DOI:10.1145/2380656.2380660 http://cacm.acm.org/blogs/blog-cacm

Levels of Abstraction; Pre-Teens and Career Choices

Mark Guzdial writes about the need for programming languages to support multimedia at all levels. Judy Robertson shares insights about 12-year-old students' lack of understanding about computer science.

Mark Guzdial
"Why Don't Languages Support Multimedia All the Way Down?"

http://cacm.acm.org/blogs/blog-cacm/109917
June 22, 2011

Donald Knuth gave the keynote talk at ITICSE 2003 on "Bottom-Up Education." He argued that the hallmark of thinking like a computer scientist was being able to shift levels of abstraction, from the highest levels of application, all the way down to the bits, if necessary. He was arguing for his MMIX processor, but the same argument can be made in lots of different pedagogical contexts.

That's really what Barbara Ericson and I are doing in our Media Computation approach to introductory computing. Students today use digital media every day. They recognize the manipulation of media as a relevant and useful activity. In our approach, we teach students to manipulate digital sounds at the sample level and digital pictures at the pixel level. They can then write simple loops to create Photoshop-like effects, like flipping an image or removing red eye, or to create digital sound effects, like creating an echo, splicing, or reversing sounds. Manipulating pixels and samples is fun and easy—we've shown that it's a CS1-level activity. It's another case of manipulating the lowest-levels of abstraction to create an effect at the application level.

The problem is finding languages and libraries that support this level of access and manipulation. Sure, lots of languages can show pictures and play sounds, but that's getting stuck at Knuth's highest level of abstraction. How many languages and libraries, even those aimed at students, let you shift levels of abstraction with media?

Barbara and I wrote our books in Python and Java by cheating. Java does support shifting levels of abstraction. We chose a version of Python, Jython, that lets us reuse the classes we wrote in Java. I have also been able to construct our Media Computation examples in Squeak as well. Jennifer Burg has shown how easily it can be done in C. Then that's really about it.

Our publisher has encouraged us to look into using Media Computation with other languages, especially Python 3.0. And that's where we run into problems. We can manipulate pixels; in fact, Nick Parlante at Stanford has started teaching JavaScript using Media Computation with pixel-level manipulations. A recent review of audio packages for Python shows that *none* of them support sample-level manipulations cross-platform. I have been able to write small examples in PyGame, but there are some significant bugs in that package. For example, if you open up a sound that is not CD quality, PyGame "re-samples" the sound, so a sound that you open and save back out might double in size. If you care about the byte level, it is disconcerting for more of them to appear without warning.

I have found no packages that let me do pixel- and sample-level manipulations in other languages. There is a book on learning Haskell with multimedia, but it is all at the highest level of abstraction. I have tried to find such supports for Scheme, but the only audio package I have found allows you to play sounds, but you still can't access the samples in those sounds. It's frustrating because, if a language or library supports *playing* the sounds, then those samples are *somewhere* there in memory. Let us at them!

Now, I'll bet there are libraries for

manipulating pixels of images and samples of sounds in many of these languages, but my experience suggests that they're not obvious, not easy to find. Why not? Don't we think that Donald Knuth is right, and it really is important for CS students to be able to get all the way down *easily* and *obviously* to understand how to build it all back up?

There is an argument that real application developers don't typically work at that level. Video game programmers leave the pixel and sample manipulations to the gaming engine. Most application developers just want to show pictures and play sounds and videos. But that doesn't excuse not providing access for *students*. Learning is a conscious process. It's so much easier to be conscious about things we can see. How do you study something that you can't see, that you can't manipulate? How do you learn samples and pixels if they're always hidden inside some library or engine? Sure, it's possible to learn things that are invisible, but it works much better if they are visible, accessible, and malleable.

Media is something that I care about, but I wonder if it's an instance of a larger problem. It's important for students to shift levels of abstraction. How well do our languages for students support shifting levels of abstraction; that is, being able to see everything from the application level down to the bytes? And if they don't, we should be asking, "Why not?"

Judy Robertson "Foggy Futures: The Confused Computing Career Aspirations of 12-Year-Olds"

http://cacm.acm.org/blogs/blog-cacm/115085
July 25, 2011

Can you remember what it was like to be 12 years old and have an infinity of possible careers in front of you? What made you choose computing? Was it a positive choice, or did you drift into it? I have been thinking about this today because I have been listening to recordings of interviews with 12-year-old boys and girls about their attitudes to computing, and their future career choices.

I chose computing because it was difficult. I wanted the challenge. I distinctly remember trying to work out how to write a sorting algorithm as I trudged along my morning newspaper delivery route. Naturally because it was so hard it seemed the obvious thing to want to do with my life. (Go figure!) Back in those days, computers weren't part of everyday life. My exposure to computing was from learning to program at school, and from watching my dad type expert systems code from the back of a book into an Amstrad word processor.

JUDY ROBERTSON

"I chose computing because it was difficult."

But now, children's exposure to computing is ubiquitous and centered around the use of computers rather than more fundamental computer science concepts. In our recent interviews with 12-year-olds who had just completed a game-making project, we asked them about what they understood by the term "computing." It became clear that their understandings were partly related to the label for the subject on the timetable, such as "ICT," "Information Technology," or "Computing Studies." None of the classes were labeled "Computer Science."

When asked what they might expect to do in a computing class, the children typically told us about using applications: spreadsheets, databases, PowerPoint, Word, and sound recording packages. The "Internet" was often featured, in the sense of learning to use Internet-based applications safely and effectively. They thought that in a computing class they might learn how to use computers in general, and learn to use programs they had not come across before. A couple of students mentioned learning about what computers can do, and what parts are inside them. Oddly, no one mentioned that they would expect to study the fundamental properties of computation, or the patterns for effective software design.

In terms of future careers, the students often explained that while they thought computing was an important aspect of many lines of work, it was not something they wished to focus on. A boy who wanted to be a pilot mentioned that "there are a lot of computers in that. You have to login when you're going out and log out and your computer's inside the plane." A girl who wanted to be a doctor conceded that she would learn computing if it were necessary to do the job. Worryingly, a couple of the girls had misconceptions about how programming might fit into careers:

Girl A: "To be an optician or a vet, you have to use the computer quite a lot for that."

Girl B: "Programming and stuff."

Girl A: "To be an optician you have to program what it is, know what it is, certain parts. Like what's wrong, how they can help and stuff."

Interviewer: "Have you done any programming yet in school?"

Girl A: "I don't know."

Girl B: "We did. We did our own program. 'My computer of the future,' that was a programming project."

Girl A: "We know that programming is like typing and stuff."

Girl B: "Is it?"

Girl A: "So I believe...."

Typing? Opticians? This calls into question an attitude questionnaire I recently used that included a perfectly reasonable-seeming question about how much the respondent enjoyed programming. The results may not be very reliable if some of the kids think programming is merely typing.

This brings me to a broader point about computer science education. With many excellent initiatives to encourage students to study computing under way, we are going to need to evaluate their effectiveness. To do so, we need surveys that reliably and validly uncover changes in attitudes to computing. But such instruments will need to be designed very carefully if there is such a mismatch between researchers' and students' understanding of basic terms such as "computing" and "programming." Perhaps vocabulary development needs to be part of the computer science education itself. We need to clearly articulate to pre-teens what computer science is, as well as why it is so important. C

Mark Guzdial is a professor at the Georgia Institute of Technology. **Judy Robertson** is a lecturer at Heriot-Watt University.