# Our Curriculum Has Become Math-Phobic!


**Allen B. Tucker**
Computer Science Dept.
Bowdoin College
Brunswick, ME 04011
allen@bowdoin.edu

**Charles F. Kelemen**
Computer Science Program
Swarthmore College
Swarthmore, PA 19081
cfk@cs.swarthmore.edu

**Kim B. Bruce**
Computer Science Dept.
Williams College
Williamstown, MA 01267
kim@cs.williams.edu



## Abstract

The paper [2] argued that mathematical ideas play an important role in the computer science curriculum, and that Discrete Mathematics needs to be taught early in the computer science curriculum. In this follow-up paper, we present evidence that computer science curricula are drifting away from a fundamental commitment to theoretical and mathematical ideas. We propose some actions that can be taken to help reverse this drift.


## 1 Evidence I: Industry and Faculty Views

This section presents evidence from the industry and faculty sides that the curriculum has become math-phobic.

### 1.1 Questioning Mathematics Requirements

Robert Glass has written several books in the software field. He is Editor-in-Chief of the Journal of Systems and Software, Editor/Publisher of The Software Practitioner, and author of the Practical Programmer column in the Communications of the ACM.

In his article "How Important is Mathematics to the Software Practitioner?", Glass says, "Academics, I would assert, are largely in the 'math is vital' camp. Practitioners, I would further assert, are in the 'math is of little consequence' camp."

This article also reports from a survey conducted by Timothy Lethbridge [4], who has studied the difference between what software engineers remember being taught in academe, and their perceptions of what is important to them in practice.

Many subjects are covered in this survey; Lethbridge was interested in the broader questions of priorities in engineering education. However, one of the most striking findings of his research involved the similarity between what practitioners felt they "learned most in their formal education" and what they felt to be the "least important" topics.

Here are the topics that Lethbridge found to be "learned most in their formal education." Note that most, although not all, of these topics are mathematical.

1. specific programming languages
2. differential and integral calculus
3. linear algebra and matrices
4. probability and statistics
5. data structures
6. physics
7. differential equations
8. set theory

   ...

And here are the "least important topics" identified by the same practitioners (the number on the left represents the standing of the topic among a set of 75 topics; the higher the number, the lower the standing):

72. differential equations

68. combinatorics

67. differential and integral calculus

56. linear algebra and matrices

52. computational methods and numeric problems

48. set theory

39. predicate logic

31. probability and statistics

...

These practitioners identified the "most important" topics to their professional work as follows:

1. specific programming languages

2. data structures

3. software design and patterns

4. software architecture

5. requirements gathering and analysis

6. human computer interaction / user interfaces

7. object oriented concepts and technology

8. ethics and professionalism

...

Lethbridge concludes, "Relatively little mathematics turns out to be important for software engineers in practice, and it tends to be forgotten." ... "If we are to continue to teach the amount and type of mathematics [that we presently teach], we must justify it by other means than saying it is important to a software developer's work: our data show that is normally not ... At the very least, educators should look at the mathematics elements of computing [curricula] and examine how they can be made more relevant..."

In response to his title question, "How important is mathematics to the software practitioner?", Glass reports "Not very much" according to Lethbridge. Glass then asks, "What should we do about that?" We will speak to that in a later section of this paper.

As CS is buffeted by IT, IS, MIS, multimedia, WEB design, etc. some argue that there should be a common core. Since some of these disciplines do not feel math is very important, there is pressure to remove mathematical topics and rigor from early CS courses.

### 1.2 A Small Survey of Educators

To determine if educators were feeling this pressure, Kelemen and Tucker surveyed the attendees at their talk, "Has Our Curriculum Become Math-Phobic? (An American Perspective) " at ITiCSE 2000 [2]. The questions with responses tallied are given in Figure 1.

While this is not a completely representative survey of CS educators, it does reveal some interesting information. First, about *half* of the respondents agreed that they felt pressure to make the CS curriculum less mathematical and that the curriculum is, in fact, becoming less mathematical. Moreover, *most* of the respondents agreed that the discrete mathematics course should be a prerequisite for the data structures course and that the CS curriculum should not become less mathematical in the future.

## 2 Evidence II: Data Structures Textbook Content

To determine the extent to which mathematical topics are integrated into the computer science curriculum, we reviewed the current texts that are used in the second course in the curriculum - the data structures course. Our view is that the following topics in discrete mathematics are needed for this course, and their use should be integrated into the data structures subject matter.

- Complexity of algorithms: growth of functions, O notation, worst case analysis, etc.
- Correctness of algorithms: preconditions, postconditions, and loop invariants
- Recursion, recurrence relations and inductive proofs

In 1999, 27 different texts were used by students in the data structures course [5]. A rough estimate of their relative market shares and amount of mathematical treatment for the five most widely used texts (identified as A - E to preserve anonymity) is given in Figure 2.

This figure shows that the five most popular texts were used by about 55.8 per cent of all students enrolled in the data structures course.

We also estimated the number of pages in these texts that used or integrated the mathematical topics listed above. These estimates were derived by examining the table of contents and index of each text and counting the number of pages allocated for each topic. These estimates show that these five data structures texts give some attention to the mathematical treatment of complexity and recursion, and little or no attention to correctness and induction proofs. None of these five texts presents these mathematical topics in a way that is convincingly integrated among the main topics in the study of data structures. We also believe that other texts for this course give similar or even more limited levels of treatment to these mathematical topics.

Further evidence for the scarcity of mathematical treatment in the data structures course appears in the prefaces of these texts. In one instance, the author emphasizes that the discrete mathematics course is not a

| | | | | |
|---|---|---|---|---|
| Your employment category: | Academia: 62 | Industry: 0 | Other: 0 | |
| Location of your employer: | US: 20 | Europe: 30 | Asia: 3 | Other: 9 |
| Do you feel pressure to make the CS curriculum less mathematical at your institution? | Yes: 30 | No: 29 | N/A: 3 | |
| Is the CS curriculum becoming less mathematical? | Yes: 33 | No: 24 | N/A: 5 | |
| Should the curriculum become less mathematical? | Yes: 14 | No: 42 | N/A: 6 | |
| Should Discrete Math topics be a prerequisite for the Data Structures course? | Yes: 43 | No: 13 | N/A: 6 | |
| Should Discrete Math topics be better integrated in the Data Structures course? | Yes: 42 | No: 13 | N/A: 7 | |

Figure 1: Survey answers from ITiCSE

| Data Structures Text | A | B | C | D | E | 22 Others | Total |
|---|---|---|---|---|---|---|---|
| 1999 Sales | 8,975 | 4,491 | 3,080 | 2,580 | 1,989 | 16,721 | 37,836 |
| Percent of Total | 23.7 | 11.9 | 8.1 | 6.8 | 5.3 | 44.2 | |
| Total Pages | 770 | 790 | 750 | 550 | 750 | | |
| – Complexity of algorithms | 35 | 74 | 20 | 40 | 30 | | |
| – Correctness of algorithms | 0 | 8 | 0 | 0 | 0 | | |
| – Recursion | 44 | 90 | 30 | 28 | 15 | | |

Figure 2: Leading Data Structures Texts' Mathematical Treatment

prerequisite for using the text. That text does contain several mathematical proofs (using induction), but the author notes that those sections can easily be skipped. In another instance, the preface contains no mention of discrete mathematics at all. That text gives a gentle introduction to algorithm efficiency in a later chapter and no proofs appear anywhere in the text. In a third instance, the author had moved all of the mathematical material from an earlier edition (where it had been well-integrated) to an appendix, noting that that material had become less central to the study of data structures than in earlier times.

Finally, we suspect that instructors who teach the data structures course tend to omit the modest amount of mathematics that is found in these texts from their syllabi, for a variety of reasons (including the guidance of the authors themselves). Notably, none of these books requires that students complete a discrete mathematics course as a prerequisite. Thus, computer science students complete their data structures coursework with little or no understanding or appreciation for the relevance of (discrete) mathematics in computer science.

## 3 Evidence III: Curricula 2001 and Mathematics

[Note: All references to KU's of CC2001 refer to the unpublished June 5 draft.]

The Joint IEEE Computer Society/ACM Task Force on the "Year 2001 Model Curricula for Computing" (CC-2001) was formed to update the 1991 curricula recommendations. The task force originally planned on offering recommendations encompassing all of the computing disciplines (MIS, IS, CS, SWE, CE, etc.), but has recently decided to focus its immediate efforts on a report covering a core curriculum in Computer Science. Similar chapters will appear later on other subdisciplines.

While the task force's work is continuing, their "Strawman" report from March 2000 provides a clear indication of the committee's thinking. Two main goals shine through. The first is a desire to keep the core curriculum small, with material that can be fit into 6 or 7 courses (not counting required mathematics courses). The second is an effort to expand the scope to incorporate new material introduced in computing over the last decade or more. As a result of these goals, many of the traditional areas of computer science have been cut back substantially in the strawman report. The overall number of hours in the core has declined by 15%, with that in traditional core areas cut more because of the addition of areas such as Net-Centric Computing and Human-Computer Interfaces.

Because of repackaging, it is difficult to compare Curricula '91 with the current Curriculum 2001 proposal. However, if we restrict ourselves to the mathematically and theoretically oriented material, there are very noticeable differences.

Much of the theoretical material in the Algorithms (AL) section has been omitted in the new curriculum. The Programming Languages (PL) material has been cut from 46 hours to 5 hours, omitting all theoretical material as well as much non-theoretical material (though some of the non-theoretical material appears elsewhere).

Both the old and new curricula introduce the use of big "O" notation and use it in analyzing algorithms. However there is no longer any mention of complexity classes like P and NP in the core (deleting the old AL5, which was allotted 4 hours).

Core units on Computability and Undecidability (AL7: 6 hours), Finite State Automata and Regular Expressions (PL7: 6 hours), Context-Free Grammars and Pushdown Automata (PL8: 4 hours) have been compressed into a new AL5: Basic Computability with a total of 6 hours; a loss of 10 hours of theoretical core material. Programming Language Semantics (PL10: 2 hours) has also disappeared from the core.

The area of Numerical and Symbolic Computation (7 core hours) has disappeared from the core, replaced by Computational Science with no core hours. This seems to have resulted in no coverage of round-off and similar representation-dependent errors nor of iterative approximation methods.

It is perfectly reasonable to argue that the omitted material is no longer as relevant for all computer science undergraduates. However, we find it of great concern that this has resulted in the omission of large proportion of theoretical material, with the addition of very little new in that area. In fact the only new material that seemed to be theoretically oriented was IS3: Knowledge representation and reasoning in Intelligent Systems (formerly AI), adding 4 hours. This KU includes material on the use of propositional and predicate calculus and automated theorem proving.

Perhaps even more distressing are more subtle changes which de-emphasize mathematical reasoning. AL 3 on Recursive Algorithms in Curricula '91 lists a lecture topic connecting recursion to mathematical induction. The corresponding knowledge unit (PF 5) in Curriculum 2001 omits all mention of induction.

While the task force has not yet attempted to add prerequisites to the knowledge units of Curriculum 2001, it appears that very few will have mathematical prerequisites. Our best estimate is AL1, AL3, AL5, GR1, IS3, SE4 (on software validation), and perhaps SE2 are the only KU's which will need mathematical prerequisites. An indication that we are not far off in our estimates is that one of the unpublished proof-of-concept curriculum designs using the new knowledge units had no mathematical prerequisites until the 4th semester course.

So far we have not mentioned the mathematics requirements of Curriculum 2001. We have not done this because there is a huge disconnect between the mathematics requirements and the actual use of mathematics in the core.

The new curriculum specifies discrete math (DM) core knowledge units comprising 40 hours of lecture. The set of topics described seems to be quite good. The strawman report also requires a probability and statistics course and one other mathematics course, though the pedagogy focus group on supporting courses has recommended a two semester discrete math sequence which folds in some probability and statistics rather than having a separate probability and statistics course.

Whichever way this comes out, there will be very solid mathematics courses required for computer science majors. However, based on the core curriculum proposed at this point, students will have little opportunity to use this mathematics. This will simply serve to confirm the earlier reports from practitioners that mathematics has limited relevance to their lives.

However one feels about the amount of theory that should be included in the core curriculum, the mismatch between mathematics requirements and their use in the curriculum is clearly a problem that needs to be addressed. Our preference is to increase the amount of theoretical and mathematically-based material.

This issue of balance between theory and practice was reviewed extensively by a group of 18 computer scientists from selective colleges, called the Liberal Arts Computer Science Consortium, or LACS for short (see www.lacs.edu for more information). This group has developed and studied curriculum for several years, and is responsible for the "Revised Model Curriculum for a Liberal Arts Degree in Computer Science" [6].

In a letter to the Curricula 2001 Committee [3], the LACS group expressed these same concerns about the diminishing amount of theory and the lack of integration between theory and practice in the core curriculum. This letter made some concrete recommendations that would restore this balance. After careful consideration, the Committee decided not to accept these recommendations, for the following stated reasons [1]:

- The centrality of theoretical areas (algorithms and programming language theory) is declining relative to that of many other areas in computer science (net-centric computing, graphics, human computer interaction, information management, software engineering, and social and professional issues).
- The curriculum model must serve the widest variety of institutions.
- The curriculum model must serve the interests of the present and the future, rather than retreating to a "computer science of the past."

We are dismayed that the Committee views its role so narrowly. That is, the Report seems to favor only a collection of computational artifacts that happen to be prominent in the year 2001, represented in the Report in proportion to their perceived importance to the practice of computing. Thus, it appears that we have become less willing as an educational community to ask questions like "What are the principles upon which the discipline is grounded?" Or, "Is the decline of theory in software engineering a good thing?" Or finally, "Is computer science only about the practice of computing?"

## 4 Can We Do Better?

The recent decline of theory and mathematics in the computer science curriculum characterizes our discipline as (one of) the least mathematical among the science and engineering disciplines. Pressure to reduce the mathematical and theoretical content of our courses surely comes from many directions – our students, our technology industries, our deans, and even ourselves.

What can be done about this? We believe that several steps can be taken to reverse this trend and strengthen our curriculum.

1. Develop a curriculum report that reflects a more balanced treatment of theory and practice.

2. Show more convincingly how (old and new) theoretical material can be well-integrated among the other topics in the core curriculum.

3. Develop a dialogue among educators, subject area experts, and textbook writers that begins to explore how specific core courses can be moved onto sound theoretical ground that is supported by strong practical examples.

The first suggestion above is illustrated in the Revised Model Curriculum mentioned above [6]. Implementations of the specific core courses defined in that model, each of which integrates a significant body of theory with practice, can be developed, class-tested, and evaluated by willing faculty members.

The second suggestion above is illustrated by many examples, both traditional and contemporary. For instance, an operating systems course could include the development of a mathematical proof that a system is secure (i.e., prove that rogue applets cannot delete the file system). Or a software engineering course could present the idea that a program can include a proof that it is safe (e.g., that array subscripts won't exceed their bounds.) We would invite experts in each different subject area of computing to submit papers to SIGCSE that illustrate how theory and mathematics can be better integrated within a course in that area.

We believe that a reasonable integration of theory with practice does not signal a retreat to the "computer science of the past," as claimed by the Curriculum 2001 committee. To the contrary, we believe that it is the only way to guarantee a sustainable and defensible approach to studying the discipline of computer science.

## 5 Conclusions

We have argued that the undergraduate computer science curriculum has become math-phobic, the sense that mathematical and theoretical topics have gradually disappeared from its introductory and core courses. Evidence of this trend comes from several sources: computer science educators, data structures textbooks, and the evolving Computing Curricula 2001 report.

As a result, the vast majority of computer science graduates do not have well-developed ideas about the interconnections between theory and practice. Our discipline is thus presented as less like a science and more like a collection of techniques and artifacts that reflect current technologies. This strategy may prepare graduates for today's technology, but it will likely not prepare them well for the longer haul.

We have suggested some avenues that can be explored to reverse the trend of our curriculum toward mathphobia. We invite colleagues from all areas of computer science who are similarly inclined to contribute their own ideas about steps that can be taken to reverse this trend and help restore mathematical cohesion to the undergraduate computer science curriculum.

## References

[1] Chang, C., Engel, G., Roberts, E., and Shackelford, R. Letter to the Liberal Arts Computer Science Consortium. *e-mail correspondence on behalf of the CC2001 Steering Committee* (August 18, 2000).

[2] Kelemen, C., Tucker, A., Henderson, P., Bruce, K., and Astrachan, O. Has our curriculum become math-phobic? (an American perspective). *Proceedings of ITiCSE2000 1* (2000), 132–135.

[3] LACS. Letter to the cc2001 steering committee. *e-mail correspondence from the Liberal Arts Computer Science Consortium* (July 24, 2000).

[4] Lethbridge, T. Priorities for the education and training of software engineers. *Journal of Systems and Software 53, 1* (July, 2000).

[5] TWM. Data structures market adoptions by school. *TWM Research, Analysis, and Consulting* (2000), 24 pages.

[6] Walker, H., and Schneider, M. A revised model curriculum for a liberal arts degree in computer science. *Communications of the ACM 39, 12* (December, 1996), 85–95.