Interview with Mark Guzdial, Georgia Institute of Technology: Computing as Creation

Interviewed by Peter Denning for Ubiquity

Interview conducted May 2010

Editor's Introduction

Mark Guzdial is a Professor in the School of Interactive Computing at Georgia Institute of Technology (Georgia Tech). His research focuses on the intersection of computing and education, from the role of computing in facilitating education to how we educate about computing. In this interview with him, he discusses how we teach computing and to whom, especially his contention that a contextualized approach is a powerful tool to teach everyone about computing.

Peter Denning
Editor

Interview with Mark Guzdial: Computing as Creation

Interviewed by Peter Denning for Ubiquity

Interview conducted May 2010

Ubiquity: You're well known for your efforts to make computing education available to everyone. How did you come to be interested in this?

Mark Guzdial: In my senior year of high school (1980), I taught a community education course, "Bits, Bytes, and Basic," for adults who wanted to learn what these newfangled devices like the Apple II and TRS-80 were all about. The course was successful, and the community education program asked me to expand the offerings. I taught all through undergraduate, and in my senior year of undergraduate, I was teaching more credit hours (including a course in 6502 assembly language at a local community college) than I was taking. I grew up in computing thinking about how to teach it. At a Bell Labs internship in Summer 1982, I discovered Adele Goldberg and Alan Kay's paper "Personal Dynamic Media." That vision of the computer as a learning and learnable tool made me want to go to graduate school, which I eventually did at the University of Michigan to earn a joint PhD in Education and Computer Science.

Ubiquity: There is a lot of talk about computing for students interested in science, math, engineering, and technology (abbreviated SMET). That's an ambitious goal all by itself. Why do you want to expand to all knowledge workers?

MG: In 1961, Alan Perlis first made the argument that everyone in academia should learn to program. He argued that computer science is the study of *process*, and that process is important to everyone. The business students who study logistics are amazed when I tell them in my class that computer scientists know how to compare and evaluate processes. Perlis went on to say that the automated execution of process changes everything, and he gave an example of how economics became an experimental science at Carnegie Tech when they could start to run simulations. Perlis' argument still stands today. All knowledge workers care about process, and all knowledge working domains are transformed by automated execution of process. One of my PhD students, Brian Dorn, is studying graphics designers (who typically took art classes in

college, and no CS) who teach themselves how to program in order to use Photoshop more efficiently. Computing is for everyone.

Ubiquity: In 2003, you introduced a new introductory course at Georgia Tech. You called it media computation. How did the students respond the first time you did it?

MG: In 2003, the course was an easy sell. We were replacing a course that more than 50% of Liberal Arts, Architecture, and Management students were failing each semester. A course designed to be relevant to them (focusing on media manipulation), using a language (Python) that they found approachable, led to a lot of excitement and much higher success rates. More important is that two years later, with other people teaching the course, when students didn't remember "the bad old days," students still found the course creative, relevant, and engaging.

Ubiquity: How did the other faculty respond to this course? I understand there was a lot of resistance to using it as an alternative to the introductory course.

MG: Georgia Tech has had a requirement for all undergraduates to take an introductory computing course, and from 1999-2003, there was only one course that met that requirement. By 2003, our faculty saw that the status quo wasn't working. The idea of creating an alternative course for Engineers (using MATLAB) was a no-brainer. The idea of creating a CS1 just for the Liberal Arts, Architecture, and Management majors was a harder sell, but Kurt Eiselt and Jim Foley supported the effort and helped make it happen.

Ubiquity: What are some of the things you teach in media computation?

MG: We teach the same basic content of any introductory computing course, but where the data being manipulated are pixels of pictures and samples of sounds. For example, every CS1 has students write programs that iterate across an array, to compute an average or a maximum. We do the same thing, but we're iterative over the pixels of a picture to create a grayscale or negative image. Every CS1 concatenates arrays. We do, too, but the arrays contain sound samples, so that the concatenation is digital splicing. Every CS1 has students do something with only a part of an array. We do, too, by removing red from only those pixels in the eye without messing with the red in the person's clothes. We can motivate problems somewhat differently. We talk about how digital video special effects are created, by showing scenes of movies and then implementing those effects.

Where we end up is slightly different. We write programs to map from sounds to images (to create sound visualizations), and then *back* again, so that we can talk about information as being something different than the medium in which that information is represented. The general notion of "information" is something we rarely get to in CS1's, but it's a natural discussion in a media computation class.

In our Media Computation data structures class, we cover linked lists, trees, stacks, and queues, but all in a media context. In every data structures class, students learn to clone a node a dozen times to make a linked list, and how to weave new nodes into the existing list. And when those nodes contain numbers or strings, this is a really boring assignment. When those nodes contain MIDI notes, it becomes an exercise in music composition. Similarly, the first tree that we introduce is a scene graph, which is a standard computer animation data structure. It's the same content, but with different data which leads to greater motivation and engagement

Ubiquity: From your description, it looks like you like to bring out the scientific principles underlying media computation. A good example is sampling. We know from communication theory that if you take samples from a continuous waveform at twice the highest frequency in the waveform, you can completely recover and regenerate the continuous signal from the sampled data. The finite data sample contains all the information about the infinite continuous wave! Sampling would be a principle that is not covered in traditional introductions to computing.

MG: In general, we're not changing the learning goals or objectives of introductory computing, as much as we're changing the *context*. We deal with arrays and control structures and objects, but use digital media as our data. Our media computation course shifts from the context of programming to individual creativity in digital media. That context calls on us to explore scientific principles appropriate to it.

Because of the context of media, we can explore scientific principles that, as you say, are not traditionally covered in introductions to computing. For example, the same principle of "sampling" explains how we shrink an image (by purposefully and regularly under-sampling an image) and scale up an image (by oversampling). Now, we extend from this principle back to "algorithm." We can under-sample the samples of a sound in order shift it higher in frequency, and we can over-sample the samples of a sound to shift it lower in frequency. We are now using the same process on different data for the same effect. We also end up teaching a bunch of psychophysics. How many bits per sample do you need to record voice? It turns out that 1 bit

is enough for intelligible speech, which leads us to ask questions of how our brains process speech such that we can understand from so little data.

Ubiquity: That's interesting. You generate deep involvement by the students by asking them in their projects to apply the principles to create new expressions. But doesn't this create artists rather than scientists?

MG: I'm not teaching art. We encourage creative exploration, and we particularly reward new expression that requires invention in the computing, for example, a new image or sound effect. Our computer engineering faculty call my course "Baby DSP" (Digital Signal Processing) because we're touching on their themes, too. My students are going to have a wide variety of careers, because I get the non-CS majors. I'm happy to hear that they have used the Media Computation course to inform careers in art, engineering, computing or science. My goal is to teach computer science. That's my focus and what I'm grading on.

Ubiquity: Let me push on that some more. You keep emphasizing "expression". Simply letting people say who they are may help motivate them. But how is this computer science?

MG: Computing is a powerful medium for self-expression. Expression is a powerful motivator. I am using that power and motivation to teach computer science.

Ubiquity: Let's talk about distance education. What is your approach to this? How do you maintain student engagement?

MG: There is an enormous need for computing education at a distance. For example, there's an effort in our community called "CS10K" whose goal is to have 10,000 high school computer science teachers to teach the new advanced placement course by 2015. Today, we only have 2,000 AP CS teachers. How are we going to ramp up 8,000 more teachers in five years? We can't do all that with face-to-face courses. We need distance education to meet these needs.

Distance education is an unsolved problem. Much of what we do in distance education is what is sometimes called a "remote classroom," i.e., to reproduce the classroom setting but with a voice-video link between the student and teacher. Studies of this mode show a failure rate about twice that of regular classrooms. The teachers say it's extremely hard to maintain student engagement and, if they aren't engaged, they drop out.

Ubiquity: In his book, *On the Internet*, Hubert Dreyfus expresses grave doubts that distance education would be able to raise someone past a level of competence in a domain. Now competence is a pretty good objective—after all, you strive for that in the normal undergraduate curriculum. But for graduate and continuing education you might want to reach higher, to proficiency or expert. Can distance education take us there?

MG: I don't know. I can see the argument for why it would not work. But Dreyfus seems to believe that good coaching and varied experiences are essential to the higher levels of competence.

I think that there is an important role for distance education for experts. Caroline Simard at Anita Borg Institute did a study of female mid-level managers in IT companies. These women have more family commitments than their male counterparts, and yet need to keep up their technical skills. Distance education on the latest, say, Web technologies or functional languages for multi-core architectures could serve this audience well, so that the women could update their skills within their limited time availability. These women already have competencies, maybe even expertise in some technologies. They simply want to update, which *can* be done at a distance.

I want to be careful here. The construction of immersive virtual worlds is beyond my area of expertise. I don't know the literature on the use of immersive virtual worlds for education. I would agree that players of MMOGs (massively multiplayer online games) are learning something, but can they transfer game learning into the real world? Does what they learn in the game match what we as a society want them to learn? I am open to good answers, but I suspect a lot of careful design work will have to happen before these environments are ready for general education.

Ubiquity: There has been a lot of talk the past couple of years about "computational thinking" and its place in general education. What does computational thinking mean to you? How would you integrate it into your curricula?

MG: It's odd, so many people like the idea of computational thinking, but there is no general agreement on a definition. I don't have a good definition for computational thinking. I do see that it's a powerful idea. I believe that our Media Computation courses touch on computational thinking ideas, in fact, more broadly than traditional programming-focused introductory courses normally do. We address issues of representation, and information, and even issues of

data scale-up (e.g., a 12 megapixel picture is a *lot* of pixels to process, and a 3-minute stereo CD-quality sound is 15 megabytes) more than most CS1's address.

Ubiquity: There has also been a lot of talk about teaching to the fundamental principles of computing, looking past current technologies, which can be faddish, and finding the unchanging aspects. Do you think students are interested in learning about fundamental principles? Or do they prefer the technologies?

MG: Students are definitely interested in fundamental principles, but we have to motivate those principles first. In our first Media Computation course, students start asking us, "How come my program is always slower than Photoshop for doing the same things?" Our answer involves talking about optimization, and interpretation versus compilation, and about machine versus higher-level languages. These are pretty deep, fundamental ideas that come about naturally in a Media Computation context, and students really do engage in that part of the course. The point of the context is to explain to the students why they should care about those principles.

