BLOG @CACM

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/2716345

<http://cacm.acm.org/blogs/blog-cacm>

Advice on Teaching CS, and the Learnability of Programming Languages

Valerie Barr considers how attitude can impact teacher effectiveness, while Mark Guzdial suggests the ultimate focus in teaching programming languages should be on usability.

Valerie Barr
**"Some Thoughts For
Computer Science
Teaching Assistants
(and Faculty)"**

<http://bit.ly/1yBDMbb>

January 4, 2015

I was recently contacted by a student who will be working as a teaching assistant for an upper-level CS course. He remembered that, during a visit to his campus, I had said that the way courses are taught is just as, if not more, important than the race, gender, ethnic background of the instructor. His question was whether I had any advice for him and his fellow TAs about how they could "conduct ourselves or talk about the course material in order to help foster as inclusive and welcoming an environment as possible."

Here is the advice I gave, which I think can be helpful for faculty as well as for TAs, lab assistants, student help desk workers, and others.

1. The most important thing is tone and attitude when answering questions. Many of us in CS (I include myself in this) have to fight against lapsing into a very patronizing tone. It is easy to slip into the "I cannot believe you don't understand that" tone of voice, or say things like "it is so obvious." Every time we use that tone of voice, or those sorts of phrases, we send the "if you don't get it already, you shouldn't be here" message. While this is likely more of a problem in lower-level courses, it can happen in upper-level courses as well. An interesting exercise for TAs and lab assistants would be to sit down and collect a list of the sorts of things they have heard over the years (or even said) and then actually say those things in a patronizing tone of voice. That should serve as a good reminder of what it sounds like, hopefully making it easier to avoid talking to students in that tone.

2. If students are working in teams, keep an eye out/ear out for problematic

dynamics amongst the team members. Suggest to teams ways of rotating responsibilities so that everyone has the opportunity to play a leadership role.

3. The electronic submission deadline for homework should be set so that it does not encourage macho behavior. Have homework due no later than 11:00 P.M. or midnight. The problem with a 6:00 A.M. or 9:00 A.M. deadline is that it encourages very macho "I stayed up all night, I'm hardcore, I'm a *real* programmer" behavior. People who do not handle their work that way are seen as not being a true hacker, as not having real "programming chops," even if the person who did not stay up was incredibly organized, completely solved the homework problems, and turned in the assignment hours early!

4. It is fine to suggest the use of Github or others, but understand that not everyone wants to put their work out in the world in that way. If it is required of all students, fine, but do not encourage the use of public repositories as a way of measuring people's abilities. I am always distressed when I hear that employers are grouping résumés based on what they find in code repositories—this automatically knocks out of the running anyone who does not choose to display their work in that way, and we know that women tend not to use repositories as much because of the level of attack they experience in that world.

5. Periodically look over the grades as a group and see if any implicit bias is in evidence. Are grade distributions roughly the same across all groups of students? Check in about who is com-

ing for help, how much time you spend with people. Are there some people who never come for help, but whose performance indicates they should? You cannot exactly go up to people and say, “what are we doing wrong, why don’t you come for help?” but consider using a quick survey partway through the term to ask students for feedback on the TAs/lab assistants/student helpers as a group.

6. Do not make a big deal about how you are doing so much and working so hard to be inclusive. Bragging about how sensitive you are will backfire and can make people feel singled out. Do it quietly, and trust that people are likely to learn more and enjoy the class more if they are feeling comfortable in the educational setting you have helped create.

I am sure this is not a comprehensive list. What do you do in your classes? What can you add to my list? Post your ideas to the comments. Thanks!

Mark Guzdial
“Programming Languages Are the Most Powerful, and Least Usable and Learnable User Interfaces”

<http://bit.ly/1g39mAX>

March 27, 2014

Andy Ko wrote a recent blog post (<http://bit.ly/1iVxF3A>) with an important claim: “Programming languages are the least usable, but most powerful human-computer interfaces ever invented.” Ko argues the “powerful” part with points about expressiveness and political power. He uses HCI design heuristics to show how programming languages have poor usability. Obviously, some people can *use* programming languages, but too few people and at great effort.

I see that his argument extends to *learnability*. There are two ways in which programming languages have poor learnability today—(1) in terms of expectancy-value and (2) in terms of social cost.

What is the benefit of a closure? Eugene Wallingford tweeted a great quote the other day:

Educational psychologists measure the cognitive load (<http://bit.ly/1ISmG0f>) of instruction, which is the effort that a

student makes to learn from instruction. Every computer scientist can list a bunch of things which were really hard to learn, and maybe could not even be imagined to start, like closures, recursion in your first course, list comprehensions in Python, and the type systems in Haskell or Scala.
<http://bit.ly/1BMu8QD>

Expectancy-value theory (<http://bit.ly/1sctSGD>) describes how individuals balance out the value they expect to get from their actions. Educational psychologists talk about how that expectation motivates learning (<http://edcate.co/1iefV80>). Students ask themselves, “Can I learn this?” and “Do I *want* to learn this? Is it *worth* it?” You do not pursue a degree in music if you do not believe you have musical ability. Even if you love art history, you might not get a degree in it if you do not think it will pay off in a career. Most of us do not learn Dvorak keyboards (<http://bit.ly/1jvaFNC>), even though they are provably better than Qwerty, because the *perceived* costs just are not worth the *perceived* benefit. The actual costs and benefits do not really play a role here—perception drives motivation to learn.

If you cannot imagine closures, why would you want to learn them? If our programming languages have inscrutable features (that is, high cognitive load to learn them) with indeterminate benefits, why go to the effort? That is low learnability. If students are not convinced they can learn it and they are not convinced of the value, then they do not learn it.

The social cost of going in a new direction. I was at a workshop on CS Education recently where a learning scientist talked about a study of physicists who did their programming in Fortran-like languages and only used arrays for all their data structures. Computer scientists in the room saw this as a challenge. How do we get these physicists to learn a better language with a better design, maybe object-oriented or functional? How do we get them to use better data structures? Then one of the other learning scientists asked, “How do we *know* that our way is *better*? Consider the possibility that we’re *wrong*.”

We computer scientists are always happy to argue about the value of one programming paradigm over another. But if we think about it from Andy

Ko’s usability perspective, we need to think about it for specific users and uses. How do we know that we can make life better for these Fortran-using physicists?

What if we convinced some group of these Fortran-using physicists to move to a new language with a new paradigm? Languages do not get used in a vacuum—they get used in a community. We have now cut our target physicists off from the rest of their community. They cannot share code. They cannot use others’ libraries, tools, and procedures. The costs of learning a new language (with new libraries, procedures, and tools) would likely reduce productivity enormously. Maybe productivity would be greater later. Maybe. The value is uncertain and in the future, but the cost is high and immediate.

Maybe we should focus on students entering the Fortran-using physics community, and convince them to learn the new languages. Learning scientists talk about student motivation to join a “community of practice” (<http://bit.ly/1kDIhFJ>). Our hypothetical physics student wants to join *that* community. They are learning to value what the community values. Trying to teach them a new language is saying: “Here, use this—it’s way better than what the people you admire use.” The student response is obvious, “Why should I believe *you*? How do *you* know it’s *better*, if it’s not what my community uses?”

Solution: Focus on usability. Communities change, and people learn. Even Fortran-using physicists change how they do what they do. The point is that we cannot impose change from the outside, especially when value is uncertain.

The answer to improving both usability and learnability of programming languages is in another HCI dictum: “*Know thy users, for they are not you.*” We improve the usability and learnability of our programming languages by working with our users, figuring out what they want to do, and help them to do it. Then the value is clear, and the communities will adopt what they see as valuable.

Valerie Barr is a professor at Union College, and chair of ACM-W, ACM’s Committee on Women in Computing. Mark Guzdial is a professor at the Georgia Institute of Technology

© 2015 ACM 0001-0782/15/03 \$15.00