BY
KEITH DEVLIN,
Guest Editor

WHY UNIVERSITIES REQUIRE COMPUTER SCIENCE STUDENTS TO TAKE MATH

The main benefit of learning and doing mathematics is that it develops the ability to reason about formally defined abstract structures like those in computer science and its applications.

SOME YEARS AGO, I GAVE A LECTURE TO THE COMPUTER SCIENCE DEPARTMENT AT THE UNIVERSITY OF LEEDS IN ENGLAND. KNOWING MY BACKGROUND IN MATHEMATICS—MATHEMATICAL LOGIC, IN PARTICULAR—THE AUDIENCE EXPECTED IT WOULD BE FAIRLY MATHEMATICAL, AND ON THAT OCCASION THEY WERE CORRECT. AS I GLANCED AT THE ANNOUNCEMENT OF MY TALK POSTED OUTSIDE THE LECTURE ROOM, I NOTICED THAT SOMEONE HAD ADDED SOME RATHER TELLING GRAFFITI. OVER THE FAMILIAR HEADER "ABSTRACT" ABOVE THE DESCRIPTION OF MY TALK, THIS PERSON HAD SCRAWLED THE WORD "VERY."

ILLUSTRATION BY JEAN-FRANÇOIS PODEVIN

A cute addition. But it struck me then, and does still, that it spoke volumes about the way many computer science students view the subject. To the graffiti writer, operating systems, computer programs, and databases were (I assume) not abstract but real. Mathematical objects, in contrast, the graffiti-writer likely believed—and I have talked to many students who feel this way—are truly abstract, and reasoning about them is an abstract intellectual pursuit. Which goes to show just how good we humans are (perhaps also how effective university professors are) at convincing ourselves (and our students) that certain abstractions are somehow real.

The truth is, of course, that computer science is entirely about abstractions. The familiar sleek metal boxes don't, in and of themselves, compute. As electrical devices, if they can be said to do anything, it's physics. It is only by virtue of the way we design their electrical circuits that, when the current flows, obeying the laws of physics, we human observers pretend they are performing reasoning (following the laws of logic), numerical calculations (following the laws of arithmetic), or searches for information. True, it's a highly effective pretense. But just because it's useful does not make it any less a pretense.

Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner. As it happens, that is something we humans have been doing successfully for more than three thousand years. We call it mathematics.

This suggests that learning and doing mathematics could play an important role in educating future computer professionals. But if so, then what mathematics? From an educational point of view, in order to develop the ability to reason about formal abstractions, it turns out to be largely irrelevant exactly which abstractions are used. Our minds, which evolved over many tens of thousands of years to reason (largely imprecisely) about the physical world, and more recently the social one, find it extremely difficult accepting formal abstractions. But once we have learned how to reason precisely about one set of abstractions, it takes relatively little extra effort to reason about any others.

But surely, you might say, even if I'm right, when it comes to training computer scientists, it makes sense to design educational courses around the abstractions they will actually use after graduating and going to work for IBM, Microsoft, or whoever. Maybe so (in fact no, but I'll leave that argument to another time). But who can say what the dominant programming paradigms and languages will be four years into the future? Computing is a rapidly shifting sand. Mathematics, in contrast, has a long history and is stable and well tested.

Sure, there is a good argument to be made for computer science students studying discrete mathematics rather than calculus. While agreeing with this viewpoint, however, I personally find it is often overplayed. Here's why.

√ ONCE WE HAVE LEARNED HOW TO REASON PRECISELY ABOUT ONE SET OF ABSTRACTIONS, IT TAKES RELATIVELY LITTLE EXTRA EFFORT TO REASON ABOUT ANY OTHERS.

A common view of education is that its main aim is the acquisition of knowledge through the learning of facts. After all, for the most part that is how we measure the effectiveness of education, testing students' knowledge. But it's simply not right. It might be the aim of certain courses, but it's definitely not the purpose of education. The goal of education is to improve minds, enabling them to acquire abilities and skills to do things they could not do previously. As William Butler Yeats put it, "Education is not about filling a bucket; it's lighting a fire." Books and CDs store many more facts than people do—they are excellent buckets—but that doesn't make them smart. Being smart is about doing, not knowing.

A number of studies by education researchers have shown that if you test university students just a few months after they have completed a course, they will have forgotten most of the facts they had learned, even if they passed the final exam with flying colors. That doesn't mean the course wasn't a success. The human brain adapts to intellectual challenges by forging and strengthening neural pathways, and these pathways remain long after the "facts" used to develop them have faded away. The facts fade, but the abilities remain.

If you want to prepare people to design, build, and reason about formal abstractions, including computer software, the best approach is to look for the most challenging mental exercises that force the brain to

master abstract entities—entities that are purely abstract—and cause the brain the maximum difficulty to handle. Where do you find this excellent mental training ground? In mathematics.

Software engineers may well never apply any of the specific theorems or techniques they were forced to learn as students (though some surely will, given the way mathematics connects into most walks of life in one way or another). But that doesn't mean those math courses were not important. On the contrary. The main benefit of learning and doing mathematics is not the specific content; rather it's the fact that it develops the ability to reason precisely and analytically about formally defined abstract structures.

In this special section, six professors of computer science give their own particular slants on the reasons mathematics is an important component of a computer science education. Together, their articles make a strong case. Yes, we all know of individuals with no mathematics education beyond high school who have developed highly successful computer programs. That success does not imply mathematics is not important for computer science or to one's ability to write innovative or bug-free code. A more plausible inference is that with a more substantial mathematical background, these successful individuals might have been even more successful.

Kim Bruce et al. argue that knowledge of, and proficiency in, discrete mathematics, in particular, is essential for practicing computer professionals. Not so much because they are likely to have to apply any particular theorem or method—though the authors do provide some specific examples. Rather, because, they say, "One of the most important goals for a college or university education is to provide the foundations for further learning." Specific techniques, either in mathematics or in any other discipline, they say, can always be learned—and are arguably best taught—through on-the-job training as needed. University education, on the other hand, should aim to provide a sound base preparing the way for subsequent acquisition of specific skills. Or, as the authors themselves put it, "Traditional university education provides just-in-case learning rather than the just-in-time learning provided by on-the-job training." They conclude by saying, "We know that mathematical thinking will be of use; we just don't know exactly when or what form it will take."

Peter Henderson sets his sights on software engineering, which he defines as "an emerging discipline that applies mathematical and computer science principles to the development and maintenance of software systems." To Henderson, writing software is analogous to the more established physical engineering disciplines, including chemical, civil, electrical, and mechanical. This makes the importance of mathematics—or at least mathematical thinking—self-evident. As he observes, "All engineering disciplines require developing and analyzing models of the desired artifact… Abstract modeling and analysis is mathematical." As for Bruce and his co-authors, Henderson still must answer: "What math?" Like them, Henderson plumps for discrete mathematics, especially logic.

Admittedly, the importance of discrete mathematics already makes software engineering quite unlike the other engineering disciplines, with their heavy dependency on calculus-based, continuous mathematics. But is software development an engineering discipline? This question leads Henderson to start off by reformulating the original question. The bulk of his article is devoted to establishing an affirmative answer—or at least making the case that the answer should be affirmative. He has played an active role in the development and subsequent updating of the ACM/IEEE *Computing Curricula 2001* and, not surprisingly, draws on that background in making his case.

Finally, Vicki Almstrum shines a very different light on the issue. Drawing on a survey she administered to 500 computer professionals, she tries to tease out what motivates individuals to study computer science in the first place. Is it the gadgetry of computing and a desire to make things—put crudely, to write code that does stuff—or is it more an intellectual activity, akin to mathematics, even a branch of mathematics? For how many of us is the driving motivation to see our code work? For how many is the goal a desire to understand what's going on? Do these differences lead to distinctions between those who view mathematics as not important to computer science and those who do?

While almost 80% of Almstrum's survey participants were women, research by others has shown there is a gender difference in what attracts people to enter or remain in the computer field, with men tending to be attracted more by "building and doing" and women more by "understanding." Nevertheless, Almstrum's survey serves to raise awareness of the breadth and complexity of why individuals become computer professionals. Perhaps of greatest relevance to the focus of this special section, only 2% of those responding to the survey felt that a good understanding of mathematics was not helpful in computer science. C

Keith Devlin (devlin@csli.stanford.edu) is Executive Director of the Center for the Study of Language and Information and a founding member of the Media X program, both at Stanford University, Stanford, CA.