The Roles of Mathematics in Computer Science

Douglas Baldwin • Henry M. Walker • Peter B. Henderson

There is evidence that the day-to-day practice of computer science involves little if any use of mathematics, despite numerous connections between the disciplines. This gap between math's practical and intellectual roles in computer science leads to an awkward place for mathematics in undergraduate computer science curricula—required mathematics courses align poorly with the needs of computer science, and students study lots of math but relatively few computer science courses use it. Computer science graduates are therefore unwilling and unable to apply mathematics on the job. Fortunately, small local changes can strike directly at major contributors to the problem.

INTRODUCTION

Scientific and engineering disciplines generally are closely coupled to mathematics. The natural sciences make mathematical models of the phenomena they study; both the natural and social sciences rely on statistics to tease meaning out of raw data; engineers depend on mathematical models at all stages of system design, construction, and maintenance. The one pair of exceptions to this rule appears to be computer science and software engineering. Practicing software developers make little use of mathematics [20, 31], and conventional wisdom says the same of computer science students. Yet it would be very strange if the relationship between computer science, software engineering, and mathematics were really as loose as it seems. At the very least it would be suspicious for computer science and software engineering to be the only non-mathematical members of the science and engineering family; at the worst it would be downright dangerous for the disciplines to reject methods that characterize the fields whose names they use.

This paper argues that, although the day-to-day practice of computing often requires little if any mathematics, there are nonetheless important connections between computer science, software engineering, and mathematics. The next section discusses the roles mathematics plays in computer science, including how specific mathematical topics interact with specific computer science topics, and how mathematical reasoning complements computer science reasoning. The third section explores the role mathematics plays in computer science education and analyzes the disparity between its role in the general discipline and its role in education. A brief conclusion then summarizes the main points and their implications for computer science curricula.

Although the rest of the paper focuses on "computer science," we use the term generically rather than to identify a single precise discipline: our ultimate concern is with the education of computing professionals, most of whom still receive that education through a program that identifies itself as "computer science." Our argument and conclusions apply to software engineering as well as to computer science.

MATHEMATICS' ROLE IN COMPUTER SCIENCE

Computer scientists use math in their professional lives in several ways. First, mathematics provides the theoretical basis for many subfields of computer science, and important analytic tools for others; computer scientists thus apply specific mathematical topics to specific computing problems. More generally, mathematics provides a framework for reasoning about computing and computing problems, and even more broadly, provides a mental discipline for solving those problems.

Specific Mathematical Topics

It is reasonably easy to identify individual pieces of mathematics that find use in specific areas of computing (e.g., "Boolean algebra can be used to manipulate conditional expressions"). What is hard is identifying an appropriate level of detail at which to analyze computing's uses of mathematics, and imposing some standard of completeness on that analysis. In an attempt to do these things, we used the 2012 ACM computing classification system [19] as a guide to the subject matter that makes up "computer science," and attempted to identify the mathematics that is important in each top-level category. To recognize "important" mathematics, we looked for references to mathematics in the classification system itself, we consulted books that are considered definitive references within some of the categories [7, 10, 14, 28], and we drew on our own knowledge. Note that we excluded the "general and reference" category from our analysis as orthogonal to that analysis, and "mathematics of computing" because our goal was essentially to match its elements to the other categories. The value of this approach is that it brings some objectivity to the process of aligning mathematics and computer science; the price of that objectivity is that everyone will no doubt see ways in which the alignment differs from their personal perceptions. We offer the approach as a first step in developing a comprehensive understanding of what math is important for computer science, but certainly don't expect our analysis to be the last word on the subject.

Figure 1 shows our association of mathematical topics (blue) with computing classification system categories (yellow). We show associations only where a mathematical topic occurs particularly prominently or in multiple places within a category. Even so, because the categories are very broad, it is entirely possible to work within a category without encountering all of the mathematics we associate with it. This phenomenon is particularly pronounced in the "computing methodologies" category, which includes a mathematically diverse set of topics. On the other hand, as looking at ACM article classifications will show, the classification system is designed so that most computing activities have multiple, intersecting, classifications. Computer scientists thus seldom work within just one classification category at a time, but more commonly work in several simultaneously, and therefore may draw on the mathematics associated with several categories at once.

Figure 1 makes it clear that there are myriad connections between mathematics and computer science. However, the degree of connectivity is not uniform. The most-connected mathematical topic by far is probability and statistics, reflecting the widespread appearance of performance and reliability analyses in many computer science categories (and also see Sahami [25] for additional reasons why probability is important in computer science). Other highly connected mathematical topics include propositional/predicate logic (which we consciously lump together because it is hard

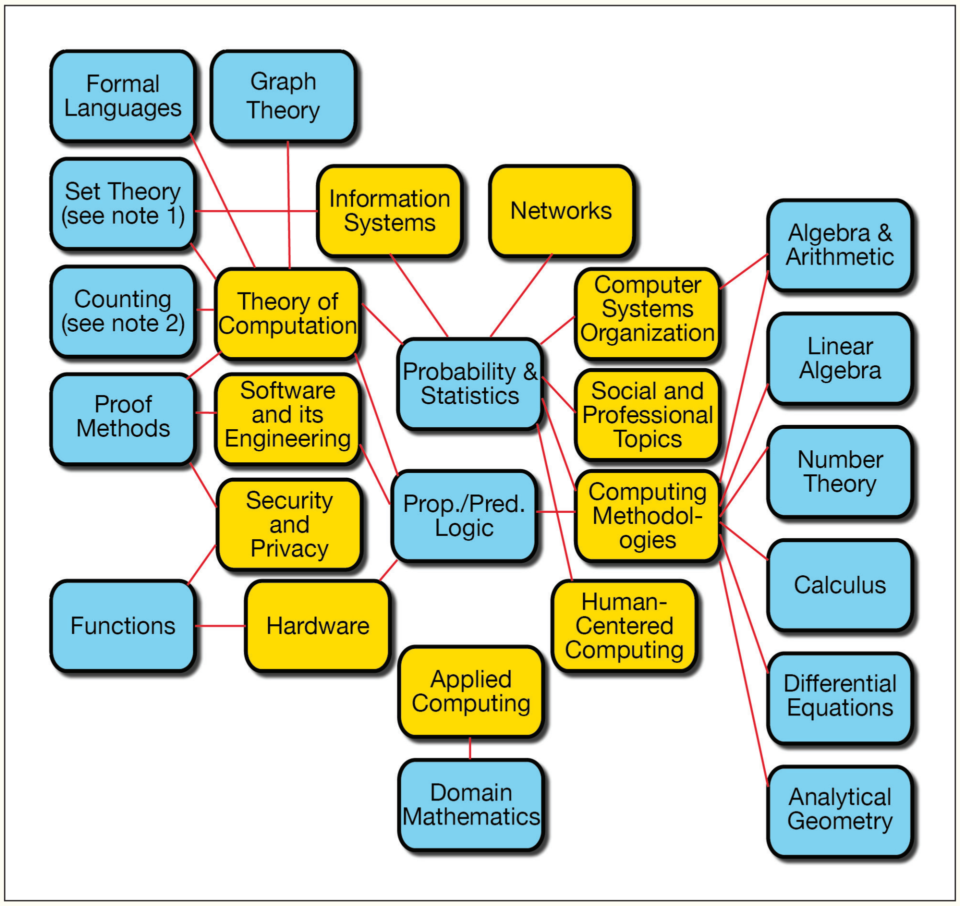

Figure 1: Mathematics associated with computer science areas. (**Note 1:** We include in "set theory" functions and relations as associations between sets, as well as the basic definitions of and operations on sets. **Note 2:** "Counting" includes standard combinatorics concepts such as combinations and permutations, as well as such techniques for evaluating counts as summations and recurrence relations.)

to separate them in the computing classification system) and proof methods. Many of the associations involving these two topics arise because formal approaches to specification or verification are pervasive in the computing classification scheme - for specification or verification of network protocols, software designs and implementations, security properties, etc. These formal techniques lead to associations with logic, proofs, or set theory according to how the formal tools are used.

Certain computing topics also have much higher connectivity than others. Not surprisingly, "theory of computation," which includes mathematical models of computation plus analysis of algorithms, is connected to many mathematical topics. The high connectivity of "computing methodologies" is perhaps more surprising - it is due to the category being a broad one that contains many subtopics.

[P]eople who create new algorithms or designs need some ability to independently apply mathematical techniques, and at the high-math end of the spectrum, those who conduct research in an area need a deep ability to work with its mathematics.

Further comments seem appropriate for the "domain mathematics" topic linked to "applied computing." All applied computing is in some domain that has its own mathematical tools or foundations, and at some level the people involved in any applied computing project have to understand that mathematics.

In reviewing Figure 1, note that no attempt is made to define the depth of mathematical ability required to work in each computing category. Different people with dramatically different degrees of mathematical skill can work in the same category, and the same person may deploy different skills on different tasks in a category. At the low-math end of the scale, writing code from classic algorithms or external specifications likely requires only broad familiarity with the terminology and overall concepts of a mathematical area. Lethbridge's 1998 survey of the knowledge required by software engineers [20], supported by Surakka's smaller survey five years later [31], suggests that many productive software professionals operate near this level of mathematical knowledge. However, people who create new algorithms or designs need some ability to independently apply mathematical techniques, and at the high-math end of the spectrum, those who conduct research in an area need a deep ability to work with its mathematics.

Mathematics and Reasoning

Many activities within computing require practitioners to analyze problems and potential solutions logically and carefully - often applying tools and techniques from mathematics. For example,

- Whenever problems are put forward or solutions proposed, users should ask what assumptions are being made and how those assumptions might impact any results obtained or program behaviors.
- When an algorithm is proposed as a solution to a problem, developers and researchers must determine whether the algorithm is correct and uses resources efficiently.
- When programs are put forward as implementations of algorithms, testing organizations and users may formally as well as empirically verify that the software behaves according to identified specifications. (Instances of required formal verification do exist - for example electronic gambling devices are subject to mathematically defined fairness requirements in some jurisdictions [22]; one of the authors recently saw a position announcement from a gaming company seeking someone to "create, test, and analyze new games" but also to "compose ... mathematical proofs for game submissions to ... regulators" [27].)
- When several potential solutions are suggested for a problem, practitioners should be able to analyze the relative advantages and disadvantages of those solutions under varying assumptions.

Bruce [5] provides several specific examples of mathematical reasoning in these and other computing activities. The bottom line is that computing professionals need to reason logically - not just in hypothetical or classroom settings, but in real research and development projects.

More abstractly, there are close connections between problem solving in computer science and in mathematics. Devlin [9] observes that computer science is a mass of abstractions built on other abstractions, and that mathematics is the age-old language and practice of abstraction. Ralston [24] argues that even if computing professionals seldom use math explicitly, the logical thinking central to mathematics is also central to computing. In her widely cited "computational thinking" paper [35], Jeannette Wing develops this idea in depth. She credits computer science with a distinctively powerful approach to problem solving, which, among other defining characteristics, "complements and combines mathematical and engineering thinking." The term "computational thinking" is broadly defined in her paper, and has since been applied by other authors to almost any thought process remotely associated with computing. Yet the key point of her argument and that of others remains: the general reasoning and problem solving skills characteristic of computer science are powerfully effective and closely interwoven with those of mathematics.

Altogether, computing researchers and practitioners must be able to reason about problems and their solutions in both informal and formal ways. This reasoning involves analysis, synthesis, and evaluation - the three deepest levels of understanding and mastery identified by Bloom [4]. In many cases, this level of reasoning is an integral part of the creative problem-solving process that engages computing professionals. Although some software developers may seldom use math explicitly, there are rich connections between it and computer science that can enhance this reasoning.

MATHEMATICS' ROLE IN COMPUTER SCIENCE EDUCATION

As seen in the previous section, the relationship between mathematics and computer science has two faces: many software engineers perform well without relying on mathematics, while at the same time there are rich connections between the fields that can be exploited by those prepared to do so. How then does, and should, mathematics fit into undergraduate computer science curricula?

The Current State of Mathematics in Computer Science Curricula

As an indication of what strong undergraduate computer science programs around the world consider appropriate mathematics content, we examined the mathematics requirements of 25 of the first 26 programs (we were unable to find an English description of one) listed in US News and World Reports' best universities in computer science in 2012 [34]. We emphasize that this is not a statistically rigorous study of what "typical" computer science undergraduates experience, but rather an effort to get an international selection of high-quality programs that can provide a general sense of how math is integrated into computer science education. However, the amount of math in the high-quality programs is consistent with the amount of math required in CAC-accredited US computer science programs surveyed in the late 1990s [21], and our personal experiences suggest that observations about the high-quality programs also apply to other programs.

Table 1 provides a summary of how many programs require what sorts of math. The table shows a slight inconsistency between mathematics requirements and the actual connections between math and computer science from Figure 1. Almost all the programs require students to study discrete mathematics, which is appropriate as it includes much of the foundational mathematics for computer science (e.g., logic, some proof methods, set theory, etc.). Three programs, however, do not require this foundation. Furthermore, probability and statistics is the least commonly required of the topics we looked for, despite its heavy use in computer science. Calculus, which has relatively limited uses in computer science, is required almost as often as discrete mathematics, and more often than probability and statistics. Lethbridge's survey [20] noted similar inconsistencies, in particular finding calculus to be one of the most taught but least important subjects in the software engineers' repertoire.

TABLE 1. MATHEMATICS REQUIREMENTS OF 25 HIGH-QUALITY COMPUTER SCIENCE PROGRAMS.

Topic	Number of Programs Requiring
Discrete Math	22
Probability and Statistics	15
Calculus	21

The number of mathematics courses required by the programs varies greatly, from a minimum of 1 to a maximum of 8, with a mode of 5. Figure 2 shows the complete distribution. Programs at the higher end of the distribution often require multiple courses in calculus, linear algebra, and/or differential equations. Very few programs require more than one course in discrete mathematics or in probability and statistics. High numbers of required math courses therefore do not indicate extensive study of the mathematics central to computer science.

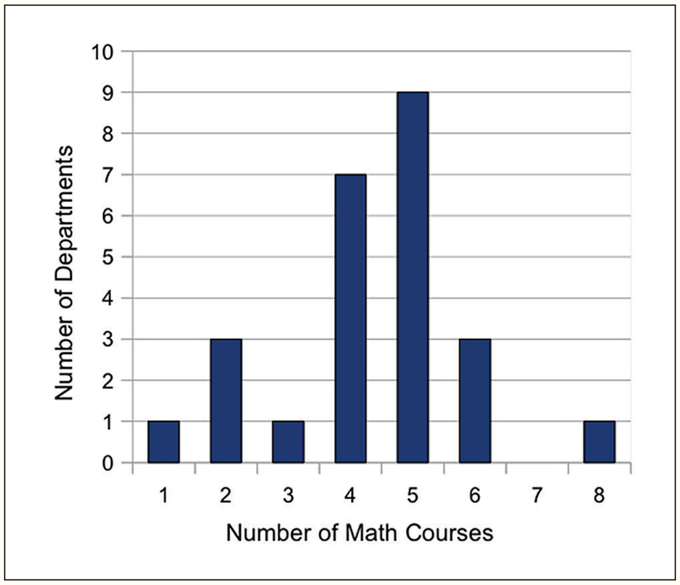

Figure 2: Number of mathematics courses in sample computer science programs.

The prominence of calculus in computer science programs is puzzling. Some amount of calculus can be explained by the fact that one or two calculus courses are a prerequisite for other mathematics in most schools. However, many programs also require multivariable calculus, differential equations, etc., far exceeding what is plausibly necessary to study the mathematics more central to computer science. This amount of calculus may be due to programs being housed in schools of engineering, or being historically derived from engineering programs, which traditionally require substantial amounts of calculus.

More important than the quantity or type of mathematics required in undergraduate computer science programs is its utilization outside of mathematics courses. Folk wisdom among computer science educators seems to be that there is little such utilization. This is certainly our own sense from reading the computer science education literature and talking to other educators, participating in program reviews, etc. Further, Cohoon and Knight argue for better integrating discrete mathematics and software engineering courses by asserting (albeit without citing evidence) that…

> The standard discrete mathematics courses provide minimal motivation and material application. The standard software engineering courses provide little if any application of discrete mathematics, and the formal method courses are usually optional and late in the education of a Computer Science major. [6]

The prominence of calculus in computer science programs is puzzling. Some amount of calculus can be explained by the fact that one or two calculus courses are a prerequisite for other mathematics in most schools.

To a certain extent we can compare this folk wisdom to reality. We have read course descriptions from some of the high-quality computer science programs discussed above (specifically MIT, Stanford, Oxford, the Hong Kong University of Science and Technology, and the Ecole Polytechnique Fédéral de Lausanne), as well as from one of our own institutions (Geneseo), and the model course descriptions that accompany the 2001 ACM/IEEE computer science curriculum guideline [17] and its 2008 update [16]. Based on these sources, we observe certain patterns in the utilization of math in computer science programs. Most strikingly, the courses that visibly use mathematics concentrate in certain subfields of computer science. Subfields taught with some reference to mathematics include algorithms and data structures, computer graphics, artificial intelligence, and CS2-level courses to the extent that they lay the groundwork for later study of algorithms and data structures. Electrical engineering courses, when included in computer science programs, are also heavily mathematical. Courses with little evident use of mathematics tend to fall in such subfields as introductory programming, computer systems (architecture, operating systems, networking), and software engineering. This pattern reflects some, but not all, of the connections identified in Figure 1. In particular, the courses that use mathematics lie in the heavily connected "theory of computation" and "computing methodologies" categories, while the courses that make little use of mathematics lie elsewhere. However, these non-mathematically-oriented courses still lie in categories that have mathematical connections (e.g., to probability and statistics or to logic), and those connections are not evident in course descriptions. Numerically, between ⅓ and ½ of the courses in each program seem to use math (excluding courses whose main goal is to teach math), with occasional programs extending as low as ¼ or as high as ¾. There may be some geographical or cultural variation, with programs in Europe perhaps making more use of mathematics, and programs in the United States making less. Taken all together, these observations suggest that the actual use of mathematics in computer science curricula is not as limited as the folk wisdom implies, although it is far less than it could be.

Is There a Problem?

Computer science programs require students to take a reasonable number of mathematics courses, but much of that mathematics is of limited relevance to computer science as a whole. The remaining mathematics is, on balance, under-utilized in the computer science curriculum, with large sub-fields of the discipline being taught with little, if any, reference to it. Such a curriculum produces graduates who are ill-equipped to use mathematics in their professional careers, and who see little need to do so. At least on the surface, this is not a problem—those graduates do, after all, get and keep jobs in computing. However, at a deeper level, inability to apply mathematics to computing constrains graduates' long-term potential.

Within a learning environment, understanding typically starts at the beginning levels of Bloom's taxonomy—knowledge of specifics (e.g., jargon, truth tables, formal rules of logic), comprehension (e.g., paraphrasing formal rules), and simple applications. Computer science, whether in its mathematical aspects or not, is no exception. Such foundational work is needed as a base for reasoning about algorithms, programs, systems, etc. However, this elementary level of reasoning and understanding is insufficient for actually using computer science in the real world. Students must learn much more than the mechanical application of routine steps. Such learning happens when later courses build upon the foundation laid by introductory ones and provide practice at deeper levels of analysis in both structured and open-ended settings. Although such analysis may not be part of every discussion of every topic in upper-level courses, students need to experience it repeatedly and in multiple contexts. When undergraduate computer science programs fail to do this with the mathematics they require, they limit graduates' ability to use mathematics in either subsequent study or employment.

In the workplace, mathematics and mathematical methods are increasingly important in software development. For example, the IEEE Computer Society's Certified Software Development Associate/Professional (CSDA/CSDP) examinations [15] include mathematics roughly equivalent to that in the 2001 ACM/IEEE computer science curriculum recommendations [17], plus additional probability and statistics. Formal methods are slowly gaining traction in the software industry; of particular note, programming for concurrency is sweeping through the industry, and automatic model checking is an increasingly vital tool for coping with the subtle timing and synchronization bugs that concurrency brings [18, 8]. While there is ample need for programmers who can write code to given specifications, the more senior developers who produce those specifications often need facility with the mathematics of the application domain [30]. Undergraduates who continue to graduate school, particularly at the doctoral level, will find themselves in a world of mathematical sophistication unimaginable from the undergraduate perspective - the most pronounced example is probably the study of programming language theory as a non-mathematical descriptive activity in undergraduate texts such as [26], but as an entirely mathematical exercise in modeling language semantics in such graduate texts as [33].

Are There Solutions?

A number of measures can be (and sometimes have been) tried in order to close the gap between the role mathematics plays in computer science and the way that role is conveyed in undergraduate computer science education. None of these measures, however, shows clear promise of success. For example…

- *Computer science educators can be alerted to the problem.* Three decades ago Ralston argued that undergraduate computer science programs need mathematics support, and provided a detailed list of mathematics topics [23]; a decade and a half ago Lethbridge called attention to the amount of inappropriate math covered in computer science curricula [20]; at approximately the same time an informal working group organized itself online to advocate for the importance of math in computer science education [2]; a few years later Tucker argued that computer science curricula were "math-phobic" [32]. Although there have been some changes in mathematics' treatment in computer science curricula over this time, notably the incorporation of discrete structures into the ACM/IEEE CS curriculum recommendations in 2001 [17], most of the above authors' criticisms are still valid today. Raising awareness of the problem does not solve it.
- *Mathematics requirements can be used more efficiently.* Computer science curricula generally have room for a number of mathematics courses, but few of those courses actually teach mathematics that is central to computer science. However, this inefficiency is difficult to correct. Prerequisite structures in departments of mathematics may require computer science students to take some amount of foundational, but not directly applicable, math, and computing programs in schools of engineering may be under explicit or implicit pressure to include mathematics that is traditional for the physical sciences even if not crucial to computer science.
- *Mathematics can be integrated into computer science courses and vice versa.* Under-utilization of mathematics in the computer science curriculum is the largest problem, and there have been many attempts to address it. In the 1980s through 1990s Henderson evolved a first course for computer science majors that emphasized mathematical methods of reasoning and problem solving as the foundation for studying computer science [12, 13]. During the early 1990s there were other efforts to integrate discrete math concepts early in the computer science curriculum, notably the two textbooks *Foundations of Computer Science* [1] and *A Logical Approach to Discrete Math* [11]. In the later 1990s and early 2000s Baldwin and Scragg developed a course that introduced much of the discrete math needed by computer scientists in the context of elementary design and analysis of algorithms [3]; Cohoon and Knight independently developed a similar set of courses for software engineers [6]. Currently, Sitaraman and his colleagues are promoting a software engineering course that emphasizes mathematical techniques for deriving correct programs [29]. None of these efforts, however, has gained traction beyond their developers. The computer science community has regarded them as interesting, and perhaps even praise-worthy, but not as paradigm-changers that must be adopted.

CONCLUSION

Computer science, like the physical sciences and traditional engineering fields, widely uses mathematics to model the phenomena it studies. Furthermore, computational and mathematical reasoning are closely connected. Yet, paradoxically, many computer science and software engineering graduates function quite well as professionals without consciously applying mathematics to their work. This paradox leads mathematics to sit uncomfortably in undergraduate computer science curricula: while most such curricula include appropriate mathematics, they often also include much mathematics that is not strongly connected to computing, and while they teach some applications of math to computing, they often overlook others. This awkward treatment of mathematics in computer science education has been surprisingly resistant to correction, for a surprisingly long time. While the precise reasons differ from institution to institution, we believe that the overarching one is that computer science faculty simply do not see the problem as urgent. And indeed, as long as computer science graduates find jobs or places in graduate schools in the field, and the field itself is growing, the problem does seem minor.

However, taking a longer view, there is reason for concern. Software development and testing are slowly adopting mathematical tools and methods, and today's graduates will need to adapt to those tools and methods throughout their careers. As graduates advance through their careers, they will come into positions where they have responsibility for system design, assessment of test results or quality metrics, selection of architectures or algorithms, and similar activities that require quantitative evaluation of data and comparison of options.

In addition, computer science is a young discipline whose definition is still in flux. It is arguably the one science that studies artificial rather than natural phenomena. The distinction between computer science and software engineering is not nearly as clear as the distinction between other sciences and their related engineering fields; indeed, some would probably question whether there is a distinction at all. The role of mathematics in computer science and computer science education is therefore tied to larger questions of where the "science" in computer science lies and what the scientific basis for software engineering is or should be. The designers and implementers of computer science curricula today will play a large role in framing these questions for future generations and establishing the groundwork for answering them.

The time has come for all of us, as computer science and software engineering educators, to reform the role of mathematics in our curricula. The authors recommend incremental reform, since past experience suggests that a sudden and universal leap in enthusiasm for mathematics in computing education is unlikely. However, individual programs can make important improvements by pruning from their requirements one or two math courses with limited applications to computer science, and replacing them with ones with more relevance, or with electives that allow students to explore connections they find pertinent. Programs, or even single instructors, can ensure that more computer science courses use mathematics to illuminate the computing material they present, even without creating new courses or changing the choices of topics studied. Individually these are small, local, and therefore relatively easy, changes; collectively, pursuing both until they gain momentum would eliminate two major reasons why our graduates lack the inclination and ability to make math part of their professional toolkit. **Ir**

Acknowledgments

The original version of this paper was written at the invitation of the ACM/IEEE CS Curriculum 2013 committee as a position statement on the role of mathematics in undergraduate computer science education. We thank members of the committee for comments on that version. The views stated in this paper are those of the authors, and do not necessarily reflect the position of the committee or its members. Thanks also to ACM *Inroads*' reviewers for helping us clarify several aspects of the present version.

References

[1] Aho, A. V. and Ullman, J. D. *Foundations of Computer Science*. (New York: Computer Science Press, 1992).
[2] Baldwin, D. and Henderson, P. B. "A working group on integrating mathematical reasoning into computer science curricula." http://www.math-in-cs.org/. Accessed 2013 April 30.
[3] Baldwin, D. and Scragg, G. *Algorithms and Data Structures: The Science of Computing*. (Hingham, Massachusetts: Charles River Media, 2004).
[4] Bloom, B. *Taxonomy of Educational Objectives: Handbook 1: Cognitive Domain*. (Longmans, Green and Company, 1956): 201-207.
[5] Bruce, K. et al. "Why math?" *Communications of the ACM*, 46, 9 (2003): 41- 44.
[6] Cohoon, J. P. and Knight, J. C. "Connecting discrete mathematics and software engineering" in *Proceedings of the Thirty-Sixth ASEE/IEEE Frontiers in Education Conference*. (New York: IEEE, 2006): M2F-13 - M2F-18.
[7] Cormen, T. et al. *Introduction to Algorithms*. 3rd ed. (Cambridge, Massachusetts: MIT Press, 2009).
[8] Desnoyers, M. "Proving the correctness of nonblocking data structures." *Communications of the ACM*, 56, 7 (2013): 62-69.
[9] Devlin, K. "Why universities require computer science students to take math." *Communications of the ACM*, 46, 9 (2003): 37-39.
[10] Foley, J. et al. *Computer Graphics: Principles and Practice*. 2nd ed. (Reading, Massachusetts: Addison-Wesley, 1990).
[11] Gries, D. and Schneider, F. B. *A Logical Approach to Discrete Math*, (New York: Springer Verlag, 1993).
[12] Henderson, P. B. "Discrete mathematics as a precursor to programming." in *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education*. (New York: ACM, 1990): 17-21.
[13] Henderson, P. B. "The role of mathematics in computer science and software engineering education." *Advances in Computers*, 65 (2005): 350-396.
[14] Hennessy, J. and Patterson, D. *Computer Architecture: A Quantitative Approach*. 3rd ed. (Amsterdam: Morgan Kaufmann Publishers, 2003).
[15] IEEE Computer Society, "TechLeader OnCourse." http://www.computer.org/portal/web/certification. Accessed 2013 April 30.
[16] IEEE Computer Society and Association for Computing Machinery Interim Review Task Force. "Computer Science Curriculum 2008: An Interim Revision of CS 2001." http://www.acm.org/education/curricula/ComputerScience2008.pdf. Accessed 2013 June 27.
[17] IEEE Computer Society and Association for Computing Machinery Joint Task Force on Computing Curricula. "Computing Curricula 2001: Computer Science," http://www.acm.org/education/education/education/curric_vols/cc2001.pdf. Accessed 2013 April 30.
[18] Jhala, R. and Majumdar, R. "Software model checking." *ACM Computing Surveys* 41, 4 (2009). doi: 10.1145/1592434.1592438.
[19] Kedem, Z. et al. eds. "The 2012 ACM Computing Classification System." http://www.acm.org/about/class/2012. Accessed 2013 April 20.
[20] Lethbridge, T. "Priorities for the education and training of software engineers." *Journal of Systems and Software*, (2000): 53-71.
[21] McCauley, R. and Manaris, B. "Computer science education at the start of the 21st century—a survey of accredited programs." in *Proceedings of the Thirty-Second ASEE/IEEE Frontiers in Education Conference*. (New York: IEEE, 2002): F2G-10 - F2G-15.
[22] Nevada State Gaming Control Board Gaming Commission. "Regulation 14." http://gaming.nv.gov/modules/showdocument.aspx?documentid=2921. Accessed 2013 July 17.
[23] Ralston, A. "The first course in computer science needs a mathematics corequisite." *Communications of the ACM*, 27, 10 (1984): 1002-1005.
[24] Ralston, A. "Do we need ANY mathematics in computer science curricula?" *inroads—the SIGCSE Bulletin*, 37, 2 (2005): 6-9.
[25] Sahami, M. "A course on probability theory for computer scientists." in *Proceedings of SIGCSE 2011, the Forty-Second Technical Symposium on Computer Science Education* (New York: ACM, 2011): 263-268
[26] Sebesta, R. *Concepts of Programming Languages*. 9th ed. (Boston: Addison-Wesley, 2010).
[27] SHFL Entertainment. "Game Designer / Mathematician." http://www.creativeheads.net/job/12078/game-designer--mathematician-in-las-vegas. Accessed 2013 July 17.
[28] Sipser, M. *Introduction to the Theory of Computation*. 3rd ed. (Boston: Cengage Learning, 2013).
[29] Sitaraman, M. et.al. "Building a push-button RESOLVE verifier: progress and challenges." *Formal Aspects of Computing*, 23, 5 (2011): 607-626.
[30] Stringer, M. Personal communication to Douglas Baldwin. Sept. 2012.
[31] Surakka, S. "What subjects and skills are important for software developers?" *Communications of the ACM*, 50, 1 (2007): 73-78.
[32] Tucker, A., Kelemen, C., and Bruce, K. "Our curriculum has become math-phobic!" in *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*. (New York: ACM, 2001): 243-247.
[33] Turbak, F. and Gifford, D. *Design Concepts in Programming Languages*. (Cambridge, Massachusetts: MIT Press, 2008).
[34] US News and World Report. "World's Best Universities: Computer Science." http://www.usnews.com/education/worlds-best-universities-rankings/best-universities-computer-science. Accessed 2013 April 28.
[35] Wing, J. "Computational Thinking." *Communications of the ACM*, 49, 3 (2006): 33-35.

DOUGLAS BALDWIN
Department of Mathematics
SUNY Geneseo
Geneseo, New York 14454 USA
baldwin@geneseo.edu

HENRY M. WALKER
Department of Computer Science
Grinnell College
Grinnell, Iowa 50112 USA
walker@cs.grinnell.edu

PETER B. HENDERSON
Department of Computer Science and Software Engineering
Butler University
Indianapolis, Indiana 46208 USA
phenders@butler.edu

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education—Computer Science Education; G.0 [Mathematics of Computing]: General.
General terms: Design
Keywords: Mathematics, Computer Science Education, Software Engineering Education

DOI: 10.1145/2537753.2537777 © 2013 ACM 2153-2184/13/12 $15.00