White Paper: Making Programming Meaningful to Induce Passion for Broadening Computing

Mark Guzdial and Barbara Ericson

Donald Knuth argued in his 1974 Turing Award lecture that computer programming is an art form. He went on to write a famous and oft-cited series of books on The Art of Computer Programming. He meant the word "art" in several senses, including art as something beautiful, art as the work of creative virtuosity, and art as the artifact of an artisan.

Dan Garcia echoes Knuth's sentiment in naming his course "The Beauty and Joy of Computing." Computing can be beautiful and create beauty. It can bring joy. Garcia's popular (now, six part) series of SIGCSE Symposium presentations, "Rediscovering the passion, beauty, joy and awe: making computing fun again" brings in an important quality to both Garcia and Knuth's notion of computing: It demands passion, to devote the hours necessary to develop literacy and even virtuosity with a creative form of expression like computer programming.

As our colleagues Betsy DiSalvo and Amy Bruckman (2011) wrote:

Computer science is not that difficult but wanting to learn it is.

Programming is the magic spark of computer science. Without programming, computer science is discrete mathematics. Without programming, computer "science" is the scientific study of a sophisticated machine running someone else's programs. Programming is the passion behind computer science. Efforts to introduce computing without programming are inherently limited. We might be able to get students to think about algorithms or representations (like in CS Unplugged), but if students are going to use these ideas and progress in the study of computer science, they will have to program. What's more, we want them to want to program!

The central research question that we are exploring here is: How do we induce passion for programming while simultaneously broadening participation in computing? It's pretty easy to induce passion among those students who are passionate about how mainstream computer science looks today. If you are excited about systems programming, about coding algorithms like sorting, and about manipulating data types, you are the kind who will easily become passionate about programming. We have to invent a meaning for programming that goes beyond the computer itself and is recognizable by our target audience.

We argue that:

- What induces passion for programming in the students we don't have is unlikely to be the same as what our current students (and practitioners) find meaningful.
- To teach programming solely in terms of current practice is to ignore the possibility of change in
what we program and how we program.

To induce passion for programming among the many currently NOT in computer science, we have to make programming meaningful for them. Students must see that programming is a valuable activity, that is worth the cost of struggling to learn a foreign notation and the hours of practice to develop literacy and even virtuosity. This is an area that we have been exploring in our work in "Georgia Comput3r!" through contextualized computing education curricula (like Media Computation); through our summer camps and workshops around robots, cell phone programming, art, and animation (Ericson & McKlin, 2012); and through the work of our colleagues, like Betsy DiSalvo in the Glitch Game Testers project.

The question of inducing passion is not solely a curricular question. As we learned in our 10 year study of Media Computation (Guzdial, 2013), curriculum can help retain students, but it is unlikely to recruit them. Few students will enroll in a course of study without already finding some meaning in it. Thus, we have to develop a sense of meaning for programming before the classroom.

We imagine a research program in inducing passion for computer programming to broaden participation to have several threads:

- Developing a collaborative design process for meaningful and passion-inducing activities.
- Exploring performance for recruiting.
- Changing and scaffolding programming process to emphasize meaning.

**Designing Meaningful and Passion-Inducing Activities**

What induces passion for programming in the students we don’t have is unlikely to be the same as what our current students (and practitioners) find meaningful. Programming robots, video games, and cell phone programming all make sense to us as meaningful activities -- but we're in the computer science research community. Reaching the students who are today unlikely to enter computer science will require us to discern what is likely to be meaningful for them, what is likely to induce passion for programming.

Betsy DiSalvo used a collaborative design process to draw African-American teens into computer science (DiSalvo et al., in press). Building video games was seen as too complicated by these students, but testing video games was attractive and served as a draw into a passion for computer science (DiSalvo et al., 2013).

We have considerable experience in finding and developing computing activities that students (especially under-represented minorities and females) find meaningful (Ericson & McKlin, 2012; Guzdial, 2013). What we don’t yet have is a design process that supports finding and developing computing activities. Like DiSalvo (2013) and Joseph (2000), we believe that a design process for meaning and inducing passion will involve collaborative design activities. Developing and defining that collaborative design process is a critical need.

**Exploring performance for recruitment**

Where do students get their ideas what computer science is about in the first place? Is it from mass
media, hearing from people who program, or seeing the geeky kids in the hallway?

Geoff Cox (2013) argues that most people probably don't ever see program code, programming, or programmers. They use technology, but the process of creating it is invisible to them. How could they possibly engage in the community of practice (Lave & Wenger, 1991) of programming, or even imagine themselves as part of it?

Whether potential students first see 'programming in mass media or from watching someone code, we might describe this view of programming as a kind of performance. What aspects of a programming performance engage the student in the audience? What should the performance be achieving to give meaning to the performance?

Live coding provides one model for what this kind of performance might look like. Live coders are musicians who improve music through writing and modifying program code. When they perform, their code is on display for the audience. As the live coders say in their manifesto: "Give us access to the performer's mind, to the whole human instrument." A live coder is an embodiment of a programmer for an audience that may never have seen a programmer before. The act of live coding is programming with a purpose, to generate music.

Is live coding a kind of performance that engages the student to pursue programming further? What are the variables of the live coding performance that make it more or less successful as a recruitment mechanism?

Changing and scaffolding programming process to emphasize meaning

An activity is meaningful (i.e., is worth doing) if the product or benefit of the activity is greater than the cost of the activity. A Sisyphean task is by definition not meaningful, because there is no real product or benefit and the cost is infinite. For many students, programming is perceived as a Sisyphean task. Students struggle with details like syntax and data definitions that seem unrelated to the product (Stefik, Siebert, et al., 2011).

Practitioners are loathe to introduce new programming languages and practices because it's not what is in current practice. To teach programming solely in terms of current practice is to ignore the possibility of change in what we program and how we program. Programming will change: Languages and practices. We should have the flexibility to try new languages like Quorum that are provably more usable (Stefik et al., 2011) and new practices (like live coding) that make the meaning more evident.

Conclusion: We can't avoid programming

Certainly, we can introduce computer science without programming. Certainly, computer science is more than just programming. But computer science certainly includes programming, and programming is a critical part of the passion, beauty, joy, awe, and art of computer science. We have to engage with

---

1 Perhaps just described.

2 See http://www.toplap.org as the center of the live coding community.

3 http://toplap.org/wiki/ManifestoDraft
the hard part of broadening participation in computing, by introducing programming. We have to develop the design processes so we can figure out how to introduce programming to a broader audience; we have to develop performances that recruit and engage students; and we have to be free to change programming so that it is meaningful and has a chance to induce passion.

References


