Flipping Out in Computer Science

MARGO SELTZER

Margo Seltzer is the Herchel Smith Professor of Computer Science and the Faculty Director for the Center for Research on Computation and Society in Harvard's John A. Paulson School of Engineering and Applied Sciences. Her research interests are in systems, construed quite broadly: systems for capturing and accessing provenance, file systems, databases, transaction processing systems, storage and analysis of graph-structure data, new architectures for parallelizing execution, and systems that apply technology to problems in health care. She was a co-founder and CTO of Sleepycat Software, the makers of Berkeley DB, and is now an Architect at Oracle Corporation. She is a past President of the USENIX Board of Directors. She is recognized as an outstanding teacher and mentor, having received the Phi Beta Kappa teaching award in 1996, the Abrahmson Teaching Award in 1999, and the Capers and Marion McDonald Award for Excellence in Mentoring and Advising in 2010. Dr. Seltzer received an AB degree in applied mathematics from Harvard/Radcliffe College in 1983 and a PhD in computer science from the University of California, Berkeley, in 1992.
margo@eecs.harvard.edu

For a while, every conversation about education seemed to lead to the term MOOC (massive open online course). The hype around such courses seems to have died down to some extent, but MOOCs still exist and are largely good things, even if they have not fulfilled the promise of educating the world. However, there has been an unanticipated side effect to the (forgive me here) MOOC-ification of courses. We suddenly find ourselves in possession of some really high-quality teaching materials. What else might we do with such assets? I'd like to make the point that the wealth of online material opens up the possibility that those of us in the education business can undertake experiments in education that lead to deeper learning. In this article, I'll focus on the flipped classroom.

In 2013, I began revising all my undergraduate courses so that I could teach them in a flipped style (my graduate courses are typically research seminars, so in some sense, they are already flipped). But what is flipping? The high-level idea is that rather than spending class time absorbing information and then practicing use of the information at home, we flip those two activities around. Students use prepared materials at home for first exposure to new concepts and then come to class and work in small groups to practice applying those concepts.

I had been intrigued by the idea of flipping for a long time but hadn't quite figured out how to apply it to my own courses. My problem sets are large monolithic projects, not something on which one can make meaningful progress in a class period. So while I could easily imagine preparing materials for them to review at home, what would I have them do in class?

By pondering that question, I realized that one of the biggest challenges students face in programming courses is connecting new concepts to the programming tasks we give them. Maybe I could use in-class time to more effectively connect conceptual material to programming pragmatics, so students would not have to struggle with the question of how to get started.

My first experience flipping a course was with my (insanely time-consuming) operating systems course. Students report spending 30 hours per week completing the long but rewarding problem sets—students start with a simple operating system kernel and build user-level processes, a virtual memory system, and a journaling file system. I blogged my first experience flipping it here: http://mis-misinformation.blogspot.com/2013/08/an-index-to-my-flipping-blog-postings.html.

I ended up using three different styles of in-class exercises: gaining familiarity with the course software, completing problems that demonstrate mastery of the material presented, and engaging with open-ended design problems. I'll give short examples of each of these approaches.

Infrastructure

Traditionally, the first assignment in the course includes instructions on how students acquire the course software, install a virtual machine, configure and build a kernel, attach the debugger to a running kernel, etc. Small glitches in this process can result in students wasting a lot of time without learning much. Instead, I had them get their hypervisor licenses and install the course VM as pre-class work and then used class time to let them config and build their first kernel and complete some debugging exercises.

There were a number of positive outcomes from this structure. First, if students encountered any problems, we fixed them within a few minutes rather than having students beat their heads against the wall for hours. Second, it's actually pretty exciting to build your first kernel and watch it run. We got to all experience that together, so by the end of class there was a shared sense of accomplishment. Third, while we always encourage students to read code (and we assign them code-reading questions), as we wandered around the room interacting with the groups, we could ask questions that required that they look at code and could then gently walk them through how to approach a new code base.

Problem Solving with Virtual Memory

It's pretty easy to assume that once you've explained the four-level page table structure of the x86, students would then understand how address translation works. You would, however, be wrong.

Historically, when I taught VM, I would have the class "play MMU" and perform address translation one step at a time, having each student contribute something. This wasn't bad, but a lot of things fall through the cracks. With flipping, after introducing students to the concept of virtual memory and the x86 VM system, it was easy to create short problems that let small groups of students "play MMU" and translate addresses, draw page tables, populate the page tables, deduce what page faults really are, experience a segmentation violation from the point of the MMU, etc. Instead of each student contributing a tiny piece (and sleeping through the rest of the discussion), every student was exposed to every operation; by the end of class it was pretty clear that there was a much more uniform and deep understanding of what was going on.

Design Exercises

As the semester progresses in my operating systems classes, more of the conceptual material involves helping students develop the intuition and skills to design software and make tradeoffs. Prior to flipping, I would always present alternatives and let the class come up with the advantages and disadvantages of the different approaches. Of course, the five students who knew exactly what was going on were the ones who would pretty much answer all the questions no matter how much I cajoled the rest of the class and tried not to call on the frequent contributors. I converted these to small design exercises, requiring groups of two, three, or four students to assess tradeoffs, and then we'd come together as a class to compare answers.

As a result, everyone felt they could contribute. Even if they hadn't been entirely comfortable with the material, after discussing it with their peers for 10 or 15 minutes, they usually could effectively compare their conclusions with those of other groups. I've done a large variety of different activities around this theme ranging from peripatetic design reviews (when the class was small), to design debates, to collaborative analyses. One former student reports that she uses the skills learned in these exercises every day in her job.

I'm completely hooked on flipping at this point. I distilled the advantages I see in the approach into the following 10 bullet points:

1. It's good for an old dog to learn new tricks. This is really about making sure your teaching doesn't get stale. It's way too easy to keep teaching the same thing over and over again. Whether you use new pedagogy, new technological breakthroughs, or just good self-discipline, it's important to keep classes fresh.
2. Flipping lets me spend time with those students for whom the material is most challenging. This is so obvious in retrospect, but so exhilarating in practice. I have always run a relatively interactive class, but for the most part, the students who ask and answer questions in class are the ones who need you least—they are typically the most confident and are not struggling to understand the material. The silent ones, meanwhile, are frequently struggling, and the time spent helping these students in small groups during class time is incredibly useful.
3. Learning takes place by doing, not by listening to me. There are a lot of different styles of hands-on learning, but I think this point cannot be emphasized enough. Learning is not just the process of transferring information from the teacher to students; learning is about gaining new information and knowing how to use it, and the latter requires practice.
4. Teaching assistant engagement is critical. We call our teaching assistants "teaching fellows," or TFs for short. Flipping effectively requires a good staff that is comfortable engaging with students, walking them through problems, and posing the right questions. I am extraordinarily fortunate to have a truly amazing and dedicated teaching staff.
5. It takes a lot of effort to come up with effective in-class work. It's important that the in-class exercises or problems relate both to the concepts the students are learning and to the homework or problem sets they will be doing. Designing these exercises so they can be completed in the time allotted and add real value to the course is demanding.

6. Pre-class Web forms are AWESOME. They allow me to engage with students in an entirely different way and to gather lots of interesting data. This is perhaps the best surprise of all! I used Google Forms to have students submit answers to the pre-class questions. This created a mechanism I could use to obtain all sorts of useful information, including how things were going in partnerships, how much time people were spending on various parts of the assignment, what was working for students, what wasn't working, etc. Once you have students regularly filling out forms, they will answer anything you put there, and you can use that to make the class better. Score!
7. My operating systems course, CS161, is even more time intensive than I thought. I had been saying 20 hours per week for decades; when the going gets rough, students were regularly reporting 30-hour weeks. Oops.
8. It would be useful to help students learn what it really means to design something. Software design is really hard! We spend a lot of time in class doing small group design exercises—I could imagine developing an entire course around this idea.
9. Flipping is a great equalizer when students enter with different experience levels or exposure to different topics. It's relatively easy to provide supplementary material as pre-class work, so that students who have gaps in their background can catch up.
10. Fully integrated and coordinated materials take real effort but pay off tremendously. This should be a no-brainer, but thinking deeply about the relationship between the videos I prepared, the exercises we completed in class, and the problem sets was time well spent.