One of the best sources of teaching techniques is in the education literature. I try to follow best practices in my courses.

To support the **hidden agendas** of my courses, I often ask the students to *work in small groups* during lecture, in recitation, and on the homework. Working in groups often allows students with varied experiences to share, and it prepares them for the real world where they will need this skill. In my discrete math course, I went as far as to replace one third of the lectures with *workshops* in which students would solve problems in small groups and the TAs and I would roam around the room helping them and giving them instant feedback on their work.

To help students better understand their strengths and weaknesses, I try to give students "quizzes" which they get feedback on, but are not factored into their grade. These "quizzes" provide students with a low stress way of testing their understanding. I try, as much as possible, to make the "quiz" questions mirror standard exam questions—so they are study preparation as well.

How can I align assessments with the learning outcomes?

In order for students to get a deep understanding of the course materials, they need to be *exposed* to it in a variety of ways, they need to *actively work* on problems, and then they need to form bridges between the various levels of **abstraction**. In my teaching, I provide the students with multiple opportunities (lecture, recitation) for exposure to the material. Then, I provide them with opportunities to practice (writing proofs, designing algorithms, solving problems) with instant feedback (workshop, quizzes). Armed with exposure, practice, and feedback, I ask the students to synthesize the **abstractions** by working on their homework. Ultimately, I check their conceptual understanding on exams by giving them more time than should be necessary for every exam.



How can I use technology to supplement learning?

My research focuses on using various technologies to support learning. So, whenever possible, I try to supplement or enhance the courses I teach using technology.

In all of the discrete math courses I've taught, we assign students lots of proofs to give them enough practice with the concepts. Unfortunately, this grading is incredibly time-consuming for the TAs, and students didn't always receive their feedback in a timely fashion.

To combat this, I write tools to (1) eliminate human grading where possible, and (2) make human grading more consistent and fast where possible. These tools range from web applications that allow a number of "attempts" to a system which exploits the redundancy of grading to make it faster.

I also write tools to help students when we can't. In our CS2 course, many students struggle with visualising data structures and recursion. We traditionally use an IDE called jGRASP which allows students to see what a data structure looks like in isolation. Unfortunately, these visualisations don't capture how references work in Java or what stack frames look like. To solve this problem, I wrote an Eclipse plugin which gives them a visualization of the whole program as they debug it.

Conclusion

I believe that the most important impact people can have on each other is to exchange their perspectives. Teaching allows me to constantly share my experiences—in computer science and life—and help my students prepare for their futures. Every time I am able to guide a student to something new, no matter how trivial, the impact I have made is invaluable. Teaching is a balancing act between support and accidentally blocking discovery, and I absolutely love it.