

# Statement of Teaching Philosophy

Ray Toal  
Professor of Computer Science  
Loyola Marymount University

All of us have an innate desire for a happy and fulfilling life. If we fully use our talents we can avoid some things that make us less happy. When we are shown we have talents that we didn't know we had, we hit an inner high. Good teachers fully use *their talents* to bring out *new talents in their students*. What could be cooler?

I teach to improve myself, improve my students, and improve the world — or maybe just the first floor of my building. I teach to satisfy my own need to serve, and feel the payback of this service on seeing students thrill when they master something new. Mentoring the students through four years of college and laughing with them about their incredible growth (yeah, remember that code you wrote as a freshman? — LOL) is beyond rewarding.



## Goals

My goal as a teacher is to go beyond the usual conveyance of "understanding", "critical thinking skills", and "lifelong love of learning" standbys; I want the students to gain a mastery of — or, more precisely, a fluency in — the things they want to do. Think of the world's greatest athletes, dancers, artists, and writers: their "work" has a beauty and elegance that is awe-inspiring. And they (think Jordan, Baryshnikov, Te Kanawa) perform with a naturalness and ease that seems to require no effort at all. I want my students to show that fluency: to speak articulately, to describe mathematical and philosophical concepts in elegant prose, and to bring to life beautiful code from the subconscious straight through the fingertips.



## Strategies

Okay, okay, so how? Andre Agassi batted tennis balls in his crib. Tiger Woods was golfing at age two. Lance Armstrong rode, like what, six hours a day? Students need practice to attain fluency. But they are brash young adults: they'll work only if it's fun and they see self-satisficing value in it. So I design work for them that seeks the right mix of (1) fundamental principles that enable learning, (2) skills that employers will be looking for, and (3) things the students perceive as cool (graphics, AI, open source, security exploits). I motivate them further by describing the lifelong dedication and training regimen of the great ones. Then I remind them "yeah, the *programming* jobs are getting outsourced, but not the jobs of (liberally educated) *computer scientists*." At least that gets their attention. If that doesn't work I order pizza for the class.

But wringing practice out of students is only one small part of my role as a teacher. Being effective as a teacher means being available to students, being concerned about

student's well being, advising, listening, mentoring, sharing experiences, and showing them how to get beyond where they find themselves at the moment. It means being prepared and knowing everything you can about the class. (Being entertaining and applying a little education theory doesn't hurt either.) Therefore, I:

- Stay available via IM an inordinate amount of time (often into the early morning) and encourage students to contact me any time I'm online (though I ignore them if they ask "how was your weekend?" — I'm not, and do not want to be, a friend!)
- Proactively cruise the lab asking students how their work is going, offering suggestions, and asking if they need anything. I've learned over dozens of years how to do this delicately, without making the students uncomfortable; I often get them to ask questions about things they might not feel comfortable initiating a discussion on by coming to my office.
- Take on a large share of supervising independent studies courses and Master's Theses. So many students have special interests that are not met by the regular courses; they appreciate the opportunity. The one-on-one time with these students is such a plus for both them and me.
- Whenever possible, take an apprenticeship-style approach. Perhaps the biggest influence on my teaching has been (Harvard cognitive scientist Howard) Gardner's *The Unschooled Mind*, which suggests the most successful models of education are apprenticeship relationships and children's museums. One apprenticing technique I use is sitting beside the student, programming together, passing the keyboard back and forth. Another is to give the students code and make them extend it: this naturally requires them to study and learn from my work and mimic it as they build on it. I've found this model quite successful. It gives me 50-hour work weeks, but doing it this way is still my passion after 19.5 years. (It makes me a better coder, too.)
- Teach practice before theory! Start simple. Use inductive examples. Use tons of examples. It's easier for students to generalize. Theory in a vacuum bores them.
- Make use of life experience. I use a fair amount of anecdotes from consulting efforts in the classroom. Students listen. Something about this "I need to get a job some day" feeling makes them perk up. I also segue into tangential topics (once a week at least). A few students think this makes lectures unorganized; most, though, are very appreciative of seeing how computer science relates to other disciplines and to the "real world."
- Am careful with the learning-from-mistakes technique. We all know confronting and correcting our mistakes makes for powerful learning. I often give assignments and exams which purposely hit on things a little beyond the student's expected abilities. Going over exams then exposes excellent learning opportunities: students see how to solve problems for which they had to attack in a pressure situation. This can result in low raw scores, so I always announce before handing out exams: "The purpose of these exams is for me to have a basis for assessment, not to make you feel good." I say this without being demeaning; they usually understand. And, when they do nail answers to questions beyond what they thought the limits of their knowledge, they *are* happy!
- Write tons of my own lecture notes and sample code. Students don't like textbooks.

- Take assessment very seriously. I grade all assignments and exams myself, and give written feedback. Sometimes this is difficult, since university courses generally test students' higher cognitive abilities and creative thinking skills (evaluation, synthesis, etc. — you know them.)



## Duties and Responsibilities

Besides the obvious duties a teacher has to the students, he or she also has duties to colleagues and to the university. Colleagues are to be respected, encouraged, and always treated as equals, regardless of discipline and experience. Pettiness is never acceptable.

Teachers have a duty to follow the mission of the(ir) university. LMU's mission statement mentions the education of the whole person, the promotion of justice, and the appreciation of other cultures and societies around the globe. I have been able to address these (and other) aspects of the university mission in my Computer Science teaching:

- *education of whole person* — when we program, we have to think of what we know and how we know it, and we can express it. I emphasize these deeply ontological and epistemological roots of Computer Science. (That's also why I like to suggest music theory and foreign languages as electives during advising.) I also pester the students to sleep and exercise when they are working too hard. And the compiler construction final is full-court basketball.
- *promotion of justice* — Programming is a service and LMU graduates are expected to produce products that can serve their users. I spend several weeks in my Internet class on the details of making sites and tools useful for people with disabilities (whether physical or mental).
- *"...around the globe..."* — I constantly strongly advocate (in upper division courses at least) the importance of internationalization and localization in software.



## The Future

One last responsibility is to keep growing as a person and a teacher. That usually means making a lot of mistakes and undoing the damage where possible. I've tried a couple experiments in courses that were perhaps a little disappointing (like the Spring 2003 Compiler Construction class where I let the students design their own language), but at least I warned the students in advance and we were able to get a good deal out of the experience. They learned how to design a language! (Oops, they were supposed to *implement* one.)

I'm working on continuing to improve; lately I've learned to listen better, for example. I still volunteer to invent and teach new courses. I take pride in my work. I stay current by limited consulting. But I know there are many great teachers I'd like to emulate and in whom I see skills I sorely lack. Perhaps after 30 more years of teaching I'll be okay.