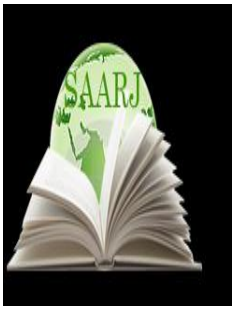
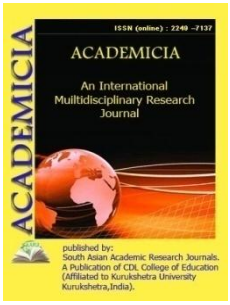


ACADEMICIA
An International Multidisciplinary Research Journal
(Double Blind Refereed & Peer Reviewed Journal)



DOI: 10.5958/2249-7137.2021.02191.1

MATHEMATICS: INTEGRAL PART IN COMPUTER SCIENCE FIELD

Dr Vipin Kumar Solanki*

*SOEIT, Sanskriti University,
Mathura, Uttar Pradesh, INDIA
Email id: hodmaths@sanskriti.edu.in

ABSTRACT

Despite the many connections across disciplines, there is evidence that computer science practices need little or no mathematical knowledge. This disconnect between the practical and intellectual roles of mathematics in computer science results in an awkward position for mathematics in computer science curricula, which necessitates math courses that are poorly aligned with computer science needs and students who use a lot of math but very little computer science. As a result, computer science graduates are hesitant and unable to utilize mathematics on the job. Fortunately, modest local changes may have an instant impact on the problem's major players. Although everyday practice needs little, if any, mathematics, computer sciences, software engineering, and mathematics are nevertheless linked, according to this article. Our primary concern is the education of computer professionals, the majority of whom are still educated via a curriculum that calls itself "informatics." Rather of referring to a single subject, the term "informatically science" is commonly used. Both software and computer science are addressed in our rationale and results. This research will help engineering students concentrate on the most essential topics in the curriculum.

KEYWORDS: *Computer science, Discrete mathematics, Mathematics, Science, University.*

1. INTRODUCTION

In general, science and engineering are intimately linked to mathematics. Natural science makes mathematical models of the phenomena they investigate; natural sciences and social sciences rely on statistics to tease raw data meaning; in all phases of systems design, building and maintenance, engineers depend on mathematical models. Computer science and software engineering seems to be a few of exceptions to this norm. The usage of mathematics is low in practising software developers[1]. Yet it would be surprising if relationships were as loose as it

looks between computer science, software engineering and mathematics. In the worst possible case, it would be rather harmful for disciplines to reject techniques that characterise areas with the titles they use. The only non-maths of the S&E family would be computer science and software Engineering[2].

In their working life, computer scientists utilise math in many ways. Firstly, mathematics offers the theoretical foundation for many computer sciences subfields, and for others key analytical tools; computer scientists therefore apply specific mathematical themes to specific computer issues[3].More generally, mathematics gives a framework to discuss computers and computing issues and provides a mental discipline to solve these problems more widely.In view of the condition of each participant, the student's identities as computer scientists were linked to their mathematical viewpoints. This study concentrated on a few student histories. This study also showed that students' perceptions of maths and their study programmes, each affecting the other, are interwoven.

Understanding how mathematics training may affect the views and identity of a student, in particular at an early age in CS, is vitally linked with an important problem present in the field of computer education: diversity. Outline of the current ACM computer science bachelor's programme instruction is broadly based on its mathematical needs, and many CS programmes are deliberately constructed early in the course of the study with a mathematical component. The link between these two disciplines must be understood in order to grasp the significance of mathematics for students who study computer science (CS) and software engineering (SE).This essentially minimises the difference between science and engineering[4]. The distinction between chemical engineering and chemistry or between physics and electrical or engineering is understood by many. One is science, which primarily advances disciplinary knowledge, and the other, which consists largely of applying this information to the achievement of humanity's technical requirements.

In their first mathematical course, the students are generally exposed to the principle of mathematical induction. Recursion and iteration are key philosophical and informatics principles for implementation. However, the connections between mathematical induction, recursion and iteration are not well understood by many graduates. Most basic CS textbooks and data structures / algorithms literature give a minimum of induction math since the classes are not usually preconditioned by discrete mathematics[5]. Some curricula require mathematics as the basis for the course on the data structures to establish these important connections.

In order to succeed, the teachers of these two courses must coordinate carefully and deliberately. The mathematics faculty, who may not be well aware of the linkages and/or could feel that discrete mathematics is a difficult subject, which requires many preconditioned classes, usually teaches CS/SE students' Mathematics courses. This creates an even larger gulf for CS/SE students who are already a little mathematical phobic. The Mathematical Association of America (MAA) acknowledged these problems and dealt with them in its 2004 guidelines for curricula[6].

The qualities of a graduate in computer science could hardly, as taught today, be distinguished from those of a graduate in software. The latter might have some additional software engineering courses; both are, however, mostly trained in programming entry-level roles. It is not really science or engineering, actually.

ABET, an accrediting board for engineering and technology, has been established in the United States to certify the undergraduate programmes in computer science and software engineering. Below is the definition of engineering by the US Engineering and Technology Accreditation Board[7]: The following: "Engineering is a profession where math and natural science is used with judicial judgement and learnt the materials and strengths of nature in the interest of a cost-effective method".

Typical curriculum for education in computer science do not cover discrete mathematics and software. University programmes generally need discrete math and software engineering courses, but they are often given as an alternative if the place they merge—courses in the field commonly termed formal methods. Standard discrete math courses offer minimum motivation and application of materials.

The conventional courses in Software Engineering do only use a little if any discrete mathematics, and the formal courses in computer science normally are optional and late.These factors lead to pupils having a minimal education in current software theory and practise based on mathematics. In addition, pupils are not given an opportunity to evaluate the usefulness and value of this content and to ask, literally, why they should attend discreet maths classes. A rich background is the discreet mathematical teaching.

This definition is not very well met by computer engineering. What is "natural science" or "natural materials and forces?"it is believed that think computer science, a man-made discipline, is the foundation of software engineering, that "materials" are mostly physical rather than conceptual and that "natural forces" are truly "universe laws." The alternative definition could therefore be as the profession of engineering in which knowledge of fundamental mathematics and sciences, acquired through study, experience and practise, are applied to the development of ways to use the materials, concepts and laws of the universe economically in the interest of the human race.

Fundamental mathematics and science" appear to represent this connection and dependence better and more generally. The following graphic containing basic mathematics as discrete Mathematics, including Logic, may be shown for software engineering. The approach is being developed largely by the Bachelor of Informatics as preparation for the graduate studies in advanced knowledge and as a professional track in software systems design/development[8]. Graduates of both programmes, while this would not be their major professional route, should be skilled programmers. Programmers are essentially technicians, and experts who are able to achieve their objectives without a thorough grasp of the basic mathematics or physics behind them.

In earlier parts of this article, the validity of a recursive or iterative pattern was argued in the context of a mathematical induction. The linkages are seen to be essential to the motivation of students may, and must be established in the first year. Mathematical induction is widely acknowledged a challenging topic for pupils. This was examined by mathematical educational scientists[9].

In the first discrete mathematics course, the classical technique to the teaching of mathematical induction is to use multiple numerical examples and problems as a modular unit.For CS/SE major induction, other courses generally touch on mathematical induction (e.g., algorithm

analysis, theory of computation, etc). This is considered insufficient for students to comprehend and utilise induction and to establish the required relationships with CS.A person with knowledge of HOW will always have a job, but the individual with knowledge of HOW will always be his boss[5].

Training vs education! Training versus education! Most students try rather than to grasp the basis for the information they have to obtain a career (HOW) as well as (WHY). This is the idea of education "to fill a vessel, "those which is also a component of the software-practitioner knowledge survey. The driving factor behind many professions' curriculum growth is frequently knowledge rather than understanding, mostly owing to the demand of companies, managers and students. Knowledge-based courses are also more easily taught and evaluated by students.

1.1 Who Should Teach Mathematics?

This is a question more significant than the substance of the course. Most CS/SE departments can do discrete work just as much as most physical departments can teach calculus. CS/SE departments are professors. (This would not be pushed too far into analogies, as physicists frequently know the topic of the calculus enough well to teach it, but few have a broad mathematical perspective or background to accomplish this properly. On the other side, CS/SE faculty members frequently have the necessary perspective and background, although this may not be true much longer.)

Anyway, CS/SE professors have as many teaching courses directly inside the subject as their colleagues in physics without strictly teaching extra mathematical courses. In general, physicists are quite pleased to teach their prospective pupils' mathematics, especially since, at least in the past, they swung enough weight to ensure that calculus courses covered the necessary materials[10]. Thus, CS/SE may almost dictate the curriculum for such courses, because they are by far the major customer departments for those courses.

In addition, the mathematical departments are certainly hungry enough to be willing to teach the courses at least in theory. For their design and operation, the modern systems depend heavily on software. Specification, design and execution of reliable software with rigorous development procedures must be simple for the next generation of developers. To assist prepare this generation, we designed a teaching strategy and two-fold-focused materials: to increase the knowledge and enjoyment of discrete mathematical structures (DM) that underlie software engineering theory (SE).

However, it was true that a decade ago, very few mathematics faculty departments were sufficiently aware of computer science and that they had to give them a decent mathematical course for students who were going to participate in CS or SE programs. Does that continue to be true? (It is usually accurate, with certain quality institutions of tiny liberal arts being possibly the primary exceptions.)

In any events, CS/SE programmes should certainly strive for their mathematics courses in maths. It is a long-term objective of the CS/SE programmes. If your Department of Mathematics is still unable to provide you excellent enough work, it is certainly advantageous to urge your Mathematics Department to employ discrete mathematicians to accomplish that.

1.2 Arguments Favouring Mathematics in CS/SE Curricula

Some practitioners are going to need some arithmetic. However, not a single specialised math course in CS/SE curriculum can reasonably be justified by either the number of mathematicians or the quantity of mathematics they will need. However, excellent reasons exist for incorporating a CS/SE curriculum in some mathematics classes. Here is a couple here.

- Its impact on the mind is the important yet unproven reason for mathematics instruction at any school level. In other words, study mathematics improves pupils' learning skills. Mathematics is very crucial for students of CS/SE as the logical thought contained in all mathematical ideas is so close to the logical idea necessary in all the construction of software.
- Some, but not many of the CS/SE graduates will pursue professions where math is required. This number can rise as and when developers of software are usually more formal than they are today so that they begin to be utilised more extensively, for example in The Science of Programming.
- A few CS/SE graduates will attend a graduate school in CS, although very few. Some of them, maybe far more mathematically based on what they learn than anything else in CS/SE undergraduate curricula.

1.3 Mathematics and Reasoning

Many computer tasks need practitioners to rationally and thoroughly analyse issues and their solutions – frequently using mathematical tools and methodologies. For instance,

- Whenever issues or solutions are offered, users should question what assumptions and how they might affect whatever results they have gotten or programme behaviour.
- When a problem solution is suggested to an algorithm, developers and researchers must assess if the method is proper and efficient in using resources.
- When software is proposed as algorithm implementation, testing organisations and users can verify that the software complies with the requirements stated both formally and experimentally. (The formal verification necessary exists in instances.
- The requirement to be fair in mathematical terms has been established by electronic gambling devices in some jurisdictions[1]. A position announcement by an amateur gaming firm was recently received from one author seeking "to create, test and analyse new games," and "to compose mathematical evidence for game submissions to regulators".)
- If multiple possible solutions to a problem are presented, practitioners should under different assumptions can assess the relative advantages and drawbacks of these options.

In this issue, many of the following topics are included in my article entitled "Mathematical Reasoning in Software Engineering Education." They are presented with further reasons for their completeness. They are thorough, yet they understand that there are many viewpoints on these problems.

- Abstract software.
- Mathematics and Software Engineering are commonly associated with notations, symbols, abstractions and accuracy.
- For modelling and software system behaviour, mathematics is crucial.
- Mathematically based are many fields of application (engineering, science, economics, etc).
- For most issue solving, mathematical reasoning is crucial, notably the construction of software systems.

1.3.1 Software is abstract:

The fundamental form of mathematics was one of the earliest abstractions our ancestors devised. Since the beginning, mathematics is now the fundamental instrument for humans. For non-physical objects, such as software, abstract reasoning requires no greater tool than mathematics. A software system may thus be seen as a model for a desired process or calculation that is mathematically accurate. Software experts largely agree on the abstract nature of software, but appear to prefer to use alternate, non-mathematical methods to describe, design, build, test, debug and manage software systems. This will progressively change, as systems get bigger and more complicated, mathematical instruments (e.g. modelling and model checks) become accessible, and graduates are more mathematically knowledgeable and more aware of the mathematical potential as a reasoning tool.

1.3.2 Notes, symbols, abstracts, precise

All four of them rely largely on the software. For ordinary objects and concepts, notations and symbols are abstractions. This is why y = axe + b is known from algebra and the programming count = 0. In their context of use, both are widely understood and accurate. Students are encouraged to master programming language notations, symbols, accurate syntax and semantics. Actually, this is none other than mathematics, which is generally easier to understand. Students, however, regard mathematics as static and rotary. Programming is seen as dynamic and exciting, attractive to our operating brains.Other computer languages and tools deliver comparable delights when instructors find and embrace them. These include standard ML, Miranda and Haskel languages, language design and testing, and languages like Maple, Mathematica or Axiom programming mathematics. Their usage is wide and is suitable for pupils who acquire a declarative thinking style. They employ notes, symbols, abstractions and accuracy.

1.3.3 Software modelling systems:

Before any item begins to be built, a model, even a conceptual one, must be developed. Today software development is more of an art in which the initial concept takes shape slowly – like melting a piece of clay. However, for projects in which more accurate understanding of the desired item is necessary before building, such adhoc techniques are not acceptable. The first solution is to construct, analyse and test a "mathematical" model. New tools, languages and approaches for modelling software are changing whose use will once become the standard. Languages are now available for system specification.

1.3.4 Domains of application:

Mathematics is a rich global, inclusive language for communication across different communities. Accordingly, the programme offers software practitioners a tool for efficient communication of mathematical underpinnings to customers and associates of all disciplines (ingenriers, scientists, mathematicians, statistics, actuarials, and economics).

2. DISCUSSION

The understanding of details (i.e. jargon, true tables, formal rules of logic) and understandings (i.e. paraphrasing formal rules), and rudimentary applications usually start at the beginning of Bloom's taxonomical processes inside a learning environment. Computer science is no exception, whether or not in its mathematical elements. This is the basis for thinking about Algorithms, programmes, systems etc. This basic work is necessary. This basic level of reasoning and comprehension, however, is insufficient to actually make practical use of computer science. Students need to understand more than the routine mechanics. Such learning occurs when subsequent courses are based on introductory courses and practise in both organised and open settings at a more profound level of analysis. Although this analytical analysis may not be included in every topic discussion in higher levels, students must regularly and in many different circumstances encounter it. If undergraduate programmes do not work with the mathematics that they demand, they restrict the capacity of graduates to apply mathematics in later studies or jobs.

There are (and have been) a number of methods to bridging the gap between the role of mathematics in informatics and the way it is taught in computer science undergraduate education. However, none of these indications shows a clear promise of success.

Mathematical requirements are applied more efficiently. Although most informatics curricula provide room for different mathematics courses, few actually teach mathematics, which is critical to the field. However, correcting this inefficiency is difficult. Under explicit or implicit pressure, prerequisite structures in mathematics departments may require computer science students to take basic, but not directly applicable, programs in engineering schools and computer programs to include maths that are traditional to physical sciences, even if they are not critical to computer science.

Mathematics may be included in computer science courses, and vice versa. The primary issue, which has been addressed in a number of ways, is the underuse of mathematics in the computer science curriculum. Henderson taught a first course for IT majors in the 1980s and 1990s that focused on the foundations of informatics research in terms of mathematical reasoning and problem solving. In the early 1990s, further efforts to integrate discrete concepts with mathematical disciplines, such as Foundations of Computer Science, were part of the curriculum. In the late 1990s and early 2000s, Baldwin and Scragg developed a course that introduced many of the discrete mathematical skills required by computer scientists in elementary algorithm design and analysis; today, curricula of software engineering that emphasize mathematical technologies to produce accurate programs are used. However, none has gained momentum beyond its creators. They have been seen as interesting, and perhaps even commendable, by the computer science community, but not as paradigm-shifting innovations that must be adopted.

3. CONCLUSION

The computer science is frequently used to simulate the phenomena it examines, like conventional areas of engineering. In addition, computational and mathematical thinking are intimately linked. Paradoxically, though, many graduates of computer science and software engineering perform effectively as professionals without mathematics explicitly being applied to their job. This contradiction leads mathematics to be discomfort able in IT courses: while the majority of these programmes include suitable mathematics, they frequently include a lot of non-computer-based mathematics, yet they often ignore other applications when teaching certain mathematical applications. This odd approach of computer science mathematics has surprisingly long remained resistant to revision. Although the exact causes varies each school, we feel that the overall explanation is because the faculty of computing just does not consider the issue as a matter of urgency. In addition, in fact the problem appears to be small, as long as graduates of computer science find work or employment in graduate schools in the subject and the area itself grows. However, there is cause for alarm from a longer perspective. Slowly mathematical instruments and procedures are adopted by software development and testing and today's graduates must be adapted over their careers to such tools and processes. As graduates go through their professions, they become responsible for system design, assessment of test findings or quality metrics, architectural choices or algorithms and similar actions requiring quantitative examination of data and comparison of possibilities.

REFERENCES

1. T. C. Lethbridge, "Priorities for the education and training of software engineers," *J. Syst. Softw.*, 2000, doi: 10.1016/S0164-1212(00)00009-1.

2. S. Surakka, "What subjects and skills are important for software developers?," *Commun. ACM*, 2007, doi: 10.1145/1188913.1188920.

3. J. A. Laub, "Assessing the servant organization; Development of the Organizational Leadership Assessment (OLA) model. Dissertation Abstracts International," *Procedia - Soc. Behav. Sci.*, 1999.

4. D. L. Parnas, "Software engineering programmes are not computer science programmes," *Ann. Softw. Eng.*, 1998, doi: 10.1023/a:1018949113292.

5. A. B. Tucker, C. F. Kelemen, and K. B. Bruce, "Our curriculum has become math-phobic!," in *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, 2001, doi: 10.1145/366413.364593.

6. W. Barker and S. L. Ganter, "Fundamental mathematics: Voices of the partner disciplines," in *A Fresh Start for Collegiate Mathematics: Rethinking the Courses Below Calculus*, 2006.

7. D. Thandapani, K. Gopalakrishnan, S. R. Devadasan, and R. Murugesh, "Implementation of European Quality Award in Engineering Educational Institutions via Accreditation Board for Engineering and Technology," *Int. J. Bus. Excell.*, 2013, doi: 10.1504/IJBEX.2013.050576.

8. "Session details: Why universities require computer science students to take math," *Commun. ACM*, 2003, doi: 10.1145/3262533.

9. L. Cassel, A. Clements, and G. Davies, "Computer Science Curriculum 2008: An Interim Revision of CS 2001," 2008.

10. A. O. Bilska *et al.*, "A collection of tools for making automata theory and formal languages come alive," *SIGCSE Bull. (Association Comput. Mach. Spec. Interes. Gr. Comput. Sci. Educ.*, 1997, doi: 10.1145/268085.268089.