COMPUTER SCIENCE STUDENTS NEED ADEQUATE MATHEMATICAL BACKGROUND

Hasmik GHARIBYAN PAULSON
Computer Science Department,
California Polytechnic State University,
1 Grand Avenue, San Luis Obispo,
CA 93407, USA
e-mail: hpaulson@calpoly.edu

ABSTRACT

Computer science is a relatively young discipline. The first computer science instructors were mathematicians and the first CS curricula were just modifications of mathematics curricula. As computer science has grown and matured, and some of its fields became independent disciplines of their own, the role of mathematics in it has faded significantly. The number of mathematical courses in the CS curriculum naturally has been decreased. The amount of mathematics required for a computer science majors has become dangerously small - so small, that it is starting to jeopardize student's ability to learn, understand and appreciate fundamental theory of computer science. This paper is devoted to studying the influence of computer science students' insufficient mathematical background on their ability to learn, understand and appreciate theoretical courses of computer science, particularly the Theory of Computing.

KEY WORDS: Mathematics in Computer Science, Computer Science Education.

1. Introduction

Throughout the history of computer science education there has been debate on what should be the appropriate mathematics background for CS majors. During the last decade this topic became more actual, since with the development of computer science and software engineering there has been a pressure to make the CS curriculum less mathematical. As a matter of fact the CS curriculum did become less mathematical. A number of studies have been carried out and surveys have been published during the last few years in favor of less mathematically rigorous CS curriculum [1-4]. Timothy Lethbridge, for example, surveyed a number of software developers on the importance of the knowledge they obtained at university in their jobs [5]. Mathematical knowledge ranked very low in this survey. The results lead Lethbridge to the conclusion that "Relatively little mathematics turns out to be important for software engineers in practice." Another argument in favor of cutting down the number of mathematical courses in the CS curriculum is to encourage more students to enter this field, since mathematics is one discipline that scares away many students interested in joining the information technology workforce [1,2]. Also, there is another viewpoint about why mathematics should be weighed less in the CS curriculum – it doesn't deny the importance of mathematics, but emphasizes, that if we were to include all the math that is useful for a CS major, it would result in a specialized math major, leaving us short on computer science itself.

The pressure for a less mathematical CS curriculum has alarmed a big army of computer science educators. A huge number of papers and articles have raised serious concerns regarding the role of mathematics in computer science and software engineering. A number of studies and surveys have been carried out to show that mathematics and mathematical thinking are central to computer science education [3,6-9]. The basic argument is the following: mathematics is a mindset that fundamentally improves one's ability to devise and implement algorithms. Mathematics is used to model the problem domain, to specify and design high quality software, develop correct and efficient algorithms. The main benefit CS professionals get from mathematics they learn at the university is the experience of rigorous reasoning with purely abstract objects and structures. "It is not what was taught in the mathematics class that was important; it's the fact that it was mathematical," states Keith Devlin [10].

The goal of this paper is not to repeat the importance of the mathematical knowledge for a CS major in a long run, but to emphasize its necessity while at school, when the student is taking certain required courses, namely, theoretical courses of computer science. As mentioned in Computing Curricula 1991 and repeated in Computing Curricula 2001 [9], "Theory is one of the three primary foundations of computer science. It depends on mathematics for many of its definitions, axioms, theorems, and proof techniques. In addition, mathematics provides a language for working with ideas relevant to computer science, specific tools for analysis and verification, and a theoretical framework for understanding important computing ideas." It seems natural to expect that by the time students get to the theoretical courses, they have been received the adequate mathematical background that will allow them to handle these courses without any difficulty. But is this true in reality?

This paper is to show how mathematically unprepared today's students are for theoretical courses of computer science, particularly the "Theory of Computing". Numbers and facts introduced in this paper are based on a study carried out at California Polytechnic State University in San Luis Obispo – one of well-respected public universities in the USA.

2. Motivation for the study

I have been teaching at the university level for 19 years: first 15 years at Yerevan State University (YSU), Republic of Armenia (a former USSR republic) and 4 recent years at California Polytechnic State University (Cal Poly), San Luis Obispo, USA. I taught different courses of computer science in both universities – programming courses, as well as theoretical courses. Comparing students in two different universities (with two different CS curricula – mathematically charged CS curriculum of YSU and significantly less mathematical CS curriculum of Cal Poly), it was hard not to notice, that having approximately the same level in programming courses, students' performance in and attitude towards theoretical courses of computer science is dramatically different. YSU students don't have any distinguished feelings about theoretical courses of computer science. Students at Cal Poly, on the other hand, find theoretical courses very hard and have a certain fear, if not hostility, towards them. This caught my attention from the very beginning of my career at Cal Poly. I started to observe closely students' performance in a senior level required class Theory of Computing (holding the reputation of one of the especially tough courses), trying to find out what is causing the irritation and difficulty. Throughout six quarters I gathered data representing students' performance in this course (12 sections, 315 students total – 2 sections per quarter, around 25-30 students per section), analyzed test results, compared student's grades in this course with their grades in other courses, surveyed students, talked to different professors teaching theoretical courses and got their opinions which were in absolute agreement with my observations and conclusions. The outcomes of this study are presented below.

3. The real picture

Theory of Computing is a heavily math-flavored course. Textbooks are written in formal mathematical language – all concepts are defined formally, all results have mathematical proofs, all algorithms and techniques are presented with the help of formal mathematical notation. To be able to handle this course, one must have an adequate mathematical background – first of all be very comfortable with mathematical notation to be able to read and understand the text; be knowledgeable in set theory, graph theory, combinatorics; understand very well functions and relations, proof techniques etc.

The six-quarter study of students' performance in Theory of Computing gave a clear picture of what is going on and why students don't like this course. Below we'll show some of the interesting outcomes of this research. Since the results for different quarters of investigation came out very similar and consistent, we'll bring the results for one quarter only – Fall 2001 (two sections, 56 students all together).

Results of the study show that for the majority of students the grade in Theory of Computing is lower than their average grade in other courses. Even in comparison with the majority of other "difficult" courses, students' performance in Theory of Computing is noticeably less satisfactory. For example in Fall 2001, the grade in Theory of Computing of 60.9 % of students was at least by one letter grade lower than their grade in Data Structures course – a relatively hard course, containing both programming and theoretical elements. More precisely,

19.6% of students had a decrease of two letter grades,
41.3% of students had a decrease of one letter grade,
34.8% of students had maintained the same grade,
4.3% of students had an increase of one letter grade.

The consistency of this situation leads to a necessity to find out why are students having difficulty maintaining their average grade in Theory of Computing. Where exactly lies the problem? What qualities do students lack that keeps them from being successful in this course? Here are some major flaws that have repeatedly surfaced during the period of investigation:

1) Many students are uncomfortable with mathematical notation – mathematical text is very incomprehensible for them. As a result these students are unable to read and understand the textbook on their own. The teacher has no choice but to spend a great deal of lecture time on interpreting what the book says in a more "human" language, instead of using that time presenting interesting results, techniques, examples. Needless to say, that these students are unable to present (reproduce) information with the help of formal notation as well. In the anonymous survey conducted at the end of the quarter, 45.5% of students admitted that they cannot read and understand the textbook on their own; 60% of students confessed that if teacher doesn't explain, they will not be able to understand the concept looking at its definition; 76.36% of students said that they cannot understand on their own the full meaning of the results stated in theorems; 47.27% believed that even after understanding the theoretical material (with or without help), they will not be able to reproduce it on their own.

2) Many students do not know basic mathematical concepts. As a result these students are unable to understand the new concepts represented in the course. Consequently, they are incapable to understand the course material at least at the theoretical, abstract level. For example, in the quiz on languages 53.98% of students answered "yes" to the question "Can a string contain infinite number of elements?" (a string is defined to be a finite sequence of alphabet letters), 62.33% of students answered "no" to the question "Is $\varnothing$ a language over the given alphabet?" (a language is defined to be a subset of the set of all strings over the given alphabet). In the final exam, defining a Pushdown automaton as a mathematical system, 46% of students failed to specify the domain and 68% of students couldn't specify the range of the function (called transition function) representing the work of the machine.

3) Many students do not have an understanding of proof techniques. Consequently, these students are not able to use major results of Theory of Computing to prove certain characteristics of objects they are working with. For example, in the final exam students were required to prove that the given language is not regular. To do this, one needs to use the "proof by contradiction technique" and the property established in Pumping Lemma for regular languages (the two Pumping Lemmas are fundamental theorems in the Theory of Automata and Formal Languages). The results of the final exam show that 72% of students failed to do this assignment due to their vague understanding and incapability of using the "proof by contradiction" technique.

4) Many students cannot remember the names of concepts (old or new), definitions, important results. Consequently, these students are not able to express their thoughts correctly and precisely, cannot formulate clear questions, and lead a mature conversation on the topics of the course. For example, in the final exam 48% of students failed to name the three basic μ-recursive functions, and 60% of students couldn't list the three operations that are used to create new μ-recursive functions from the basic ones. 56% of students were not able to state the Church-Turing Thesis – one of the most fundamental hypotheses of the Theory of Computing. And the average grade of the class for the essay question in the final exam was only 50.68%.

It is worth mentioning that professors teaching other theoretical courses in the department strongly agreed with these conclusions, and reinforced them with facts from their own experience.

On the other hand it will be only fair to mention, that when the turn comes to practical issues such as creating an automaton to accept a language, constructing a grammar to generate a language, designing a Turing machine to perform the required job, applying an algorithm to, for example, transform the given nondeterministic finite automaton into a deterministic machine, these same students are impressively bright and inventive. As a matter of fact, students' grades in tests on practical material is significantly higher than their grades in theoretical tests: in the final exam the class average for the practical part (exercises on constructing abstract machines, grammars, applying algorithms) was 76.2%, while the class average for the theoretical part (short questions testing their understanding of theorems, concepts, their knowledge of definitions) was only 62.86%.

4. Conclusions

Our study shows that quarter after quarter students keep making the same, almost exclusively mathematical, mistakes – a vivid evidence of insufficient mathematical training. According to Cal Poly's CS curriculum, students are required to take two quarters of Calculus (Calculus I and II) and one quarter of Discrete Mathematics (Discrete Structures) in their first year of education. Later, in their third year, they are required to take a one-quarter course in Statistics and two quarters of math or statistics elective courses. Theory of Computing is a senior level course that students take in their fourth year of education, and normally, by the time they get to it, they have taken all abovementioned math courses already. Well, looking at the results of our study, it is quite obvious that the amount of math courses that students take does not build a steady mathematical background. This limits students' ability to learn, understand and appreciate theoretical aspects of computer science. Particularly, one quarter of Discrete Mathematics is not enough to master such topics as Set Theory, Graph Theory, and Proof Techniques, which are used in different theoretical courses of computer science. And considering also that Discrete Mathematics is required to take in the first year of education, it is quite natural for students to forget in couple years all the knowledge received in this course.

We, professors who teach theoretical courses in the Computer Science department at Cal Poly, strongly believe that our students need more mathematical training. Discussions with professors in different universities give the impression that this is not an unusual situation for other American universities as well. Of course it would be easy to suggest adding a few math courses to the CS curriculum and taking care of the problem, but how realistic that suggestion is. The truth is that CS curriculum is heavily loaded already, and it would be naive to assume that new courses can be added to it easily. In spite of this we need to take actions and find an acceptable and reasonable solution to the problem. Obviously it will take a lot of efforts and few compromises, but if we don't do anything about this now, it will be hard to justify the existence of the word "science" in the title of our discipline in the near future.

REFERENCES

[1] Glass R. L., " A new Answer to 'How Important is Mathematics to the Software Practitioner'?", IEEE Software, November/December 2000, pp.135-136.

[2] Gunstra N., "Universities aren't serving the IT workforce", Potomac Tech Journal, July 9, 2001, available at http://www.potomactechjournal.com

[3] Keleman C. and Tucker A.B. , ITiCSE audience survey, available at http://www.cs.geneseo.edu/~baldwin/maththinking/ITiCSE-survey.html

[4] Lethbridge T., Software Engineering Education Relevance survey, available at http://www.site.uottawa.ca/~tcl/edrel/

[5] Lethbridge T. , “What knowledge is Important to a Software Professional”, IEEE Computer, May 2000 (33:5), pp.44-50, available at http://www.site.uottawa.ca/~tcl/edrel/

[6] Henderson P., “ ‘Foundations of computer science 1’ Stony Brook Alumni Survey”, available at http://www.sinc.sunysb.edu/cse113/survey/

[7] Roberts E., LeBlanc R., Shackelford R., Denning P., Srimani P., Gross J., Curriculum 2001: Interim report from the ACM/IEE-CS Task Force. Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education, New Orleans, Louisiana, March 1999, pp. 343-344.

[8] Roberts E., Cover C., Chang C., Engel G., McGettrick A., Wolz U., “Computing Curricula 2001: How will it look for you?”, Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, North Carolina, Febrary 21-25, 2001, pp. 433-434.

[9] Year 2001 Model Curricula for Computing, available at http://www.acm.org/sigcse/cc2001/

[10] Devlin K., “The real reason why software engineers need math”, Communications of the ACM, October 2001/Vol.44, No.10, pp.21-22.