

Teaching Statement

Daniel Ellard

The opportunities to teach and work with students and to develop new educational materials and techniques are my primary reasons for seeking an academic career. My academic background, teaching experience, and career as a software developer make me well-prepared to teach introductory programming and Computer Science courses as well as more advanced courses on operating systems, distributed systems, databases, software engineering, and data structures and algorithms.

Over the last decade, I have dedicated much of my time to teaching and developing courses and educational materials. As a result, I have acquired extensive teaching experience at the undergraduate and graduate level as an instructor, head teaching assistant, and teaching assistant for several courses at Harvard. More importantly, I have been a central figure in the development of several of these courses and I have created teaching materials that have been adopted by many courses both within and beyond Harvard.

Teaching Experience

In 1997 I designed and developed my own course on data structures and algorithms and taught this course for two years in the Harvard Summer School. This was a great experience because I was able to explore every aspect of running a course, including creating a syllabus, choosing a textbook, hiring and supervising teaching assistants, developing assignments and supplementary materials, and writing and presenting lectures.

In 1996 I created Ant, a new microprocessor architecture, for use in CS50 (Harvard's introduction to Computer Science). Ant is simple, so that it can be mastered quickly, but it is also realistic and full-featured, so that it can be used to illustrate almost any aspect of machine architecture. The assembly language tutorial for Ant was less than half the length of the tutorial we had used for a different processor in previous years and required fewer lectures to explain, yet covered all of the same topics. Over the next several years, the Ant project received grants from the NSF, Microsoft, and Apple Computer to extend and package Ant for use in nearly any course that deals with machine architecture, including courses on compilers and operating systems. Ant is currently in use at several schools (as well as by three courses at Harvard alone). All Ant materials are freely available via the project web site (<http://ant.harvard.edu>).

From 1994 until 1996, I served as a head teaching assistant for CS50. During those years, the course enrollment nearly doubled, reaching more than 400 students. The size of the teaching staff increased proportionally and it became difficult to find enough experienced teaching assistants and ensure a consistent quality of instruction across the different sections. To address this problem, I initiated a training program for all teaching assistants. This program is still active.

Before becoming a head teaching assistant for CS50, in 1993 I helped to completely overhaul the course, migrating it from PASCAL to C and extensively updating the curriculum. My primary responsibility was designing and creating a new suite of assignments (including scaffolding code, solution sets, test suites, grading standards, and other supporting material) to match the new curriculum. Many of these assignments are still in use ten years later.

As another example of my initiative, in the Spring of 1999, when Harvard's graduate-level course on distributed systems was canceled (due to unavailability of a professor to teach the course), I organized and led a seminar on distributed systems for which the participants were granted full graduate credit.

Teaching Philosophy

The most important thing that we can teach our students is that Computer Science is interesting, relevant, and fun. A student who is curious and interested in the subject is easy to teach, but unfortunately not all students arrive in the classroom in this state of mind. It is the responsibility of the teacher to present the subject in an interesting and engaging manner that shows the elegance and beauty of Computer Science as well as its applicability to solving concrete and real-world problems and to nurture each student's latent desire to learn.

It is also important to teach students how to approach the subject. This is especially true for introductory-level courses. Introductory Computer Science is somewhat unusual in that we expect the students to learn general concepts such as abstraction, functional decomposition, and object-oriented design, while at the same time requiring that they internalize the syntax and often archaic details of their first programming language. It is the responsibility of the teacher to balance and differentiate these two tasks, so that the students always understand that the programming language they are learning is merely one of many ways to express the higher-level concepts. New language constructs or programming techniques should be introduced to illustrate and implement higher-level concepts, and never vice versa.

In higher-level courses, I believe that the emphasis should be on collaboration, planning and design, and critical review. Complex scientific and engineering projects are rarely the work of an individual; students must learn to organize and work as teams as early as possible. Similarly, planning and design are essential elements of any large project, but are difficult skills to master (but are often given little weight in the curriculum). Finally, the ability to intelligently and objectively critique the work of others is a necessary precursor to being able to accurately evaluate one's own work.

Teaching Style

My personal style of teaching is based on the following principles:

Engage the students. Students must be active participants in the learning process, rather than passive observers. This is particularly important for lecture courses.

Establish fair and clear grading policies. Despite our best efforts to inspire students to learn simply for the joy of learning, there will always be many students who focus primarily on whatever aspects of the material they believe will result in their receiving a good grade. However, this is not always a bad thing – the proper grading and assessment policies can guide these students to focus their attention on the essential points.

It is also important that grading policies be fair and relevant to the objectives of the course; few things are more discouraging to students than receiving a low grade for work that they believe is good. Grading standards must also be flexible so that unconventional or original solutions are not penalized simply because they do not match the anticipated solution.

Set clear and realistic goals. Students respond best to goals that are both challenging and achievable. For example, extremely easy assignments are boring, allow students to become careless, and do not give the students any sense of accomplishment. In contrast, excessively difficult assignments are frustrating and intimidating. Unclear or ambiguous assignments are even worse because the students are apt to waste their time solving the wrong problem (and justifiably resent the poor marks they receive when they fail to correctly guess what problem they were supposed to solve).

Identify and fix misconceptions early. Once a misconception takes root, it is difficult to remove. Waiting until the next assignment or test has been graded to discover that students are confused is a grave mistake.

Let the students make mistakes. Learning what doesn't work is just as important as learning what does. Students learn more from understanding *why* an incorrect answer is wrong than from simply memorizing the correct answer. Experimentation is essential to education; students must be encouraged to learn from their mistakes.

In fact, I even encourage my students to make mistakes. For example, I show them that adding intentional syntax errors to a working program is a good way to learn what error messages the compiler will generate in response so they can recognize those messages when they see them in the future.

Always respect the students. A teacher must respect the goals, needs, and individuality of each student and help each student do his or her best to achieve these goals. Not all students respond to the same methods, come from the same background, or have the same level of preparation. For example, one mistake that I made in my first year as a teaching assistant was to use metaphors and idioms that were meaningless to many of my international students. This is a very easy mistake to make and quite awkward to repair.

Teachers must also respect that students have other interests and engage in time-consuming activities outside of the classroom; there are limits to how much time students can reasonably be expected to spend on one course.

I have found that a good way to engage students and identify misconceptions quickly is to give very short quizzes during class following the discussion of each major idea. These quizzes are ungraded because they primarily test how well I am explaining the material and holding the attention of the students, and it is inappropriate for the students to receive poor marks if my exposition is ineffective or boring. Even though they are ungraded, these quizzes do motivate and focus the students on the key points of each class because they know that I expect them to be able to answer the questions correctly. If they cannot, it is a warning sign that they have not mastered the material and will have difficulty with their tests and assignments.

One of the crucial elements of engagement is that students must have the freedom and means to tell the teacher when they are lost or confused. In addition to quizzes, I use short questionnaires to get immediate feedback about the reading and my lectures. The key questions I ask are "Did I explain anything in a way that you found unclear or confusing?" and "Are there any topics you would like me to revisit during the next class?" Unlike my quizzes, which I try to incorporate into every class meeting, I use these questionnaires primarily when I am trying a new set of examples or teaching a subject for the first time.

Course Development

I consider creating new assignments to be the most interesting and rewarding part of course development, as well as the most challenging and important. Good assignments must be interesting and relevant in order to engage the students, and they must match the ability and background of the students. Assignments must also be written in a manner that explains clearly and unambiguously what the students are expected to do and how their answers will be evaluated. In most cases, assignments must also provide some amount of guidance about how the concepts the students have been learning can help them to do the assignment.

Similarly, I have found that the best way to organize and structure my lectures and writing is to begin by asking myself what questions I want to enable the students to answer. I usually begin with a very specific question (e.g. "How can I prove that the complexity of mergesort is $O(n \log n)$?"), and then generalize (e.g. "What general methods are there for solving the recurrence relations that describe the complexity of divide-and-conquer algorithms?"). I then attempt to answer the question, using only knowledge and intuition that I expect the students to have. Each false start, stumbling block or impasse is a topic I must address in lecture or the reading. At the same time, however, I am always careful to remember the higher goal of teaching, which is not to prepare the students to answer the questions we pose to them today, but to answer questions we have not yet imagined.