Spencer Cornish
How Students Learn paper
CSCI495: Field Work
4-7-2018

Student Learning Reflection

Agreed aspects of learning

Students often learn in different ways, or at least that's how it looks from the outset. Many things can be shared, knowledge-wise, in a predictable way. *How people learn: Bridging Research and Practice* by Donovan, Bransford, and Pellegrino shows all sorts of different teaching strategies and observations on the student that are pretty commonly shared between each other.

One of these similarities I have observed in my own experience is how preconceptions can cause some students to revert to their previous misconceptions. In the text, one of the classic examples given is how poorly students perform when asked to hit a target with the ball at the minimum velocity. Both students from large college physics programs and elementary students alike fell back on their preconceived notions about how physics works, despite having learned Newtonian laws, in the case of the physics students.

In my own experience, I have seen this happening often with students who already had programming experience. In one case, a student had previous experience programming in Java, a language with many syntactical differences from Python. He kept trying to follow his old programming principles, such as adding semicolons and brackets, which caused him great difficulty in getting Python code to compile for obvious reasons. Another such example was with a student who had learned to program before college, and really liked to use global variables. Despite being taught to use localized variables, he still chose to use his old strategy outside of class, and on some of the in-class labs, much to the displeasure of the TAs.

Another similarity I have observed firsthand is how important deep knowledge is of a topic, rather than superficial information. Many students try to skim and avoid putting in the time to understand the deeper meaning behind what they are actually learning. I myself have been guilty of this, but I have since learned that in the end, it is much easier to just bite the bullet early and learn everything you need right away.

One of the times I recall encountering this during my time as a TA was when I was helping a student who was struggling with how to structure a function to count the number of array elements which were the same and return once a different number was hit. In this case, his understanding of how return worked was lacking, so he had it in every iteration of the loop he was using. If he would have understood that return exits the function, he probably wouldn't have put it in the places he did.

Disagreed aspects of learning

Although I agreed with most of what the paper had to say about learning, I feel as though there are some areas of student learning and teaching that do not always work the same in the real world. As is the case with many things, the real world implementation requires many changes from it's optimal, academic implementation. In the case of learning strategies, this can apply to not only students differently, but also fields of study.

One of these differences I have noticed is that, in my opinion, teachers do not necessarily have to draw out existing knowledge. This may be especially true in computer science, as software development shares few similarities with a self-taught, backyard programmer. Often, students have previous knowledge, but a great deal of it should not necessarily be trusted by either the instructor or the student. Instead, I believe an equal baseline of knowledge and experience should be built, in the hopes that the students will be even better equipped to tackle the new content which builds on this base.

In my experience in the lab, a good example of this was the student who had previously programmed. She had spent time in high school writing code, with very little instruction, and as such was essentially self-taught. Unfortunately for her, relying on this base of knowledge was at best a hindrance, at worst a total blocker for progress. This was due to the fact that in many of the assignments for the class, good technique and good basic skills were required for a successful product. She did not come equipped with those skills and instead had to start from scratch with her learning. Truthfully, this wasn't that bad at all in the end, and she ended up with a great foundation for next semester.

Another difference in opinion I noticed was related to using alternate methods of testing students. While I personally do not enjoy tests and prefer other methods of evaluation, I do still see the merit of traditional testing, both in the lab and in my own life. In the text, the authors make testing out to be an arcane practice of the old times, an artifact of old teaching methods. In reality, I do not find this to be 100% accurate.

In the lab, I have noticed that after exams were given, many students found the core concepts much easier. In a way, these exams forced the students to fully learn the ins and outs of every bit of language control structure, in a way that they normally would not otherwise. In one case, I helped a student understand classes one week, and by the next week after the exam, he was able to fluently and easily write classes and even subclasses on his own.

Personally, I find that exams "lock in" my knowledge of a topic, and provide value to the student. Overall, I find exams to be a sort of "necessary evil" for learning computer science in the modern age.

Additional Thoughts

One of the learning strategies I have found to be particularly effective in teaching which was not mentioned by the text is asking pointed questions of the student to provoke additional thought. This is one of the more effective strategies I have found for helping students think through some particularly tough logic. It also comes in handy when a student is totally stuck, and just needs a point in the right direction. Other smaller examples of this type of teaching can be seen often in any form of teaching which requires very algorithmic-style thinking, such as math, science, and of course, engineering.

Actual examples of this are present in nearly every lab session I have ever TA'ed. One recent one was a student struggling with how to "read a variable from another function". I asked him how he would do it if I was the function, he was the driver function, and I had to edit the variable on a piece of paper. We talked through all the steps we would go through, from him giving me the paper with the info I needed me doing the math, and finally, me *returning* that variable. For him, this analogy really helped him understand the control flow, and that he needed to use a return to get that variable back again.

Overall, I found the article to do a fine job of explaining how students learn, and I found it pretty adequately sums up my experience of teaching others in the lab.