Computer Science Curricula 2013

Curriculum Guidelines for Undergraduate Degree Programs in Computer Science

December 20, 2013

The Joint Task Force on Computing Curricula
Association for Computing Machinery (ACM)
IEEE Computer Society

A Cooperative Project of

ACM
IEEE
IEEE Computer Society
Parallel Processing

Traditionally, introductory courses have assumed the availability of a single processor, a single process, and a single thread, with the execution of the program being completely driven by the programmer’s instructions and expectation of sequential execution. Recent hardware and software developments have prompted educators to rethink these assumptions, even at the introductory level — multicore processors are now ubiquitous, user interfaces lend themselves to asynchronous event-driven processing, and “big data” requires parallel processing and distributed storage. As a result, some introductory courses stress parallel processing from the outset (with traditional single threaded execution models being considered a special case of the more general parallel paradigm). While we believe this is an interesting model to consider in the long-term, we anticipate that introductory courses will still be dominated by the “single thread of execution” model (perhaps with the inclusion of GUI-based or robotic event-driven programming) for the foreseeable future. As more successful pedagogical approaches are developed to make parallel processing accessible to novice programmers, and paradigms for parallel programming become more commonplace, we expect to see more elements of parallel programming appearing in introductory courses.

*Tradeoffs:* Understanding parallel processing is becoming increasingly important for computer science majors and learning such models early on can give students more practice in this arena. On the other hand, parallel programming remains more difficult in most contemporary programming environments.

Platform

While many introductory programming courses make use of traditional computing platforms (e.g., desktop/laptop computers) and are, as a result, somewhat “hardware agnostic,” the past few years have seen a growing diversity in the set of programmable devices that are employed in such courses. For example, some introductory courses may choose to engage in web development or mobile device (e.g., smartphone, tablet) programming. Others have examined the use of specialty platforms, such as robots or game consoles, which may help generate more
enthusiasm for the subject among novices as well as emphasizing interaction with the external world as an essential and natural focus. Recent developments have led to physically-small, feature-restricted computational devices constructed specifically for the purpose of facilitating learning programming (e.g., raspberry-pi). In any of these cases, the use of a particular platform brings with it attendant choices for programming paradigms, component libraries, APIs, and security. Working within the software/hardware constraints of a given platform is a useful software-engineering skill, but also comes at the cost that the topics covered in the course may likewise be limited by the choice of platform.

**Tradeoffs:** The use of specific platforms can bring compelling real-world contexts into the classroom and platforms designed for pedagogy can have beneficial focus. However, it requires considerable care to ensure that platform-specific details do not swamp pedagogic objectives. Moreover, the specificity of the platform may impact the transferability of course content to downstream courses.

**Mapping to the Body of Knowledge**

Practically speaking, an introductory course sequence should not be construed as simply containing only the topics from the Software Development Fundamentals (SDF) Knowledge Area. Rather we encourage implementers of the CS2013 guidelines to think about the design space dimensions outlined above to draw on materials from multiple KAs for inclusion in an introductory course sequence. For example, even a fairly straightforward introductory course sequence will likely augment material from SDF with topics from the Programming Languages Knowledge Area related to the choice of language used in the course and potentially some concepts from Software Engineering. More broadly, a course using non-traditional platforms will draw from topics in Platform-Based Development and those emphasizing multi-processing will naturally include material from Parallel and Distributed Computing. We encourage readers to think of the CS2013 Body of Knowledge as an invitation for the construction of creative new introductory course sequences that best fit the needs of students at one’s local institution.