Greg Durrett  
Teaching Statement

One of the most enjoyable things I have done in my graduate career has been my teaching, especially holding sections and office hours. Getting to personally know students and guiding them through the challenges of the material is a rewarding experience. Being at a large state school with exploding CS enrollments, I have also gained an appreciation for a less obvious aspect of teaching, namely the value in carefully designing and managing a class so that it runs smoothly at scale. I’m going to talk about lessons I have learned from my experiences teaching project-based classes and how these impact my views on course design. In a large class, projects need to be streamlined so that students learn the essentials and don’t waste time struggling with irrelevant details, or else the students will suffer and course staff will be overtaxed trying to deal with an onslaught of questions. Moreover, upper-level class projects mix theory and practice, so students will have bugs ranging from misunderstandings of complex algorithms to missed indentation in their Python code. This means that students need a range of ways to engage with the course staff and other students in order to resolve the problems they might have.

Designing effective projects

An effective project should have three elements: scaffolding, so that the units of the work are isolated and made clear; checkpoints, so students have guidance on how to debug their code and verify results; and flexibility, so students have the ability to be creative and are not just transcribing pseudocode.

Scaffolding  Among the most important considerations when designing a project are how the students’ deliverable work will be isolated, what supporting code they will be given, and how the project documentation will point students in the right direction. I got experience making different choices on each of these fronts when I TAed Dan Klein’s graduate level natural language processing (NLP) class at UC Berkeley. We substantially overhauled the set of projects for the course: out of five projects that we assigned during the semester, I built two from scratch and we significantly modified the content for a third. In the context of this graduate class, whose stated purpose was to equip students to do cutting-edge research in NLP, we wanted students to build complete systems on their own. To that end, we shielded them from busywork like file I/O, but gave them free reign to make core design choices in their systems.

One crucial factor in making these open-ended projects successful was a project’s framing, which should ideally guide students towards a working system and warn them about possible pitfalls. When I took the class prior to TAing it, the n-gram language modeling project was extremely difficult for many students due to the need to store millions of key-value pairs, which is important to know how to do but not the emphasis of the class. When we subsequently redesigned the project, we preemptively provided tips about how to design scalable data structures. This forewarning removed one of the major pain points of the project while still giving students the experience of building a real system.

Checkpoints  Large projects should also have natural checkpoints, to encourage students to tackle a problem in pieces and let them verify that each piece works in isolation. This was a lesson I dearly needed to learn upon entering grad school: for one of my first independent research projects, I coded up a complex machine translation model over the course of several weeks, finally worked out all the compilation and runtime errors, ran it, and promptly hit another set of even more insidious logic errors that required meticulous debugging. In both research and upper-level CS classes, problems with systems can be very subtle and are not discovered by simply running one integration test. Providing structure to a project helps gently guide students toward this realization. For the NLP class, we required students to first implement simplified versions of techniques. In a machine learning assignment involving discriminative models for parsing, students implemented the perceptron before moving on to more complex learning
algorithms. The end result of this assignment structure was that students understood each component of the project individually as well as their integration, but were not confronted with an insurmountable challenge from the start.

**Flexibility** Finally, a good project should have some flexibility so that students are forced to think critically about open-ended choices. In the NLP class, we asked students to extend every project in some way, be it making a more complex model, speeding up inference, or doing error analysis. Students could therefore delve deeper into the aspects of the class that interested them the most, just like in real NLP research. An open-ended component of a project is even more essential when projects are more limited in scope; such was the case for the undergraduate artificial intelligence class that I TAed. In that class, I observed that students would start to gain familiarity with the core concepts in section when I would present them in detail and work through examples. But having to actually make a choice about how to extend an algorithm or apply it to a particular problem forced them to understand it more completely. This meant that students could still learn and achieve something interesting without building big end-to-end systems.

**Supporting student learning**

Beyond having well-designed projects, a successful project-based course must provide students with the resources they need to tackle the assignments. Conventional outreach tools like TA office hours partially do this, but I believe technology can enable us to do even more. One great tool for connecting with students is an online discussion forum like Piazza. For the undergraduate AI class, which had around 350 students, Piazza was absolutely essential. It provided a centralized archive of questions and answers, which let me devote my office hours to resolving conceptual misunderstandings rather than answering the same minor questions repeatedly. Piazza also ended up being the go-to resource for students with low-level coding issues. When someone happened to be misusing Python dictionaries and could not figure out why their code was crashing, more often than not another student would quickly chime in because they had just worked through the same bug. This simultaneously provided an outlet of engagement for those who had already mastered the material and helped students who had less hands-on programming experience.

Another useful tool is autograding. For the undergrad AI class, autogrders were an invaluable tool for scaling the class to serve hundreds of students. They provided a way to give feedback on projects without requiring project write-ups, which would have been hard to grade consistently in such a large class. That being said, autogrders must be used carefully: I watched students in office hours engage in “stochastic debugging” by tweaking their code until it passed the autograder rather than thinking more deeply about what was wrong. One solution to this problem that we implemented in the NLP class was to make the autograder test each student’s model on a substantial amount of data, making it too slow to run repeatedly. Ideally, an autograder would provide feedback in the same fashion as a teacher, giving semantically-informed hints to help a student work towards the correct answer and better understand the concepts. The fact that source code is a structured object lets us categorize errors in a regular way, and has exciting potential for implementing more tutor-like automatic feedback. I would be interested in designing projects or problem sets like this for any kind of class, and I think this is an opportunity to help bring computer science classes to more students as well as better serve our current ones.