

What is retained from a first programming course?

I am interested in the process of learning (and teaching) a first programming language. What goes on inside the head of a novice programmer? My work so far has been a general review, some studies of predictors of success in a first programming course, and some work on the problems students have in our labs learning Java. The thoughts in this outline are the start of a new thread in this big picture.

Big questions

What aspects of programming do students find easy or hard to learn? Why? What can we do about it?

I want to unpack “programming language” a bit and see if I can tease apart different aspects. There is a lot of literature about programming and “cognitive load” [e.g. 1-3]. Is it possible to “quantify” the difficulty of learning a programming language? Of different aspects of a language? How does this relate to student's learning experiences?

The way in to these questions that I want to consider here is to look at what students remember from their first programming course. Some fairly basic results from the psychological literature are that we tend to remember general concepts and overviews (rather than specific details), and material which has been well understood and integrated (rather than isolated or poorly understood details). If this is the case then the pattern of what is retained and what is forgotten from a programming course might provide some insight into what was relatively easy vs. hard material. (It might also give us a way of substantiating or measuring aspects of complexity theories, which so far look pretty abstract).

So the specific question I want to ask is, what do students remember from their first programming course, and does this tell us anything useful about the learning / teaching process?

Specific question

There is one obvious way to focus this question on something attainable. Students for the introductory programming course all sit the same exam. Three months later, some of them return for second year courses. This captive audience could sit exactly the same exam again at the start of second year. (Aside: This has a practical benefit. Second year teachers always berate me for how little their incoming students have learned about programming - sigh! This would give them a useful benchmark.)

For the purposes of the study, what I would have is about 100 pairs of exams, one at the end of the introductory programming course, one three months later. The scripts are quite rich and detailed, it would be (painful but) possible to do a systematic “before and after” comparison of each one. It would be possible to identify specific questions where a student's performance got worse (or better?). Given a suitable “taxonomy” of question types, it would be possible to see what kinds of questions / topics they got worse at, and plug back in to the bigger questions outlined above.

So, the specific question I can answer is, what change is there in students' performance in a first programming exam three months after they originally sat it? It will be possible to look at both student based effects (which students remember stuff) and content based effects (what stuff is remembered?). I don't have any clear expectations in mind so I'm not looking for any specific evidence, just interesting patterns.

Sampling, bias, analysis, cost, value...

Sampling is an issue of course, I'm not going to sample students who chose not to carry on with CS courses. Is that a problem? The mechanics of the study are simple and cheap, but the analysis costs will be very high. I don't know what value to expect – there might be something interesting here, or not! But at the very least it will have a practical benefit, giving us some guidelines about the actual starting level of our second year students (not “they must know that because it was in first year”).

I haven't explored the literature in much detail yet, so there is a lot of stuff to read (has this been done already? with Java?), and reliability / validity issues to consider.

[1] Cant S, Jeffery D, & Henderson-Sellers B (1995) A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology* 37(7), 351 – 362

[2] Green T & Blackwell A (1998) Cognitive dimensions of information artefacts: a tutorial. <http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf>

[3] McCabe T & Butler C (1989) Design Complexity Measurement and Testing. *Communications of the ACM* 32(12), 1415 – 1425