ПРЕПОДАВАНЕ НА МАТЕМАТИКА СЪС СОФТУЕРНОТО ИНЖЕНЕРСТВО СЪВКУПНОСТ ОТ ЗНАНИЯ [SWEBOK]

TEACHING MATHEMATICS WITH THE SOFTWARE ENGINEERING BODY OF KNOWLEDGE [SWEBOK]

Dr. T. V. Gopal
Professor
Department of Computer Science and Engineering
College of Engineering
Anna University
Chennai - 600 025, INDIA
E-mail: gopal@annauniv.edu, gopal.tadepalli@gmail.com

Abstract
It is a fact that employability of the graduates and post-graduates in engineering disciplines does not explicitly require any specific mathematical knowledge. It is unfortunate but true that even though Mathematics is the foundation of computer science it is now being considered as a totally separate subject. The inclusion of topics from Mathematics in the Computing Curricula all over the world has often times been very tough to justify. Mathematical foundations any beyond the first year in a typical Undergraduate Engineering programme are rarely found in the computing curricula today. The author is associated with working on teaching computer science to K-12 schools. Integrating Computer Science and Mathematics has a huge potential but so are the risks. The focus is the IEEE Software Engineering Body of Knowledge [SWEBOK].

Keywords: Computer Science, Curricula, Mathematics, Integrating, IEEE SWEBOK

1. INTRODUCTION

The emergence of Computational Mathematics is the best possible bridge that blends the power of abstraction of mathematics with the implementation methods [1] for problem solving using the computers. The topics that make this bridge work are:

- Problem – Solving with Vectors and Matrices.
- Geometrical Coordinate Systems to Represent Linear Transformations.
- Sets, Relations, and Functions.
- Apply Logic to Statements.
- Discrete probability problems.
- Random Variables and Decision Problems.

These topics clearly indicate that, prima facie there is no need for teaching mathematics to Computer Science students any beyond the first year in the Undergraduate Engineering programme after they finish K-12 education. It sounds ironical that even the Numerical

Methods for Computer Science where the machines have been always dubbed as "Number Crunchers" also disappeared from the Computing Curricula.

One core example of the challenge is that numbers behave differently in most programming languages. Mathematics places no limit on how small or large a number can be. However, the programming languages frequently truncate values often times without any warning, leading to unpredictable results. Any 5th grader should know that 2 ÷ 4 equals ½… but in Java the teacher will have to explain why the same expression evaluates to zero.

Making matters worse, programming languages like Java, JavaScript, Python, Scratch and Alice all rely on the concept of assignment. Assignment means that a value is "stored in a box", and that the value in that box can be changed.

Taking recourse to the "Computational Mathematics", excellent library routines for number crunching were readily supported by highly successful programming languages for scientific computing. FORTRAN is a case in point. The systems began to enable linking these library routines with any other programming language. It is enough if one knows how to call the library routines from their computer programs written in any programming language. Many programming paradigms [2, 3, 4] disciplined the way programs are written through well established principles and implementation methods. Software began to blossom to a point that after Mathematics only Software spans across all disciplines of not only engineering but also human endeavour in general.

Computations entail finding an answer to a problem using mathematics or logic.

This cross-disciplinary approach [5, 6, 7] prepares professionals for careers in science, medicine, engineering and education. The success rate of Software Engineering based projects and products is just around 36% with at most 33% of the code fully tested out by the developers. The success of Software resulted in adding more and more courses in the computing curricula usually at the cost of pruning mathematics course. Verification and Validation of Software is a challenge even after 70 years of establishing its imperative need. Today Software not only impacts the "Quality of Life" but life itself. Safety – Critical systems demand the most elusive formal methods. Curriculum developers are once again looking up to the mathematical foundations with very limited justification.

Adding a new subject is easier said than done. Recruiting, training, hiring and retaining tens of thousands of teachers takes decades and costs billions. The finite number of hours in a given day and class rooms in typical colleges makes it difficult to find space for many likely candidate courses.

With the experience of being associate both with K-12 and University programmes at Under Graduate, Post Graduate and Research leading to Ph. D, the author embarked on the approach to teaching Software Engineering based on the IEEE Software Engineering Body of Knowledge way back in 2002. This work in now into the fourth edition. Many eminent professionals from all parts of the world are keenly associated with this work. This report summarizes the experiences of the author during the past two decades in teaching mathematics for software engineers.

In a nutshell:

Programming for Software Engineering = Art of Programming + Science of Programming +Discipline of Programming + Discipline of Software Engineering + Psychology of Programming.

2. EXHIBITION

Computer science is considered by some to have a much closer relationship with mathematics than many scientific disciplines. Early computer science was strongly influenced by the work of mathematicians such as Kurt Gödel and Alan Turing. There continues to be a useful interchange of ideas between the two fields in areas such as mathematical logic, category theory, domain theory, and algebra.

Edsger Dijkstra has argued that software engineering is a branch of mathematics.

The relationship between computer science and software engineering [8, 9] is a contentious issue. Some people believe that software engineering is a subset of computer science. Others, taking a cue from the relationship between other engineering and science disciplines, believe that the principal focus of computer science is studying the properties of computation in general, while the principal focus of software engineering is the design of specific computations to achieve practical goals, making them different disciplines.

The use of mathematics within software engineering is often called formal methods. However, computability theory shows that not everything useful about a program can be proven. Mathematics works best for small pieces of code and has difficulty scaling up.

In mature engineering fields routine design involves solving familiar problems and reusing large portions of prior solutions. Often these “solutions” are codified in the form of equations, analytical models, or prebuilt components. Software engineering is different from other engineering disciplines in some respects; perhaps the kind of science that underpins it is different as well. “Number Crunching” or Quantitative Processing using Computations and Software has been discussed earlier.

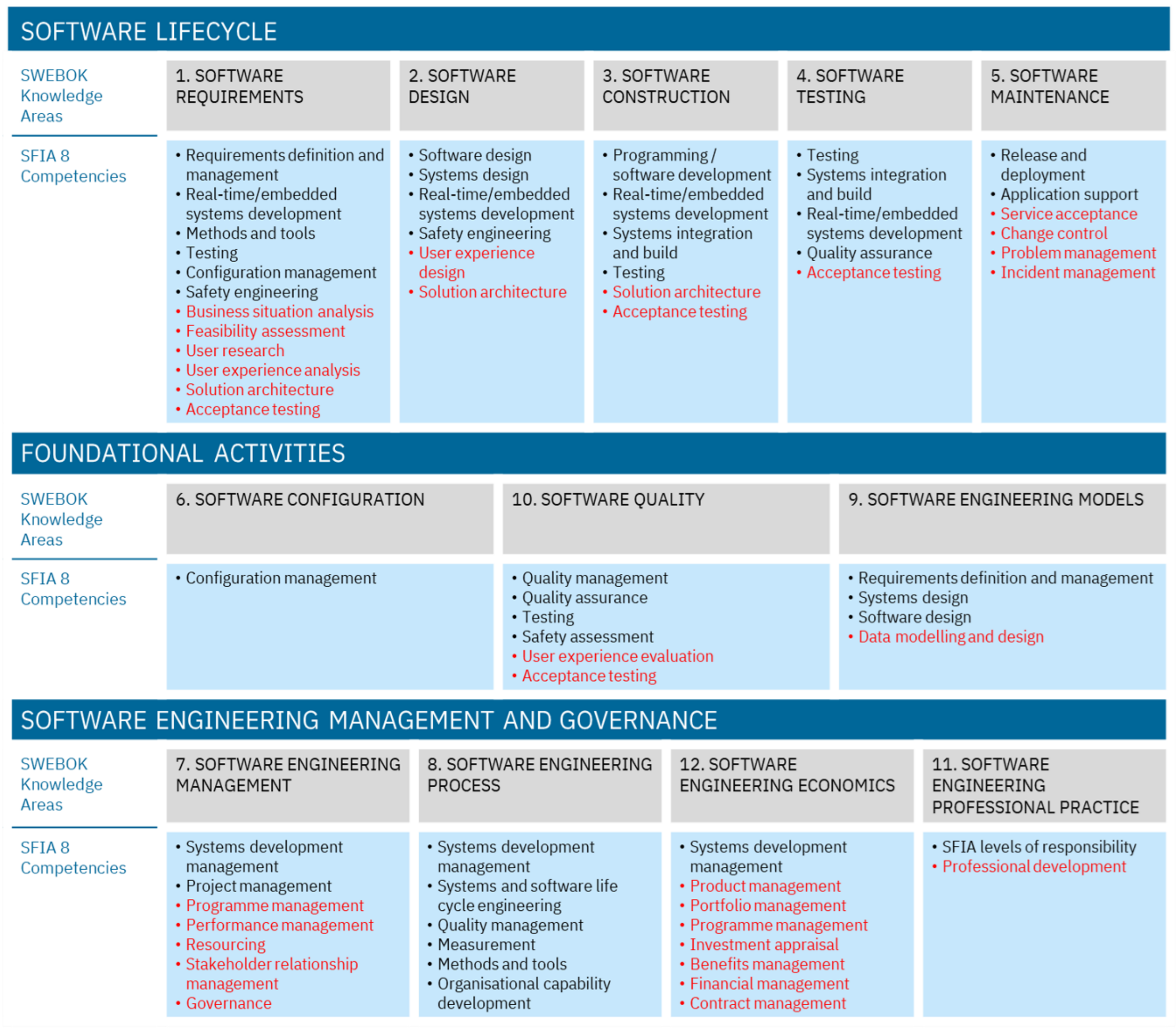
Many mathematical objects, such as sets of numbers and functions, exhibit internal structure. The structural properties of these objects are investigated in the study of groups, rings, fields and other abstract systems, which are themselves such objects. This is the field of abstract algebra.

The author is using Geometry for Cyber – Physical Systems. Trigonometry, Fractals and Topology are mathematical topics that enable the bounded regions for software testing.

"Change Impact Analysis" in Software Requirements, Development Life – Cycle and Process Models is formalized to a good extent with mathematical topics such as Calculus, Differential Equations, Dynamical Systems, Vector Calculus and Chaos Theory.

3. ANALYSIS AND RESULTS

Mathematical language also is hard for beginners. Words such as or and only have more precise meanings than in everyday speech. Mathematicians refer to this precision of language and logic as "rigor". Rigor is fundamentally a matter of mathematical proof. Mathematicians want their theorems to follow from axioms by means of systematic reasoning. Such a rigor is elusive in Software Engineering. However, a systematic integration of the mathematical foundations based on the IEEE SWEBOK is pragmatic as shown in the fig. 1 below.



SOFTWARE LIFECYCLE

SWEBOK Knowledge Areas	1. SOFTWARE REQUIREMENTS	2. SOFTWARE DESIGN	3. SOFTWARE CONSTRUCTION	4. SOFTWARE TESTING	5. SOFTWARE MAINTENANCE
SFIA 8 Competencies	• Requirements definition and management • Real-time/embedded systems development • Methods and tools • Testing • Configuration management • Safety engineering • Business situation analysis • Feasibility assessment • User research • User experience analysis • Solution architecture • Acceptance testing	• Software design • Systems design • Real-time/embedded systems development • Safety engineering • User experience design • Solution architecture	• Programming / software development • Real-time/embedded systems development • Systems integration and build • Testing • Solution architecture • Acceptance testing	• Testing • Systems integration and build • Real-time/embedded systems development • Quality assurance • Acceptance testing	• Release and deployment • Application support • Service acceptance • Change control • Problem management • Incident management

FOUNDATIONAL ACTIVITIES

SWEBOK Knowledge Areas	6. SOFTWARE CONFIGURATION	10. SOFTWARE QUALITY	9. SOFTWARE ENGINEERING MODELS
SFIA 8 Competencies	• Configuration management	• Quality management • Quality assurance • Testing • Safety assessment • User experience evaluation • Acceptance testing	• Requirements definition and management • Systems design • Software design • Data modelling and design

SOFTWARE ENGINEERING MANAGEMENT AND GOVERNANCE

SWEBOK Knowledge Areas	7. SOFTWARE ENGINEERING MANAGEMENT	8. SOFTWARE ENGINEERING PROCESS	12. SOFTWARE ENGINEERING ECONOMICS	11. SOFTWARE ENGINEERING PROFESSIONAL PRACTICE
SFIA 8 Competencies	• Systems development management • Project management • Programme management • Performance management • Resourcing • Stakeholder relationship management • Governance	• Systems development management • Systems and software life cycle engineering • Quality management • Measurement • Methods and tools • Organisational capability development	• Systems development management • Product management • Portfolio management • Programme management • Investment appraisal • Benefits management • Financial management • Contract management	• SFIA levels of responsibility • Professional development

Fig. 1. *IEEE SWEBOK Approach to Integrate Mathematical Foundations*

CONCLUSION

As complex software intensive systems evolve the need for mathematical basis increases. These systems are not only impacting the quality of life but also impacting the life forms. The author strongly advocates the mathematical foundations to be integrated with the IEEE Software Engineering Body of Knowledge.

REFERENCES

[1] D. Baldwin and P. B. Henderson. (2002) "The Importance of Mathematics to the Software Practitioner", *IEEE Software*, vol. 19, no. 2, pp. 112-111, March-April 2002.
[2] P. Bourque and R.E. Fairley (Eds). (2014). “Guide to the Software Engineering Body of Knowledge”, Version 3.0, *IEEE Computer Society*.
[3] D. Gries. (1981). “*The Science of Programming*”, Springer.
[4] D. Knuth. (2009). “*The Art of Computer Programming*”, Vol. 1 – 4A, Addison – Wesley Publishing Company, USA.
[5] E. W. Dijkstra. (1976). “*A Discipline of Programming*”, Pearson.
[6] E. Schanzer. (2017). “Integrating Computer Science in Math: The Potential Is Great, But So Are The Risks”, *American Mathematical Society Blog*, January 9.
[7] F. Neri. (2021). “Teaching Mathematics to Computer Scientists: Reflections and a Case Study”, *SN Computer Science*, Springer, Vol. 2, Iss. 2.
[8] G. M. Weinberg. (1971). “*The Psychology of Computer Programming*”, Van Nostrand Reinhold, New York.
[9] W. S. Humphrey. (1995). *“A Discipline for Software Engineering*”, Addison-Wesley Professional.