Subcommittee on Computing Thoughts

Jonathan Lam, EE '22

2022/02/21

These are my personal thoughts on what computing/programming skills should be emphasized for for engineering students, particularly electrical engineering students. I do not have any experience with curriculum planning and thus this will be focused on the contents of the courses offered more than the particular courses and their credit counts. I personally believe that a strong understanding of writing programs is a great general exercise in logical and analytical thinking that will benefit anyone in a quantitative field.

In general, emphasis on low-level computing skills (assembly languages, C, CUDA, etc.) should be emphasized for electrical engineers, while emphasis for other majors should be on scientific computing (e.g., notebook-style coding in Python, MATLAB, Mathematica), modeling, and CAD software.

The skills that I believe any engineering student should be able to achieve through general computing courses at Cooper include the following. The ones that I believe are the ones that could use the most work are bolded.

- Basic computer literacy on a common Linux/Mac/Windows system.
- Good software engineering practices such as **implicit and explicit documentation, code reviews, code reuse and modularity, common style conventions, etc.**
- Proper tooling as dictated by the industry/field. For example, in software engineering, this includes **version control, IDE's, debuggers and static analyzers, build tools, etc.**
- Best practices for scientific computing, such as vectorized computing, separate execution, local package management, and the use of proper tools and visualizations.
- **Exposure to several major programming languages and paradigms**, and the understanding that certain technologies are better suited to certain tasks than others. (I am currently a major functional programming advocate and believe that simply exposing its ideas to the imperative programmer will improve their ability to build complicated abstractions from simple ones and build more robust, predictable programs.)
- Understand how to model real-world and engineering problems appropriately and efficiently to be solved or simulated using a computer program.

- Understand how to design data and algorithmic abstractions from simpler ones to solve increasingly complex computational problems.
- Practical understanding of algorithmic efficiency and resource usage.
- **The ability to treat programming as an iterative design process. E.g., understanding error messages, narrowing a bug down to a single function/line of code or iterating from a minimal working example, using existing documentation effectively, trial-and-error debugging, understanding edge cases and being able to build test cases with effective coverage, being able to use a debugger or assertions to ensure expected behavior at any particular program point (and understanding what the expected behavior is), etc. In other words, the ability to intuit about programs, and thus be able to self-learn and debug effectively.**

I feel the last point is important because it is the general analytical thinking that is emphasized in engineering but isn't likewise emphasized when programming. Many engineers treat programming as a dumb tool, when it's a much richer and interactive environment that acts as a valuable teacher as well as a tool. As a result, I see many engineers tend to parrot code, disregard code quality, and treat programs as a one-off tool, none of which help in the long term. There is some interesting research on the difference between software engineers and scientists/engineers which may be relevant.

My thoughts on existing general foundational computing courses at Cooper:

A new CS101 course I believe this would be a great idea. The current CS102 class is highly varied between sections in many ways, such as teaching style, workload, setup, topics covered, etc., at least from what I've heard during tutoring. I like the idea of a survey class, and I think it would be a cool idea to have it be a "rotational" program with multiple teachers, all of whom share their area of expertise in a beginner's project. E.g., say there are four sections (A, B, C, D) of the course. Let's say Prof. A specializes in machine learning, Prof. B in data visualization, Prof. C in web development, and Prof. D in computer graphics. It would be cool for each professor to have 3-4 weeks to teach each section a little bit about their field and go through a small project. This way, the sections will be standardized (they will all share the same teachers), (ideally) the teachers should be able to bring their enthusiasm in their field of work, and students will be exposed to a number of different subjects.

A new advanced C programming course (I believe this came up during our discussions but not sure) I think C is covered pretty extensively at Cooper. E.g., ECE160, ECE251, ECE264, ECE365 (C++ is basically a superset of C), ECE357, ECE453, ECE455, ECE466, etc. Of course, these are mostly EE courses, but I don't believe the other majors have much need for C programming.

CS102 and ECE160 I think these classes are good foundational classes, but a clear focus would benefit both of these classes (and it would be best if they overlapped less). E.g., if CS102 introduces the student to many common programming API's and good programming practices in Python, that would be be sufficient. If ECE160 purely covers the low-level programming API's and specifics of C (including and especially memory operations), that should also be sufficient. Covering too many topics (e.g., introducing C, Git, Linux, Docker, shell scripts, etc. all in the first week, and Python and data structures later in the semester, as some sections are known to do) may harm the students' learning. The bullet point about the iterative design process should be stressed at every point through these foundational classes, so that students should feel comfortable interacting with programs at this level.

DSA I and II I think it would be great to have these be promoted to three-credit courses each. However, I think a more useful change is to make these courses more interactive. Historically, these classes have been very textbook-oriented, but data structures and algorithms are a skill more than they are knowledge. I feel that in-class exercises or demonstrations will be beneficial.

EID102 This was recently removed as a requirement from the EE curriculum, and I agree with this change: at least for the computer engineering track, it does not provide much for me. I can't say much about the signals track (who may utilize more simulation software (e.g., FPGA simulators) and CAD (e.g., PCB design)), for which this class is useful.

ECE210 This course is currently required for EE's as a corequisite for ECE211 (Signals and Systems), but open to all majors. As a result, about a third of the semester is MATLAB basics, half of the semester is geared towards the EE signals curriculum (MATLAB is used in ECE302, ECE303, and ECE310), and there are three review sessions prior to the three ECE211 quizzes. The course is zero credits but meets one hour a week, with a homework requirement (I personally design the homeworks to take roughly one hour a week outside of class.) It is taught by two to three EE juniors and seniors and overseen by Prof. Fontaine. (An ulterior motive of the class by Prof. Fontaine may be to give the instructors some teaching experience.) The combination of these facts makes this class somewhat unique at Cooper (or elsewhere). There did seem to be interest in our last meeting in adapting this class to other majors, as an optional (enrichment) zero-credit class. Prof. Fontaine is probably the best person to talk about with regards to logistics on adapting or extending this class.