

Some Theses on Undergraduate Computer Science Education

Shriram Krishnamurthi
Computer Science Department
Brown University

October 23, 2005

In lieu of a lengthy document, I want to make a few assertions that I hope will provoke discussion.

Peeves about the Present

Students seem to have great difficulty conceiving of programs in any form but as source code. They have difficulty presenting a program as a set of relational or object models, as call-graphs, as traces of temporal behavior, and so on. When we do introduce specification and modeling it is often presented in the most heavy-handed manner (e.g., UML). Do *you* wake up excited to write a UML specification?

Students are not being vested with a strong sense of software quality. By relying heavily on automated testing, courses past the introductory level set the wrong expectation: assignments regress into a game of trying to be good enough to escape detection of flaws by the course staff, not aspiring to absolute standards. We have to be willing to reject even perfectly working programs as inappropriate if they fail on other criteria.

We should stop confusing the teaching of algorithms with the teaching of program design. The 1990s taught us that even professional programmers were willing to trade some performance for structure and maintainability. Beginning students, in particular, need guidelines to progress from an empty page to a clean, working program. Neither algorithmics nor a pabulum like “top-down design” addresses this fundamental problem.

The emphasis on programs ignores the difficulty of designing their APIs. Programs die but their APIs live long after them. Therefore responsible programmers should tremble before publishing an interface. When’s the last time you taught your student principles of designing APIs?

We continue to propagate the nonsensical myth that “all languages are equal”. Programming languages are a human-computer interface. All languages are no more equal than all interfaces are equal.

We have slipped into a programming language orthodoxy. We must expose our students to a diversity of languages. A few weeks of superficial exposure in a programming languages course is unacceptably weak; judging as a consumer, it is often worse than no exposure at all. A day will come when (shockingly) even Java will cease to dominate; if the past is any indication, that day may even be when your current freshman class graduates.

Introductory courses expend too much effort combatting poor high school curricula. We have to become more involved in the processes that feed us. Our TeachScheme! Project has been engaged in this for a decade now, already training over 500 high school teachers. Join us!

Suggestions for the Future

Underspecify! Whether students proceed to industrial positions or to graduate programs, they will have to deal with a world of ambiguous, inconsistent and flawed demands. Often the difficulty is in formulating the problem, not in solving it. Make your assignments less clear than they could be. Do you phrase your problems as tasks or as questions?

Use failure as a teaching aid. Not enough courses set up students with the potential for (limited and gentle) failure. By helping students fail and then recover, we better prepare them for reality, give them confidence, and teach them how to extract positive lessons from failure.

Conduct a code-walk in an upper-division class. Treat the code as “guilty until proven innocent”. Watch how difficult it is for your students to articulate why their programs work. Be prepared to be disturbed.

Computer science will eventually fragment into two separate but complementary disciplines. We have a model for this in Mathematics and Applied Mathematics. For now, give your curriculum the flexibility to accommodate both kinds of students.

Computer science will profit from interfacing with the social sciences and arts. Look beyond traditional siblings like the physical sciences and engineering. Make your courses open enough to accommodate students with these broader backgrounds.

Computer science is a liberal art. If we were redesigning the trivium and quadrivium today, there is no question computer science—which embodies a new epistemology—would be in the middle of the action. Make sure at least part of your curriculum teaches it like it were one.