Feedback Review

Patrick Ester

To be honest. the presentation went by so quick that I do not recall much of the feedback. We had only a minute of two of feedback time. I did not feel as though I could transcribe all of their comments before responding. The only thing I remember is Professor Drogos asking me a few things. She wanted to know if I was a programmer wanting to build an educational tool, or an educator building a programming tool. She also suggested that I conduct a survey of my 6th Grade students to get an idea of the ways in which they are having difficulty in understanding the content.

I distributed the handout I showed you in your office. Only one person filled it out (Amanda Sparling). Here is what she had to say:

`Why this language?`

The reasons are many. It is a multiplatform language that runs in the browser. You do not need special hardware or pay a yearly developer fee. The language is graphical in nature, so students get immediate feedback. The Elm Reactor (time-traveling debugger) opens new possibilities for students to relate to and understand the code. The syntax, at times, can be better than other languages. Elm allows you to develop for the web without having to learn HTLM/CSS/JS.

`Do you think the lack of "real world" applicability will stop them from utilizing and using after the class.`

My goal is to help the students be creative. Instead of drawing, painting, music, or poetry, I am using teaching them how to use a computer. The class I teach is a required class, therefore there are a significant number of students who do not care for programming. They are there because they have to be. Not every student wants to be a programmer for a living. I would also argue that 6th Graders may change their mind about what they want to do for a living. This also assumes that they have given their future any significant thought. A major premise of my project is that what we consider to be "real world" programming is done in an inefficient manner. The practice of writing code needs to be rethought. I purposely am choosing a path that does not always align with the "real world."

More to the point, the students have no idea of what the "real world" of programming actually is. This is a construct that adults, typically in the profession, place on the educational initiatives to get more kids to code. The problem, I think, is that "real world" programming is hard to define. Just go to Hacker News and look up conversations on which JS frameworks or which text editor to use. There is much bickering on why the other people are wrong. Many commenters share the "right" way of doing things. One could argue that iOS development has a "real world" scenario. Develop on a Mac, in Objective C or Swift, and with XCode. However this is changing

with development models like Ruby Motion, PhoneGap, and React Native.

`My concern with this approach is it teahes how to think and be invloved with programming. It doesnot really produce a tangible skill that children can run with and tinker with on their own.`

To be blunt, I do not work at a trade school. I do not view my work as succesful or as a failure if my students do not have skills to get a job directly out of high school. The comment seems to draw a distinction between thinking about programming and tinkering. I do not view this project as doing one but not the other. I want my students to tinker and create, but I also want them to understand how what they write directly impacts the output of their program. If my work gets students to spend more time thinking and understanding then so be it. I would rather my students be able to think and understand. As mentioned above, not every studnet is destined to be a programmer. The ability to think about and understand new topics is a skill that will serve them well whatever they choose to do in the future.

The above sentiment indicates that marketable skills are the hallmark of success. I cannot disagree more. The role of education is not to fulfill the ever increasing profit motives of capitalist society. In an ideal world, education exists free from the economic, political, and theocratic pressures of society. That is not to say that education should exist in opposition to society, but that education's value comes from not pandering to popular sentiments at that moment. If this were true, education would try to prepare students for an always-moving set of goal posts. Progress would nigh be impossible.

`To build off my last thoughts, the prominence of "self-taught" coders and their "maker/hacker" mentality is so essential. I'm afraid this hinders that effort instead of tools [sic] to tinker on your own.`

This comment places primacy on the students ability to figure something out for themselves. I keep coming back to the programmer's rite of passage. To earn the title of a maker/coder/hacker, you must suffer through the confusion and frustration of programming on your own. To make programming more accessible would let in the hoi polloi and sully the reputation of the "authentic coder." The fear of the "other" is one of the reasons the technology community is so messed up. White privilege and misogony are often critiques levied at Silicon Valley, and with good reason too. By keeping the community small and insular, these problems will not go away. The tech community needs diversity, and one way to address this is to make programming more accessible to people.

I also thnk the above comment runs counter to my job as an educator. This project is not designed to allow students to program without any work at all. An educator's job is to make students work, but our job should never be to actively obfuscate knowledge under the guise of hard work. The way we program today is arbitrary. There is no law that requires text editors and recompliling or reinterpreting the code for every change. We have the computing power to create development environments that make programming more accessible. Bret Victor has shown this in detail. Apple is moving this way with Swift, and Elm is doing the same for web

development.

To be fair, the pedogocial teahcings of Piaget (and then Papert) champion an idea call constructivism. The idea is that kids are giving the building blocks to knowledge, and then they are the ones to construct the ideas and concepts they learn. At no point, however, do Piaget and Papert require learning to be an ardious task, destined for the select few. When Papert wrote the programming language Logo, he invented the idea of a turtle so all kids could relate to the activity. The syntax of the language was also simple and reflected precisely what the turtle would do.