My Approach to Teaching

Edward M. Reingold
Department of Computer Science
University of Illinois at Urbana-Champaign

January, 2000

Some Background

A bit of personal history is needed to explain my approach to teaching computer science, especially theoretical aspects of computer science. In high school in the early 1960s I became addicted to computers—all aspects of computers then current: software, hardware, and numerical methods. In the middle 1960s, I earned money in college doing programming for a wide-range of numerical and non-numerical applications, wiring plug boards for accounting machines, and operating mainframe computers for the computer center. I maintained compilers and assemblers, did minor hardware repairs, and worked as a drudge sorting cards. In short, I spent my time from 1962–67 working in the real world of computers. However, for a variety of reasons, while in graduate school I started working in theoretical computer science, and that is what my career as a professor has focused on.

How I Teach Theory

But although I have a decidedly theoretical bent, and teach mostly theory courses these days, I try never to teach even the most theoretical topics without motivating them with real-world applications. The courses I teach most often are CS 273, the introduction to theoretical computer science, and its successor course CS 373, analysis of combinatorial algorithms. Both of these courses require significant use of mathematical techniques drawn from combinatorics, probability theory, mathematical logic, and real analysis. CS 273 must introduce these techniques and CS 373 must use them—the challenge is to motivate students to understand them! Theoreticians study such material for its intrinsic beauty and subtlety, but it's not reasonable to expect undergraduates to be satisfied with that justification. So, I never introduce any technique, no matter how abstruse, without beginning by trying to solve a real-life computer programming or architecture problem and then seeing how we run into a wall of problems that require the techniques I intend to present.

CS 273 includes material on combinatorial reasoning, probabilistic reasoning, recurrence relations, elementary algorithmic techniques, and an introduction to automata and formal languages: all dry-as-dust topics when presented in isolation. However, each of these topics is incredibly useful in dealing with the real-life programming/hardware design problems the students face in their other courses and will face in their careers. So to catch their interest, I talk about problems of data structures, compilers, text processing, and others from diverse sources so the students can see the theoretical techniques *in situ*—applied to problems that will be of interest to them. As with many pedagogical goals, this is easier said than done! But because of my out-in-the-field software and hardware experience (in fact, I have kept my fingers in the pot of software design with some 15,000 lines of code that I wrote and maintain for the GNU Emacs system, along with a number of other software projects), I am able to draw on a rich collection of applications from my own experience.

Teaching CS 373 is both easier and more challenging. It is easier because by then the students understand the principles of algorithm analysis, but it is more challenging because the techniques and topics are harder. Moreover, since almost none of the students will need to use the complex data structures analyzed during the course, I view the goal of the course as being to change the way students think about the task of programming. I want them never again to write lines of code without asking themselves if they understand the way it will perform in practice, in the worst case, the average case, and in the best case. I want them to understand that "average behavior" is a subtle concept that cannot be adequately captured by running a program a few hundred times and recording the results.

How I Teach Programming

Early in my teaching career I taught mostly introductory courses, especially the first two programming courses. It cannot be overstressed that human beings have a great deal of trouble learning to think algorithmically. As Herbert L. Dreyfus and Stuart E. Dreyfus say in *Mind Over Machine*:

> The precision essential to a computer's way of manipulating symbols constitutes both a great advantage and a severe limitation. Since what the symbols in a computer represent must be absolutely precise, and the programmer must be absolutely clear in defining each symbol, the attempt to write a computer program inevitably exposes hand-waving, fuzzy thinking, and implicit appeals to what everyone takes for granted. Submitting to this rigor is an immensely valuable discipline.

It is a "valuable discipline" indeed, and a painful one. Too many teachers and textbooks approach presenting this material with problems that introduce the material, but are outside the student's ken. In teaching introductory courses, I went to great lengths to use problems drawn from the students everyday lives; this enabled them to focus on the algorithmic thinking without having to simultaneously learn or recall distracting issues. For example, asking students to print a calendar for a given month and year requires them to learn no new material except the algorithmic/programming techniques needed. Similarly, translating a number into words for a check-writing program, simulating a digital alarm clock, implementing a Monopoly board, sorting and combining page numbers for an index entry, and so on, allow the student to concentrate on learning to think algorithmically.

Why I Teach

I love teaching and it suits my personality. I am enough of an exhibitionist to love the performance aspect of lecturing, but shy enough to let students participate. I am egotistical enough to relish the sense of accomplishment at explicating difficult material and making it intelligible, but humble enough to know that the ideas I am presenting are rarely original with me and that I can learn much by listening to my students. I am bright enough to take pleasure in crafting interesting homework and exam problems that go beyond the mundane request for students to regurgitate memorized facts, but dumb enough to have had difficulty learning the material myself so I can empathize with the students.

I started my teaching career in eighth grade by tutoring a younger boy who was having trouble in math. I still recall his joy as he mastered concepts that he thought were beyond his abilities, and the gratitude of his parents in finding that his difficulties lay not with him but with his teacher's limited ability to explain the concepts. I continued tutoring throughout high school and college, always thriving on the satisfaction of seeing my tutee's eyes light up with understanding. In graduate school I resigned a research assistantship in favor of a teaching assistantship because I loved to teach. Even now, more than forty years after my first tutoring experience, I am thrilled at the prospect of making ideas accessible.