DOI:10.1145/1506409.1506420

Mark Guzdial

Education
Teaching Computing to Everyone

Studying the lessons learned from creating high-demand computer science courses for non-computing majors.

Several computing programs in the U.S. are developing new kinds of introductory computing courses for non-computing majors, some with support from the NSF CPATH program. At Georgia Institute of Technology (Georgia Tech), we are entering our 10th year of teaching computing to every undergraduate on campus. Our experience gained during the last decade may be useful to others working to understand how to satisfy the growing interest in computing education across the academy.

Computing in General Education

In fall 1999, the faculty at Georgia Tech adopted a requirement that all students must take a course in computing. We modified the academic year from quarters to semesters, which gave the campus the opportunity to rethink the curriculum and our general education requirements. Russ Shackelford, Rich Leblanc, Kurt Eiselt, and the College of Computing's then-dean, Peter Freeman, convinced the rest of Georgia Tech that all students who graduated from an Institute of Technology should know computing. We started before publication of the National Research Council report *Being Fluent with Information Technology*,[3] though that report significantly influenced implementation.

The new requirement wasn't a hard sell. Faculty in the College of Engineering had wanted to implement a programming requirement for their students, but couldn't decide who should teach it. The creation of the College of Computing in 1990 answered the question of whose job it was to teach computer science at Georgia Tech. Faculty in the Ivan Allen College of Liberal Arts (and in other colleges) embraced the new requirement. Computing was increasingly relevant for their disciplines, and was a value-added requirement for their graduates. The campus administration was kept abreast and involved throughout to maintain support. The new general education requirement was defined as an outcome—students would be able "to make algorithmic and data structures choices" when writing programs. That simple phrase describes a serious introductory course.

Teaching Everyone in One Class

For the first four years of the requirement, only a single class met the requirement: CS1321. There were several reasons for having only a single course. While we were already teaching approximately two-thirds of the students at Georgia Tech (because several of the largest degree programs already required computing), teaching everyone on campus meant well over 1,200 students a semester. The immensity of the task was daunting—splitting our resources over several courses seemed a bad start-up strategy. We were also explicitly concerned about creating a "service ghetto." Courses just offered as a "service" get less attention. By putting all students in one class, it is in everyone's interest to ensure the class is good.

The class received significant faculty interest and used innovative curricula. We started out using Shackelford's pseudocode approach to learning.[6] Faculty in the other majors complained about students not gaining experience debugging programs. We later moved to Felleisen et al.'s *How to Design Programs* text using Scheme.[4] These were, and are, approaches for teaching computing that have been successfully used at many institutions.

By 2002, however, CS1321 may have been the most hated course on campus. From 1999 to 2002, the overall success rate (leaving the course with an A, B, or C—not counting those students who received a D, a failing grade, or withdrew from the course) was 78%. That's not too bad for an introductory computing course.[1] However, this was a course with everyone in it. When we examine those majors where a computing requirement is atypical, we see 46.7% of architecture students succeeding each semester, 48.5% in management, and 47.9% in public policy. We failed more than half of the students in those majors each semester; females failed at nearly twice the rate of males. Statistics like these are a concern for both the Georgia Tech and the College of Computing—it hinders our relations with the rest of campus when computing is the gatekeeper holding back their students.

Developing Contextualized Computing Education

Around this time, several studies were published critiquing computing courses, including the AAUW's *Tech-Savvy* report[2] and *Unlocking the Clubhouse* by Margolis and Fisher.[5] These reports describe students' experiences in computing as "tedious," "asocial," and surprisingly, "irrelevant." A 2002 task force, chaired by Jim Foley, found similar issues at Georgia Tech. How could computing be "irrelevant" when it pervades so much of our world? Perhaps the problem was that our course had little connection to the computing in our students' world. While students are amazed at the Web, handheld video games, and smartphones, most introductory courses introduce students to the computing concepts behind these wonders with Fibonacci numbers and the Tower of Hanoi. What students saw as computing was disconnected from what we showed them in our computing class.

We adopted an approach that we call contextualized computing education. We chose to teach computing in terms of practical domains (a "context") that students recognize as important. The context permeates the course, from examples in lecture, to homework assignments, and even to the textbooks specially written for the courses. We decided to teach multiple courses, to match majors to relevant contexts.

In spring 2003, the College of Computing began offering three different introductory computing courses. The first was a continuation of CS1321, aimed at computing and sciences majors. The second was a new course for students in the College of Engineering, with much the same content, but in MATLAB and using an engineering context. The third was a new course for students in the colleges of liberal arts, architecture, and management using a context of manipulating digital media.

By putting all students in one class, it is in everyone's interest to ensure the class is good.

We chose to teach computing in terms of practical domains (a "context") that students recognize as important.

The engineering course was developed jointly with faculty from the schools of aerospace, civil, mechanical, and chemical engineering. Several faculty members in these schools had already started developing an alternative to CS1321, using MATLAB, a common programming language in engineering. Their model involved small classes in a closed lab working on real engineering problems. That course was prohibitively expensive to ramp up to over 1,000 engineering students each semester. The engineering faculty worked with David Smith of the College of Computing to create a course that used their examples and MATLAB, but taught the same computing concepts as CS1321.

The course around "media computation" was built with an advisory board of faculty from the colleges of liberal arts, architecture, and management. The board's awareness and support for the course was important in getting the course approved as fulfilling the computing requirement in programs of those colleges. The advisory board favored a programming language that was perceived as being easy to learn but was not associated with "serious" computer science. We chose the Python implementation, over concerns about both Scheme and Java.

Media computation is the context of how digital media tools like Photoshop and GIMP work. We created cross-platform libraries to manipulate pixels in a picture and samples in a sound. We taught, for example, iterating across an array by generating grayscale and negative versions of an image and array concatenation by splicing sounds. We were able to cover all the introductory computing concepts using media examples. In their homework, students created pictures, sounds, HTML pages, and animations. We created an integrated development environment that provided the media functions as well as tools for inspecting pictures and sounds.

Impact of Contextualized Computing Education

Faculty and students are happier with the new courses. The success rates rose above 80% in both the engineering and media courses. When comparing success rates to those same majors mentioned previously, we found the average success rate in the first two years for architecture students rose to 85.7%, management to 87.8%, and public policy to 85.4% per semester. The media computation course has been majority female, and women succeed at the same or better rates than the male students. Similar improvements in success rates in media computation courses have been seen among underrepresented groups at other campuses.[7]

New opportunities appear on campus when all students succeed at computing. We have introduced a minor in computer science. We had enough students interested in computing after the media course that we now offer a second course, on data structures within a media context. A second course was also developed for engineering students, so we now teach three second computing courses, as well as three introductory courses.

Faculty in the School of Interactive Computing and the School of Literature, Culture, and Communication (in the College of Liberal Arts) now offer a new joint undergraduate degree, a bachelor of science degree in computational media. The course was developed because of growing common interest in areas like video games, augmented reality, and computer animations. While the common research interests were clearly the motivating factor in deciding to create the new degree program, having a media computation course that could draw students into the new program from liberal arts, as well as from computing, facilitated the joint effort.

We see an increasing number of courses around campus that require students to write programs, though not necessarily as an outcome of the computing requirement. Computing is growing in importance in all fields. Non-computing faculty request us to include particular concepts or tools in the introductory courses and to provide prerequisite knowledge and skills for advanced courses. In this way, the computing requirement has become part of curricula across campus.

In the first years, the success rates for the new courses were sometimes higher than the success rate in the continuing CS1321. We realized that even computer science majors need introductory courses that connect explicitly to a context that students recognize as computing. In a joint effort with Bryn Mawr College and with funding by Microsoft Research, we launched the Institute for Personal Robotics in Education (IPRE, http://www.roboteducation.org) to develop a new introductory course that uses robotics as the context for teaching introductory computing.

Lessons Learned

We in the College of Computing believe the use of contextualized computing education has been a significant step in making Georgia Tech's universal computing requirement successful. Developing contextualized courses is challenging and expensive (for example, writing textbooks, developing new integrated development environments), but the results can be shared. Other campuses are adopting our contextualized approaches, and some are developing their own.

We recommend involving faculty from the other departments in building courses for non-major students. They understand their students' needs in later courses and in their students' future professions. Further, we need them as context informants as we develop courses that teach through examples from their domains.

Developing contextualized courses is challenging and expensive, but the results can be shared.

Building successful, high-demand courses for non-computing majors gives us a different perspective on the current enrollment crisis.

Finally, building successful, high-demand courses for non-computing majors gives us a different perspective on the current enrollment crisis. Students want these courses. Other schools on campus want to collaborate with us to build even more contextualized classes. Our challenge is not just in finding more majors, but finding enough faculty time to develop and teach these courses that bring real computing to a wide range of students.

References

1. Bennedsen, J. and Caspersen, M.E. Failure rates in introductory programming. *ACM SIGCSE Bulletin 39*, 2 (2007), 32–36.
2. Commission on Technology, Gender, and Teacher Education. *Tech Savvy: Educating Girls in the New Computer Age*, American Association of University Women, 2000.
3. Committee on Information Technology Literacy, National Research Council. *Being Fluent with Information Technology.* The National Academies Press, 1999.
4. Felleisen, M., Findler, R.B., Flatt, M., and Krishnamurthi, S. *How to Design Programs: An Introduction to Programming and Computing.* MIT Press, 2001.
5. Margolis, J. and Fisher, A. *Unlocking the Clubhouse: Women in Computing.* MIT Press, 2001.
6. Shackelford, R.L. *Introduction to Computing and Algorithms.* Addison Wesley, 1997.
7. Sloan, R.H. and Troy, P. CS 0.5: A better approach to introductory computer science for majors. *ACM SIGCSE Bulletin 40*, 1 (2008), 271–275.

Mark Guzdial (guzdial@cc.gatech.edu) is a professor in the College of Computing at Georgia Institute of Technology in Atlanta, GA.