Computer Science Education: Lessons from Industry and Academia
Darren Kessner / Marlborough School

This article appears as a chapter in **Innovative Teaching + Learning: Disrupting the K-12 Classroom** (Making and Learning Institute, Marymount School NYC, May 2018).

Computer Science Education: Lessons from Industry and Academia

Abstract: *In this chapter, I discuss computer science education from my perspective as a software developer, having written software in both industry and academia. In particular, I discuss the primary goals of computer science education from this perspective, and I describe the current landscape of software development in industry and academia. I also summarize lessons from industry and academia that can (and I feel should) be brought to the classroom. Next, I discuss the computer science curriculum I have been developing over the past few years, at Marlborough School, an all girls independent school (grades 7-12) in Los Angeles. Finally, I present my recommendations to schools and school boards for developing a strong computer science program.*

The goal of computer science education

The computer is the single most important tool created by humans, the Swiss army knife of the 21st century. This tool is now used in nearly every human activity, by artists, scientists, doctors, entrepreneurs, lawyers, financial analysts, and more.

Over the last 30 years we have seen a technological revolution, beginning with the mass production of personal computers, followed by the creation of the internet, and more recently, the handheld internet-connected computing devices that have rapidly become ubiquitous. On the software side, each of these programmable technologies has spurred the development of new operating systems, programming languages, and software engineering techniques.

This rapid rate of technological progress creates a unique set of challenges for the computer science educator. First, many of the programming languages in use today did not exist a generation ago when today's teachers were in school. In addition, programming languages continue to develop and evolve over time, often on a time scale of a few months to a year for the release of a new version, so educators are aiming at a moving target. Third, the expansion of technology has made it challenging for the computer science educator to keep up with the latest developments. This expansion has led to specialization, which creates another set of challenges. For example, the languages and technologies underlying web development are very different from those used in mobile development. Finally, the languages and technologies that students learn how to use today will likely not exist, or at the very least look much different from what will be in use 10 years from now.

In light of these challenges, how should the computer science educator proceed? Why bother teaching students about something that will be obsolete before they finish college? What is the goal of computer science education?

Clearly, computer science is more than just programming languages and technology. In fact, academic computer scientists draw a strong distinction between *computer science* (theory) and *software engineering* or *software development* (practice). At the middle and high school level, this distinction is not useful or important. However, what *is* important is the development of the thought processes and skills needed to create software. Software allows one to create something from nothing.

The software landscape

There is a bewildering array of programming languages for the educator to choose from. Block-based languages (Scratch, Snap, Lego Mindstorms) are a great first introduction to programming. These simple languages can only take the student so far, though, so what's next?

Software development today can be broadly split into two categories: application development and web development. Application development includes desktop applications, mobile apps, scientific or data analysis applications, graphics applications and video games, and even operating systems. These applications generally use a an object-oriented language in the C/C++ family (which includes C, C++, Objective-C, C#, and Java). For example the desktop applications Microsoft Word and Adobe Photoshop are written primarily in C++. Apps on iOS have used Objective-C and its successor Swift (developed by Apple), while apps on Android use Java. Scientific applications generally use C++ or Java, though Python (not in the C/C++ family, but certainly influenced by it) is very popular also. Video games are generally written in C++ or C# (developed by Microsoft). Operating systems themselves (Windows, OSX, Linux) are written in C++ or object-oriented C. In addition, micro-controllers such as Arduino are programmed in C. So clearly a strong case can be made for introducing students to an object-oriented C/C++ family language such as Java, which is the language used in the AP Computer Science A exam.

On the other hand, much of the buzz in the software world these days is in web development, which is based on a completely separate set of languages and skills: HTML, CSS, and Javascript (very different from Java, in spite of the name). Web development has some advantages that the educator should definitely consider: 1) Many tutorials and lessons are available online, which means that no extra software need be installed on student computers. 2) Student projects can easily be shared.

And there are relatively recent developments that muddy the waters. First, Javascript is increasingly being used for application development (e.g. the Atom and Brackets text editors). Second, the WebAssembly project is bringing all other languages, including those in the C/C++ family, to the web.

The Python programming language doesn't neatly fit into the two broad categories above, but should definitely be considered by the educator. It is one of the easiest languages to read and learn to write. It is extremely useful for real scientific and data analysis applications. Python is also used in web development and can be used to program the Raspberry Pi mini computer. It is open source, and there are lots of tutorials available online.

In addition to specific languages, there are several skills used by professionals in the software industry that are useful in the computer science classroom: 1) Modularization: software applications quickly grow to be too big to hold in a single programmer's mind at any one time, so it is important to learn to break problems into components or modules. This begins with procedural and object-oriented programming, and continues with larger software components. 2) Pair programming: this is a technique in software development where two programmers work at a single computer; this can facilitate consideration of multiple alternatives when solving a problem, as well as increase correctness of the code. 3) Test-driven development: a common practice in software development where unit tests are written to test every component of the software (and the tests are generally written before the component is actually implemented); this greatly increases confidence in the correctness of the software. This is especially important (though not as common as it should be) in academic software development, where programmers don't have the benefit of a quality assurance (QA) team. 4) The ability to learn new concepts or skills on demand: Software languages are constantly changing, so the successful software developer learns to adapt quickly and learn new skills as needed.

Finally, the educator should consider the choice of how "open" languages and technologies are. Educators should be cautious about using a language or technology whose development is primarily controlled by a single corporation (e.g. Swift from Apple, C# from Microsoft, Go from Google), for two main reasons. First, a corporation can easily drop support for a language or technology, leaving the software developer in a bind. Second, corporations have a financial interest in keeping software developers locked into their particular ecosystem. Promoting a proprietary language or technology is a powerful way to do this, as it makes it difficult for software developers to switch to a different platform. Standardized and/or open source languages (e.g. C++, Java, Python, Processing, HTML/CSS/Javascript) will generally have a longer "shelf life" and skills learned in these languages will generally be more translatable to new situations.

Learning computer science

There are many different aspects to the study of computer science. First, one learns a computer language in much the same way that one learns a natural language. Students can begin by learning some common phrases and seeing how to communicate with their computer. In order to progress, the student must then study the grammar of the language a little more closely. As the student acquires command of the language's syntactical structures, the student also develops the ability to express new ideas in the language, much as a language student progresses from learning grammar to expressing ideas in an essay. As the student studies further, she learns language conventions, idioms, and alternative ways to express ideas.

On the other hand, some aspects of computer science are very mathematical in nature. For example, at some point the computer science student needs to learn how to encode numbers in binary and hexadecimal, and it is helpful to have some knowledge of discrete mathematics. The student will also need to understand logic and truth tables, so that they can understand how logical operators behave. If the student is learning computer graphics, a solid foundation in coordinate geometry is necessary, and to go further, knowledge of trigonometry, parametric equations, and linear algebra is very helpful. In addition, thinking about algorithms (computational thinking) is a very mathematical skill, though not one found in standard math classes.

The process of writing software is also similar to the iterative process involved in any engineering / design project. The software developer must first understand the intended audience. She then brainstorms to generate some ideas, and then picks one or more ideas to prototype. After evaluating the prototype(s), she decides what changes to make, and creates the next version. During this process, the software developer faces the same issues as any engineer, including time and resource constraints. Because of these constraints, there may be times when the developer must create workarounds that aren't the most elegant, but get the job done.

Finally, learning to write software is much like learning to work in any creative medium (e.g. visual arts, performing arts, athletic activities). When preparing for a future performance or competition, much of the performer/athlete's time is spent doing drills and learning basic skills, to practice for the main event. As she learns how to use the tools of her art, the artist develops her ability to create. In addition, much as an athlete or performer, the student programmer will make the most progress by working on her skills regularly (preferably daily).

There are some programming concepts that are common to most computer languages, and the beginning computer science student will generally follow the following progression in understanding of these key concepts:

- data types and variables
- control structures (conditionals and loops)
- encapsulation (functions / procedures)
- object oriented programming (OOP)
- advanced constructs (functional programming, components / application programming interfaces (APIs))

Computer science curriculum

The biggest decision for the computer science educator is which languages or technologies will serve as the foundation for the curriculum. As I discussed above ("The software landscape"), one could focus on application development or web development, or even create an entire curriculum based on Python.

At Marlborough School, some practical considerations guided our curriculum choices:

- The availability of Processing, a simplified version of Java focusing on computer graphics programming, open source and free from MIT Media Lab.
- The AP Computer Science A exam uses the Java language.
- FIRST Technical Challenge (FTC) robots are controlled by Android phones, which are programmed in Java.

Because of these considerations, our curriculum focuses on application development: we use Processing in our introductory computer science courses, and Java in our Robotics, AP Computer Science and more advanced courses.

Processing offers a very gentle introduction into text-based programming, while allowing students to progress to more difficult topics, such as classes and objects, arrays and lists, or even recursion. Processing also offers an easy entry into the world of interactive computer graphics: drawing shapes in a coordinate system, handling user interaction, specifying colors by RGB channels, image processing, and access to a simplified OpenGL (a graphics industry standard) interface. Processing is also open source, and you can use the Processing libraries from Java programs, making it a good tool for AP Computer Science A as well.

At the same time, our robotics teams participate in the FIRST Technical Challenge, and their robots are controlled by Android phones, which are programmed in Java, so we are seeing a lot of synergy between our robotics and computer science programs.

At its core our computer science curriculum is very traditional, emphasizing object-oriented programming in the C/C++ family of languages. On other hand, the Processing language was developed only recently, and the Android platform is a new place to apply Java programming. Arduino is another new application of an old language (C), and easy to move into from Processing/Java.

In addition, we introduce some web development (HTML/CSS/Javascript) in the introductory course, and some students end up working on projects in this area. In addition, we use Python in several math courses, as well as occasionally in the computer science courses. Python is good for data analysis projects, and also for playing with the Raspberry Pi mini-computer.

Of course, one could envision a very similar curriculum with similar topics, but based completely on Javascript, or based completely on Python. However, I do see an additional benefit to giving students experience with the C/C++ family. I feel that students gain a much better sense of the history of computer science, as well as the future of computer science, knowing that the code they are writing for computer graphics applications, or for programming a microcontroller, or for moving a robot, looks just like the code underlying the operating systems running on their computer or device.

While the languages and technologies will look different 10 years from now, anyone writing software will still need to think about code readability and maintainability, modularization, and testing. And they will still need to think about their software from the point of view of their users and their collaborators. So while it can feel daunting for the educator to choose which languages or technologies to focus on, I feel it's important to emphasize skills and concepts that will be useful to them no matter what language or technology they are working with. Above everything else, I feel the most important thing is to open up doors for students, and to show them that they can create something from nothing.

Recommendations

From my experience of teaching coding in the classroom, I have found that when exposed to computing in elementary school, students quickly outgrow block-based programming and are ready for text-based programming in middle school. And growing up as digital natives, these students already know how to navigate their computers and devices better than most adults. As educators, we should seize this opportunity to offer 5-6 years of instruction in computer science at the middle and high school level. However, this will take significant investment from teachers, schools, and school boards.

For some perspective on the amount of investment that will be required, consider a typical mathematics teacher. In general, a typical math teacher will have studied math every year through high school, plus 4 years (or more) in college, in addition to any credential program that they complete. In other words, a typical math teacher will have been doing math for nearly 20 years before even stepping foot in a classroom.

Now compare this to the situation in computer science. Schools have two options for finding a computer science teacher with equivalent expertise: 1) Hire a teacher with several years of professional software development experience, or 2) Invest in computer science training for current teachers. There has been a nationwide effort to increase funding for the second option, to train new computer science teachers. This effort has led to the creation of the AP Computer Science Principles exam, and many high schools around the country offered a new AP Computer Science Principles course in the 2016-17 school year. This is an introductory course aimed at making computer science more accessible to a diverse range of students. There has been an ongoing effort to train teachers to teach AP CS Principles, and the primary barrier has been that new teachers are generally fearful that they will be walking into a classroom where students may know more about the subject than they do. Understandably, much of the focus of teacher training has been to reassure new teachers that it is okay not to be a subject expert, and to lead the class as an active learner.

Clearly, the current situation in computer science education is not ideal. We would not ask a math teacher to walk into a classroom with students who know the subject better than they do, and we would not expect that the students will learn as much as if they had a teacher who is an expert in the subject. Furthermore, all of these efforts have been directed toward training teachers to teach the *first introductory course* in computer science. What do we offer our students after this first year?

If we are serious about offering quality computer science education to our middle and high school students, teachers, schools, and school boards must all invest more. Computer science teachers need to invest a significant amount of time and effort just to keep up with new developments in programming languages, software, and hardware. School administrators need to recognize this need, and provide computer science teachers with enough time and resources to do this. And school boards need to allocate enough funds to provide continuing education and training for computer science teachers. Students around the country are ready and willing to learn computer science, and there is a great need for more computer science classes at the middle and high school level. As educators we would be remiss not to fulfill this need.

Links

- Processing open source language and development environment: https://processing.org/
- Python open source language: https://www.python.org/
- Arduino open source hardware and software platform: https://www.arduino.cc/
- Raspberry Pi Foundation: https://www.raspberrypi.org/

Bio

Dr. Darren Kessner is a Math and Computer Science instructor and STEM+ Program Co-Head at Marlborough School, an independent all-girls school in Los Angeles. In addition to overseeing the Computer Science curriculum at Marlborough, Darren works with faculty in the Math, Science, and Visual Arts departments to include computational thinking and coding in existing classes. Darren also supervises students in their research and interdisciplinary projects involving computer science and engineering.

In addition to having taught at the high school and college levels, Darren has 20 years of experience as a software developer in various fields, including computer security, computer graphics, and scientific applications. He has published several scientific papers and open source software projects, and regularly runs computer workshops for teachers and research scientists.

Marlborough School

Marlborough School, founded in 1889, is an independent, urban day school serving a diverse group of young women. The school is committed to delivering a superior college preparatory education in an environment imbued with high ethical values. Marlborough is dedicated to the philosophy that academic excellence, leadership skills, and confidence flourish best in an environment exclusively devoted to the education of young women. The Marlborough community enables each student to develop her fullest potential so that she may become an actively engaged global citizen.