Elementary math to close the digital skills gap

Pia Niemelä[1] and Antti Valmari[2]
[1]*Tampere University of Technology, PO Box 527, FI-33101, Tampere, FINLAND*
pia.niemela@tut.fi

[2]*University of Jyväskylä, Jyväskylä, FINLAND*
antti.valmari@jyu.fi

Keywords: K-12 computer science education, computing in math syllabus, digital skills gap, professional development of software professionals, effectiveness of education, continuous vs. discrete math

Abstract: All-encompassing digitalization and the digital skills gap pressure the current school system to change. Accordingly, to 'digi-jump', the Finnish National Curriculum 2014 (FNC-2014) adds programming to K-12 math. However, we claim that the anticipated addition remains too vague and subtle. Instead, we should take into account education recommendations set by computer science organizations, such as ACM, and define clear learning targets for programming. Correspondingly, the whole math syllabus should be critically viewed in the light of these changes and the feedback collected from SW professionals and educators. These findings reveal an imbalance between supply and demand, i.e., what is over-taught versus under-taught, from the point of view of professional requirements. Critics claim an unnecessary surplus of calculus and differential equations, i.e., continuous mathematics. In contrast, the emphasis should shift more towards algorithms and data structures, flexibility in handling multiple data representations, logic; in summary – discrete mathematics.

1 INTRODUCTION

21st century society is digitizing rapidly and job descriptions of current professions are changing accordingly. Digitalization triggers pressure to change the current education system. Both domestic and multinational governing bodies have recognized the skills gap of computer science and the growing need for a digitally fluent workforce. Consequently, the EU has outlined a strategy for improving e-skills for the 21st century to foster competitiveness, growth, and jobs. Just-published technical reports provide guidance for educators and politicians at the European level (Redecker and Punie, 2017; Bocconi et al., 2016), highlighting the pervasive and ubiquitous nature of digitalization. Digital literacy, responsible use of technology, and civic participation are thus relevant to everybody. In consolidation, digitally skillful workers are more likely to keep their positions and, if displaced, are reemployed more quickly than employees without digital skills (Peng, 2017).

The skills gap concerns not only the number of SW professionals but also the quality of their skills. The STEM shortage paradox highlights the peculiarity of having hard-to-fill open positions and at the same time an excess of graduates who cannot find a job (Harris, 2014; Smith and White, 2017). One explanation is the skills mismatch, and in compliance with this, employers point out the candidates' incapability of breaking down problems into manageable chunks and solving them, and the gaps in technical, data modeling, and analytical skills. Accordingly, data base, data management, data analysis and statistics skills outnumber other requested digital skills of job advertisements in the US (Beblavỳ et al., 2016).

The discussion of the role of computer science (CS) in education is global. A number of countries all over the world have introduced CS into their K-12 curricula. In line with others, the FNC-2014 comprises algorithmic thinking and programming as parts of the mathematics syllabus (Finnish National Board of Education, 2014). In pursuit of consistent CS support, the entire math syllabus should be reviewed along with these newly introduced additions. This study then asks:

- RQ1: What elementary math syllabus areas should be strengthened for the anticipated CS emphasis?
- RQ2: Are there math syllabus areas that are currently overemphasized from this viewpoint?

First, this study reviews the discourse of CS as a scientific discipline and the learning targets of mathematics in anticipation of supporting CS. In the Related Work section, we list already-existing directives and recommendations of institutions that aim at building a flexible future work force, such as ACM. There, we focus on suggested math courses in particular. For comparison, we check the elementary-level math and computing syllabi of current strong performers in CS, i.e., the UK and US. The Results and Discussion section cross-exposes these recommendations with feedback from in-service software engineers by focusing on the evaluated profitability of the curriculum topics. To conclude, we propose hypothetical math learning trajectories for a CS support.

2 CS&SWE VS. ICT

Most natural sciences and engineering disciplines rely on calculus, differential equations, and linear algebra as a mathematical foundation appropriate for continuous phenomena. Systems relying on such phenomena can be adequately tested. For instance, a bridge does not need tests for all possible loads between zero and a maximum value. Testing the maximum load under typical and extreme weather conditions suffices.

In contrast, Parnas highlights the different nature of software (Parnas, 1985). Unlike bridge load tests, testing a piece of software with typical and extreme values does not guarantee expected behavior with untested values. Furthermore, software is rarely concise enough to be tested inside out, and unlike mathematical theorems, it is not comprehensively checked by other experts in the field. Thus, frequent errors and failures are common (Charette, 2005).

As we will discuss later, computer scientists have suggested topics such as logic, formal grammar, and set theory as an appropriate mathematical basis for mastering software and improving its quality. In addition, the importance of algorithmic thinking has been discussed extensively. In traditional engineering degree programs, classic mathematics and physics are included early on. The rationale is to develop a suitable mindset, that is, a way of thinking that facilitates a more profound learning of engineering topics. The basis is laid already in elementary school physics and mathematics. Similarly, professional computer science and software development need a suitable mindset that should be developed before studying the bulk of the software topics. However, because software cannot be appropriately mastered with tools suited for continuous phenomena, this mindset is not the same as that of, say, an electrical engineer.

The discussion of the educational needs in Finland suffers from a poor distinction between Information and Communication Technology (ICT), Computer Science (CS), and Software Engineering (SWE). For more than a decade, the Finnish mobile phone company Nokia was very successful and its educational needs had a remarkable impact on the Finnish educational discourse. In addition to SW engineers, Nokia needed expertise in the fields of hardware, radio technology, and signal processing. Therefore, ICT and SWE were emphasized instead of CS, with SWE largely perceived via analogy to traditional engineering, less through its relation to CS. As a consequence, Finnish scholars and educators have only partially conceived the special character of CS and SWE as disciplines distinct from ICT, thus requiring a different educational foundation, which implies changes in the math syllabus as well.

To clarify the conceptual difference, we define the relation of CS to SWE more closely. Parnas equates it to the relationship between physics and electrical engineering (Parnas, 1999, p. 21): physics belongs to the natural sciences, which target an understanding of a wide variety of phenomena; electrical engineering is an engineering discipline striving to create useful artefacts. Although electrical engineering is based on physics, it is neither a subfield nor an extension of it. Analogously, CS is a science, and SWE is an engineering discipline based on CS. Therefore, CS degrees must focus on the underlying computational phenomena and the acquisition of new knowledge of these, while SWE degrees concentrate on implementing trustworthy, human-friendly software cost-effectively.

In regard to math, the latest specifications of ACM&IEEE explicate the similarity of required skills both in CS and SWE (ACM&IEEE, 2013; Ardis et al., 2014). Even if CS is more scientific as a discipline and more deeply grounded in math, SW engineers benefit from more theoretically-oriented CS education and discrete math to be able to implement quality software. Hence, the conceptual difference does not diverge the required math and computing fundamentals. Consequently, Meziane and Vadera concluded, '*There is very little difference between the SE and CS programs currently offered in English Universities*' (Meziane and Vadera, 2004).

3 RELATED WORK

3.1 ACM recommendations

The standards developed by the Association for Computing Machinery (ACM) are used as a premise in curriculum planning in a number of Finnish universities. The CS concepts introduced in the first courses are important either for their own sake or for further topics. Obviously, the first fundamental concepts are also the most evident candidates when considering to advance some basics at the elementary school level.

3.1.1 CS Knowledge Areas of ACM

ACM promotes CS as a discipline and in compliance prepares normative recommendations for teaching CS at the tertiary level. ACM (ACM&IEEE, 2013) introduces Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (ACM-CS2013). The material is divided into Knowledge Areas (KA) and further to Knowledge Units (KU) that match with no particular course. Instead, courses may incorporate topics from multiple KAs. Topics are divided into Core and Elective, and the Core is further subdivided into Tier-1 (to be fully completed) and Tier-2 (at minimum 80% coverage). The KAs with the most Tier1 hours are:

1. Software Development Fundamentals (43 h)
2. Discrete Systems (37 h)
3. Algorithms and Complexity (19 h)
4. Systems Fundamentals (18 h)

The natural flow of concepts is to introduce software development fundamentals (SDF) and simultaneously strengthen the mathematical foundation with Discrete Systems (DS). In descending order of allocated hours, algorithms and complexity (AL) come next, where mastering common algorithms is considered general CS knowledge. Complexity considerations consist of evaluating the algorithm efficiency based on execution time and consumed resources. Systems Fundamentals (SF) give an insight into system infrastructure and low-level computing by acquainting students with computer architecture, main HW resources and memory, and, e.g., sequential and parallel execution.

From the list above, items 2 and 3 link closely with math. According to ACM, DS comprises the following areas in descending order of emphasis (Tier-1 + Tier-2 hours): Proof Techniques (11), Basic Logic (9), Discrete Probability (8), Basics of Counting (5), Sets, Relations, and Functions (4), and Graphs and Trees (4). AL in turn consists of basic and advanced KUs of Analysis, Strategies, Fundamental Data Structures, Automata, Computability, and Complexity.

In sum, algorithms and data structures are at the center of gravity together with the programming basics of SDF.

3.1.2 The most relevant math to support CS

ACM-CS2013 highlights the tight and mutual interdependence between math and CS. However, instead of being prepared for every kind of career option, ACM-CS2013 focuses on the common denominator. Thus, only directly relevant requirements are specified, such as elements of set theory, logic, and discrete probability comprising the KA of DS. On the other hand, ACM-CS2013 states that "*while we do not specify such requirements, we note that undergraduate CS students need enough mathematical maturity to have the basis on which to then build CS-specific mathematics*". It also mentions that "*some programs use calculus . . . as a method for helping develop such mathematical maturity*" (ACM&IEEE, 2013).

The recommendations make a distinction between such mathematics that is an important requirement for all students in the faculty and mathematics that is relevant only to specific areas within CS, exemplifying this with linear algebra that "*plays a critical role in some areas of computing such as graphics and the analysis of graph algorithms. However, linear algebra would not necessarily be a requirement for all areas of computing*" (ACM&IEEE, 2013).

If it were decided to emphasize discrete math including logic in the elementary school math curriculum, then an age-appropriate and tested subset of ACM Basic Logic could be found in the National Curriculum and GCSE Mathematics of the UK. The UK has already emphasized discrete math for a longer period, see section 3.3. Logic is deployed frequently in programming, not only when implementing conditions in selection and iteration statements. Subsequently, university-level logic targets more sophisticated and far-reaching knowledge than this. In consequence, Basic Logic of DS introduces normal forms, validity, inference rules, and quantification.

Although probability is linked more weakly to the programming fundamentals than logic, it gives readiness for various prominent topics, such as the analysis of average-case running times, randomized algorithms, cryptography, information theory, as well as games. Its basics should cover conditional probability, independent and dependent events, and multiplication and addition rules.

3.2 SWEBOK recommendations

The Guide to the Software Engineering Body of Knowledge (SWEBOK) of the IEEE breaks down the mathematical foundations into smaller knowledge areas (Bourque et al., 2014). In the review, we focus on both Chapters 13 and 14 of the guide, i.e., Computing and Mathematical Foundations.

Computing Foundation in Chapter 13 is included because it comprises algorithms and data structures. Data structures have various classifications, e.g., linear–nonlinear, homogeneous–heterogeneous, stateful–stateless. For instance, linear structures organize items in one dimension (lists, stacks), compared to the two or more hierarchies (trees, heaps) of non-linear structures. Well-designed data structures accelerate data storage and retrieval. The efficiency of algorithms depends significantly on the selection of a suitable data structure. Appropriate data structures can foster algorithm development. When the effects are combined, performance and memory consumption may range from poor to extremely efficient.

Chapter 14 highlights CS as an applied maths topic. The foundational KAs concentrate on logic and reasoning as the essences that a SW engineer in particular must internalize. The chapter describes mathematics as a tool of studying formal systems, widely interpreted as abstractions on diverse application domains. These abstractions are not restricted to numbers only, but include, e.g., symbols, images, and videos.

The following subtopics constitute the foundational KAs of math. Our assumption is that the order implicates their importance. We divide these topics into continuous (c) and discrete (d):

1. Sets, Relations, and Functions (c/d)
2. Basic Logic (d)
3. Proof Techniques (d)
4. Basics of Counting (d)
5. Graphs and Trees (d)
6. Discrete Probability (d)
7. Finite State Machines (d)
8. Grammars (d)
9. Numerical Precision, Accuracy, and Errors (c)
10. Number Theory (d)
11. Algebraic Structures (d)

One obvious observation is a notably smaller portion of continuous math compared to traditional engineering education. In particular, calculus, differential equations, and linear algebra are missing. Instead, several topics target a better position of underlying logic (2,3); and primers for data types, data structures and algorithms (1,4,5,9,11). In addition, subtopics of Basics of Counting (4), and Discrete Probability (6) and Number Theory (10) scaffold a deeper understanding of probability and cryptography. Numerical Precision, Accuracy, and Errors (9) section reveals underlying HW and memory specifics that have an effect on, e.g., the resolution of measurements and impossibility of expressing most real numbers precisely.

3.3 K-12 math and computing syllabi of the UK and US

For comparison, we went through the National Curriculum (UKNC) and General Certificate of Secondary Education (UKGCSE) of the UK (Department of Education, 2014; GCSE, 2015), and the Core Curriculum of US (USCC) (Core Standards Organization, 2015). The logic basics are present in the syllabi of both, with a comprehensive subset. Yet Boolean logic is currently included in the computing curriculum of UKNC, not in math. However, Boolean logic would fit well in the math syllabus as a consistent continuum of inequalities.

Sets can illustrate nested number sets of natural numbers ($\mathbb{N}$), integers ($\mathbb{Z}$), and reals ($\mathbb{R}$) that match with variable types (*unsigned*, *int*, *float*) in programming. However, due to differences in how, e.g., reals appear in both, we note that this juxtaposition is prone to misconceptions. For instance, in:

```
int x=1; float y=x/2;
```

division may produce a value of zero depending on the used language. All the same, not every *int* is a *float*, in contradiction of the math subset relation of $\mathbb{Z} \subset \mathbb{R}$. In addition to primitive types, sets are the basic mathematical abstraction of containment, and are thus relevant for programming as a cognitive tool. A group of numbers may be introduced as a set, a vector or a matrix, and the same group operations apply. Therefore, set theory would be useful in any mathematics curriculum designed to support programming. Currently, sets are a part of UKNC, but absent from USCC and FNC-2014.

Linear algebra basics are included in the USCC as matrices and basic operations; and as vectors and transformations in UKNC, whereas they are missing from the FNC-2014. For example, linear algebra basics could be a beneficial addition even if supported by ACM-CS2013 only as an elective math topic, because matrices are extensively exploited in the fields of statistics, data analysis, games, and graphics, for instance. The need for matrices is increasing, be-

Table 1: Math Syllabi (KS=key stage, G=grade, HS=high school. Each key stage covers several grades ranging from two to four. The GCSE exams follow KS4.

	UKNC	USCC
Logic	(in CS) KS2: logical reasoning to explain how simple algorithms work KS3: Boolean logic (AND/OR/NOT) and its application in circuits and programming	
Sets Prob	KS3: enumerate sets, unions/intersections, tables, grids and Venn diagrams KS4: data sets from empirical distributions, identifying clusters, peaks, gaps and symmetry, expected frequencies with two-way tables, tree and Venn diagrams	G6: data sets, identifying clusters, peaks, gaps, symmetry G7: random sampling to generate data sets HS: interpreting differences in shape, center and spread of a distribution
Vectors Matrices	KS4: (in Geometry) translations as 2D vectors, addition and subtraction of vectors, multiplication with a scalar, diagrammatic and column representations GCSE: transformations & vectors	HS: addition, subtraction, multiplication of matrices, multiplication with a scalar, identity matrix, transformations as 2x2 matrices

cause of topicality of their application areas and because many libraries in, e.g., Python exploit them extensively. As a topic, matrices and vectors belong together, and various transformations (such as scaling, translation, reflection and rotation) are main operations on image manipulation and animations.

Matrices are extensively exploited, e.g., in machine learning, data analysis, pattern recognition, and game engines for 2D/3D-transformations. All suggested math syllabus areas remain at the preliminary level in UKNC and USCC and we propose the same: in logic truth tables and Boolean logic in order to simplify several simultaneous conditions; in sets, Venn diagrams and basic operations of union, intersection and cut with at most three sets; and in matrices, transformations of translation, reflection, rotation and enlargement and finding an inverse matrix. This new math knowledge should be carefully bridged with the prior knowledge with lots of visual exercises and by starting early enough. Table 1 illustrates in which order these topics are handled in the UKNC and USCC.

Thus, in lieu of the ACM DS Logic subset, a readily field-tested elementary syllabus is found in GCSE CS (GCSE, 2015). It contains the following topics:

- binary and hexadecimal notations
- binary addition and shift
- Boolean values (*true*, *false*)
- Boolean operators (AND, OR, NOT); truth tables

Sets prompt types in programming and they can be utilized in abstracting both primitives and collections. UKNC specifies the syllabus of sets followingly:

- sets visualized by Venn diagrams
- set operations: subset, proper subset, intersect, and union, combinations of these
- sets represented as lists, and
- set and its complement

In addition, in the CS syllabus of the GCSE clear learning targets for algorithms are set: at a minimum, binary search and merge sort (GCSE, 2015).

4 Method

This study complies with the scope of curriculum theory (Pinar, 2012), and its key question of what knowledge is most valuable and how this knowledge is constructed as consistently as possible. Here, we are concerned with the educational and sociological aspects due to the aim of improved employability and filling the digital skills gap. This study is restricted to elementary math and compares the FNC-2014 to the UKNC and USCC (Department of Education, 2014; English Department for Education, 2013; Core Standards Organization, 2015) and to the recommendations given by the ACM and IEEE (ACM&IEEE, 2013; Bourque et al., 2014). The comparison exploits content analysis in searching for the math syllabus anticipated to be the most useful for CS students.

In addition to the comparison, the effectiveness of the university-level SWE studies reflects back to the curriculum design. We do not collect any new data but reuse the data of existing studies (Lethbridge, 2000; Puhakka and Ala-Mutka, 2009; Surakka, 2007; Kitchenham et al., 2005). The results of the previous studies are cross-correlated to confirm their validity in order to draw conclusions about the most profitable math topics.

5 Results and Discussion

In this section, we first review the feedback from the field: SW professionals evaluate the curriculum topics according to their profitability in working life. Having being informed of both the previous section's recommendations and criticisms of the current realization, we summarize the necessary math syllabus content and bridge the learning trajectories from elementary to higher-education math.

5.1 Feedback from SW engineers

To evaluate the effectiveness of their education, SW engineers have scored the profitability of a plenty of curriculum topics (Lethbridge, 2000). An imbalance between supply and demand was discovered and as a remedy, the author recommends putting less emphasis on the topics of minor importance – or teaching them in a way that makes them more relevant to SWE students. The study was run in year 1997 and repeated in 1998. The differences between outcome remained modest. In 1998, the sample size was $N = 181$, and the survey consisted of 75 topics of CS, SWE, etc.

A few years later, in 2004, Kitchenham & al. conducted a research focusing on the curricula and graduates of four UK universities (Kitchenham et al., 2005). The methodology was somewhat different and so was the obtained list of the most under-taught topics. The findings regarding mathematics were, however, the same. Then in 2009, a decade after Lethbridge's original research setup, Puhakka et al. published an analogous study conducted in Tampere University of Technology (Puhakka and Ala-Mutka, 2009, $N = 212$). Out of the original 75 subtopics, three were removed because of their not being common in Finnish curricula. Both sub-figures of Fig. 1 illustrate the differences between math-related perceptions among SW professionals in the examined cohorts of US and Finland. First, we observe that the results correlate surprisingly well, taking into account a timespan and continent switch. The scientifically significant values of R^2 are 0.88 in the upper, and 0.91 in the lower figure.

The green circles in sub-figures designate the areas considered either useful (the upper) or in need of more emphasis (the lower) to build work-life competences of SW professionals. The lower sub-figure, however, demonstrates the rarity of topics in need of more emphasis. Negative values indicate a post-graduate knowledge loss, whereas positive values a knowledge gain, in other words, inadequate learning of such topics in higher education.

The latter sub-figure is visually telling. Only algorithms and data structures are in need of more emphasis. In addition to these, the Lethbridge top-ten consists of no other mathematical but instead such items as negotiation, human-computer interaction, and leadership.

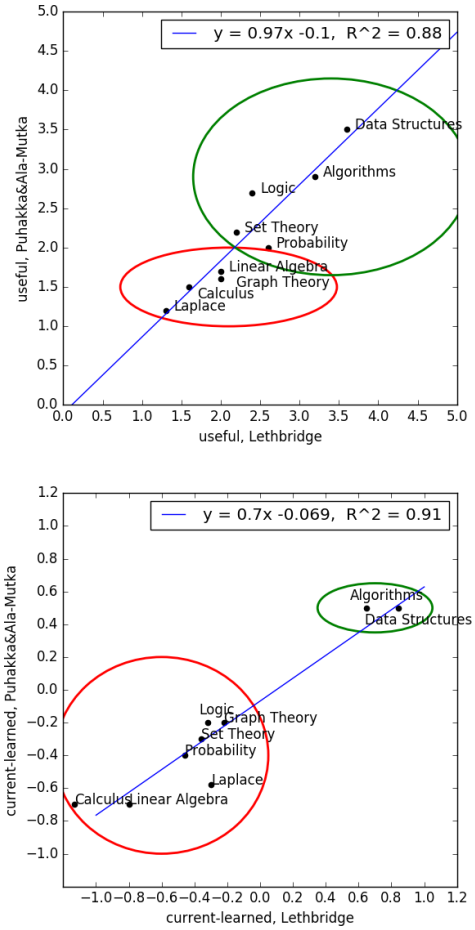

Figure 1: The comparison of usefulness and adequacy of math education evaluated by SW professionals (Lethbridge, 2000; Puhakka and Ala-Mutka, 2009, $N = 181$; $N = 212$)

In comparison with both previous surveys, Surakka separates the sample into the cohorts of SW engineers, academics (professors, lecturers) and students, see Fig. 2. The winner is again clear: algorithms and data structures, also the prominence of discrete math compared with continuous math is unchallenged, yet the bias has an academic flavour. Discrete math scores highest among professors and lecturers (3.1).

5.2 CS-supportive math for elementary

In constructing a strong basis for CS, both ACM and SWEBOK emphasize discrete math, confirmed by the feedback from the field. After programming basics,

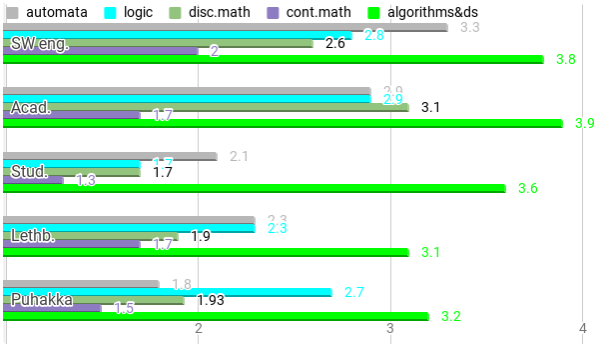

Figure 2: The math areas perceptions [1(not important), 4(very important)] of Surakka's engineers, academics, and students contrasted with Lethbridge and Puhakka et al.; $N = 11, 19, 24, 181, 212$; respectively

ACM values discrete systems as the second most, and algorithms, data structures, and complexity as the third most prominent KAs, whereas the in-service SW engineers value this area the highest. In SWEBOK, nine out of eleven math KAs comprise discrete math. UK, spearheading in CS, invests in discrete math already at the elementary level and in addition, provides CS as a separate subject with more in-depth topics.

Algorithmic thinking

The referenced studies categorize algorithms and data structures as part of the CS Core. In programming-oriented math, data structures can be seen as an application of set theory, e.g., sets conceptualize collections. In programming, collections are of various types: a set is an unordered collection of values, a list an ordered collection, and a map a collection of values identified by keys, which may also be interpreted as a representation of a mathematical function.

Denning equates algorithmic and computational thinking (Denning, 2009), which he in turn associates with general problem solving (Denning, 2017). When solving a problem, it is beneficial to start by decomposing it to smaller solvables implemented in a code as sub-routines, for instance. At its simplest, an algorithm may then be understood as a sub-routine, a sequence of commands called repeatedly as many times as desired, e.g., (CSTA, 2016). Computing is what Wing refers to as automation of abstractions, algorithms being the most prominent class of these abstractions (Wing, 2008).

The gradual division between human-completed calculation and computer-based computing has been the watershed between the disciplines of math and CS. In pondering the difference between the mindsets of mathematicians and computer scientists, Knuth points out that computer scientists need to be concerned about algorithms and their computing specifics, such as the notion of complexity or economy of operations. In most programming languages, the computing process comprises a series of sequential state changes executed assignment-by-assignment, which is an operation absent in math. Moreover, data structures in CS are inhomogeneous, which spreads the spectrum of concerns compared with more convergent mathematical structures (Knuth, 1985), excluding the data structures of advanced set theory and logic.

Algorithmic thinking has been brought within reach of school or even pre-school children with multiple initiatives such as (Liukas, 2015). It may be well taught even without computers, as demonstrated by the CS-unplugged movement (Taub et al., 2012), and algorithmic plays (Futschek and Moschitz, 2010). Puzzles and games can be thought-provoking, thus this approach is also exploited by a number of universities in familiarizing students with algorithms (Lamagna, 2015). Unplugging removes the extra cognitive load of programming details.

Data represented and modeled in multiple ways

Multiple external representations (MERs) elucidate the data and problem from different perspectives. For example, a function may be represented as an expression, a curve, a map from argument set to image set, a table with two columns, or a function machine. Flexibility in moving from one representation to another indicates a deeper understanding of the concept (McGowen et al., 2000), which facilitates problem solving. Wilkie and Clark denote representational flexibility as fluency with the order of operations; commutative, associative, and distributive laws; and equivalence of expressions (Wilkie and Clarke, 2015). In programming, representational fluency is practiced, e.g., with the syntactic diversity of operations, such as addition: $x+y$, $+(x, y)$, or $(+\ x\ y)$.

Fig. 3 illustrates the use of the MathCheck learning tool (Valmari and Kaarakka, 2016) in studying the relationship between textual and tree representations. Such exercises aim at training the precedence and left- and right-associativity rules in particular. The exercises help students to grasp the distinction between semantics and syntax by differentiating between associativity as a semantic notion and left- and right-associativity as syntactic notions. Furthermore, the example in Fig. 3 reveals that the relation operators ($=$ and $\geqslant$, and so on) are neither left- nor right-associative unlike arithmetic operators ($+$, $-$, and so on). Consequently, in $x = y \geqslant z$, the first comparison result is not passed as an argument to the second, but instead, a Boolean AND is performed on both. Thus,

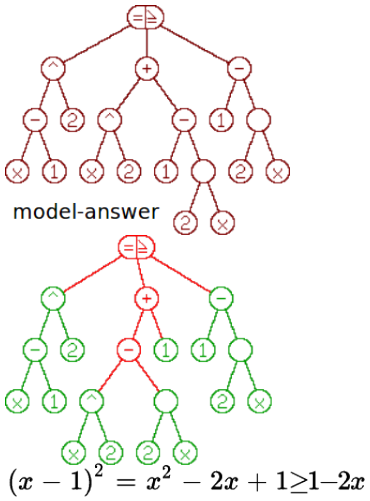

Figure 3: A tree representation of a model relation chain, and a failed student attempt to yield a similar tree

drawing $=$ as a child of $\geqslant$, or vice versa, would be misleading. Being even, the relation operators must share the root of a tree as Fig. 3 illustrates. This also makes it explicit that although y occurs only once, both comparisons use it as an argument.

In problem solving, the ability to model and abstract the data is crucial. USCC specifies Modeling as one syllabus area of HS math (Core Standards Organization, 2017; Core Standards Organization, 2015). Modeling links to a broader pedagogical idea of using open-ended problems of everyday life and it combines skills from math, statistics and technology, and ' *. . . and an ability to recognize significant variables and relationships among them. Diagrams of various kinds, spreadsheets and other technology, and algebra are powerful tools for understanding and solving these problems.*' Although modeling, say, a banking system for implementation as software is fundamentally different from modeling a physical or statistical problem, the need to recognize and formalize the essential aspects of the problem is common to all of them. Modeling requires 'specificational thinking', which is necessary for both SW engineers and their customers in order to reach a common vision, and describe use cases and requirements pellucidly. In FNC-2014, phenomenon-based learning approaches the anticipated open-endedness in problem setting.

Logic

In CS formalization, Dijkstra described its distinctiveness with a formula (Dijkstra et al., 1989): $CS = math + logic$. In accordance, he called students to learn formal math and logic to construct a well-grounded basis for CS. UKNC points out that already a novice programmer at the elementary level needs simple Boolean logic, for example, the operators of AND, OR and NOT, and combinations, see Fig. 4. In the same context, logic gates in circuits are introduced. This connection between Boolean logic and logic gates can be used to create a link with electrical engineering and physics.

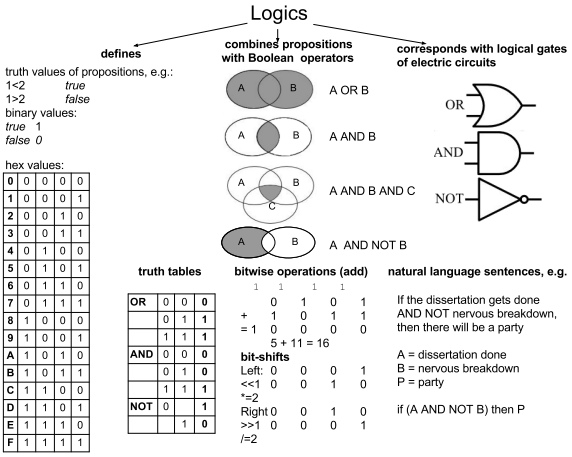

Figure 4: Logic in UKNC

To skim other logic uses, we reviewed ACM course descriptions. The chief applications were proofs, correctness, combinational and sequential logic of state machines, and in addition to these, logic of knowledge representation and reasoning that targets translating natural language (e.g., English) sentences into predicate logic statements. Such a skill would stand out when applying specificational thinking, check page 7.

Sets, statistics, probability

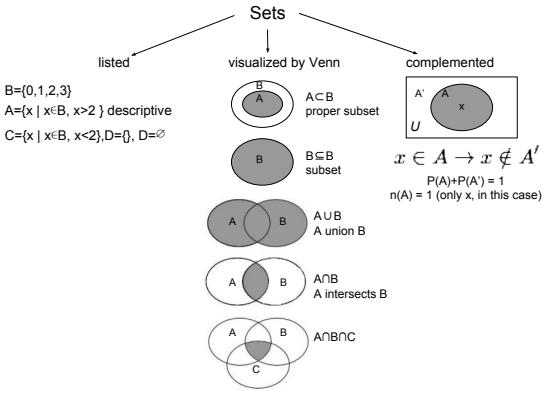

Figure 5: Sets in UKNC

The syllabus areas of sets, statistics and probability are inter-related at the elementary level, justifying a combination of these topics. Sets are missing from

FNC-2014, whereas UKNC defines a functional subset visualized in Fig. 5. Sets (naïve set theory) in UKNC are a gentle kick-start for the set theory, familiarizing students with different notations, e.g., the interchangeable use of either a list or a Venn diagram (excluding some special cases). A number of basic concepts are introduced, such as a set and its complement, a universe, and a subset. Set operations cover union and intersection.

Building the knowledge base and gaining experience of these topics may be initiated, for instance, by collecting data of concrete phenomena, such as measuring the heights of students of a class and constructing a histogram of the heights of the class. Students should be capable of reading and interpreting these charts. For instance, the shape of the height histogram should resemble the typical bell-shape of a normal distribution making it timely to introduce the concepts of mean, median, and mode in this context. In addition to histograms, the alternative way of

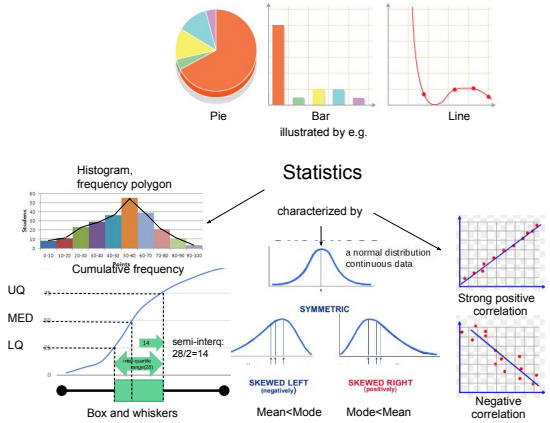

Figure 6: Statistics in UKNC

representing this information is to construct a cumulative frequency chart, in the UKNC subset visualized in Fig. 6, the left bottom corner. Ultimately, information could be reduced to a box-and-whiskers chart.

Venn diagrams and relative frequency charts prompt probability issues. The relative frequency of an event, e.g., which percentage of students are 140–150 cm tall, provides an obvious scaffold to investigate the probability of a randomly-selected student being 140–150 cm tall. In Venn, the bin of 140–150 cm students can represent the set A, where the complement set of $\overline{A}$ represents all the students not within this height category. In the universe of this class (or any other), a selector will get either a student from the set A or its complement $\overline{A}$ with 100% probability, i.e., $P(A) + P(\overline{A}) = 1$. In Finnish elementary math, probability links closely with statistics in the described manner. In contrast, UKNC progresses further by including the multiplication and addition rules (Fig. 7).

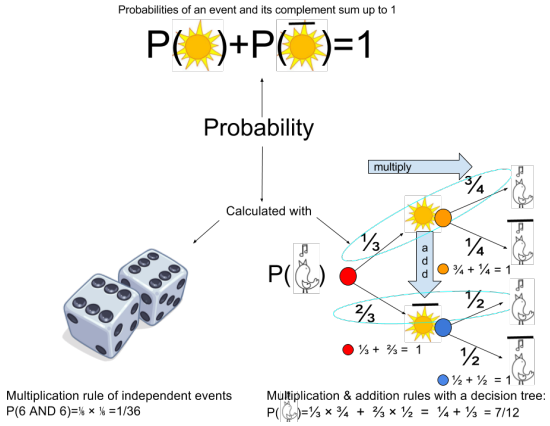

Figure 7: Probability in UKNC

The sun either shines or not, no other options exist. However, if the sun shines, a bird will sing more probably. A decision tree assists in constructing the combined probabilities correctly: the multiplication rule applies horizontally to each branch at a time, and the products are added vertically. In a tree, all the probability branches of one joint must sum up to one.

In preparation for CS and related math courses of higher-education including sets, statistics, and probability, UKNC specifies a valid and deliberately planned math syllabus for an elementary level that could be emulated as such in FNC-2014.

5.3 The learning trajectories bridged from elementary to higher-ed math

Fig. 8 divides into four horizontal layers: Elem.math, CT, HS math, and Tert.math. Elementary school is compulsory, the rest elective. Vertical dashed lines represent the learning trajectories of discrete math proposed in the previous sections. The learning trajectory of algorithms and data structures is marked with green to illustrate its prominence. Currently, elementary math does not specify any other learning targets of algorithms except the need for algorithmic thinking. It starts with problem solving and decomposition, which in programming implies subroutines. Ultimately in algorithms, e.g., the simplest sort and search algorithms were a natural learning goal. Data structures are prompted by number sets that correspond with variable types; in programming types can be, e.g., primitives or collections, such as arrays, lists, and vectors. Structuring data in various ways, modeling and visualizing it, assists in raising the abstraction level and in problem solving. The second most prominent trajectory is logic. Like algorithmic thinking, logic is included only as a requirement of

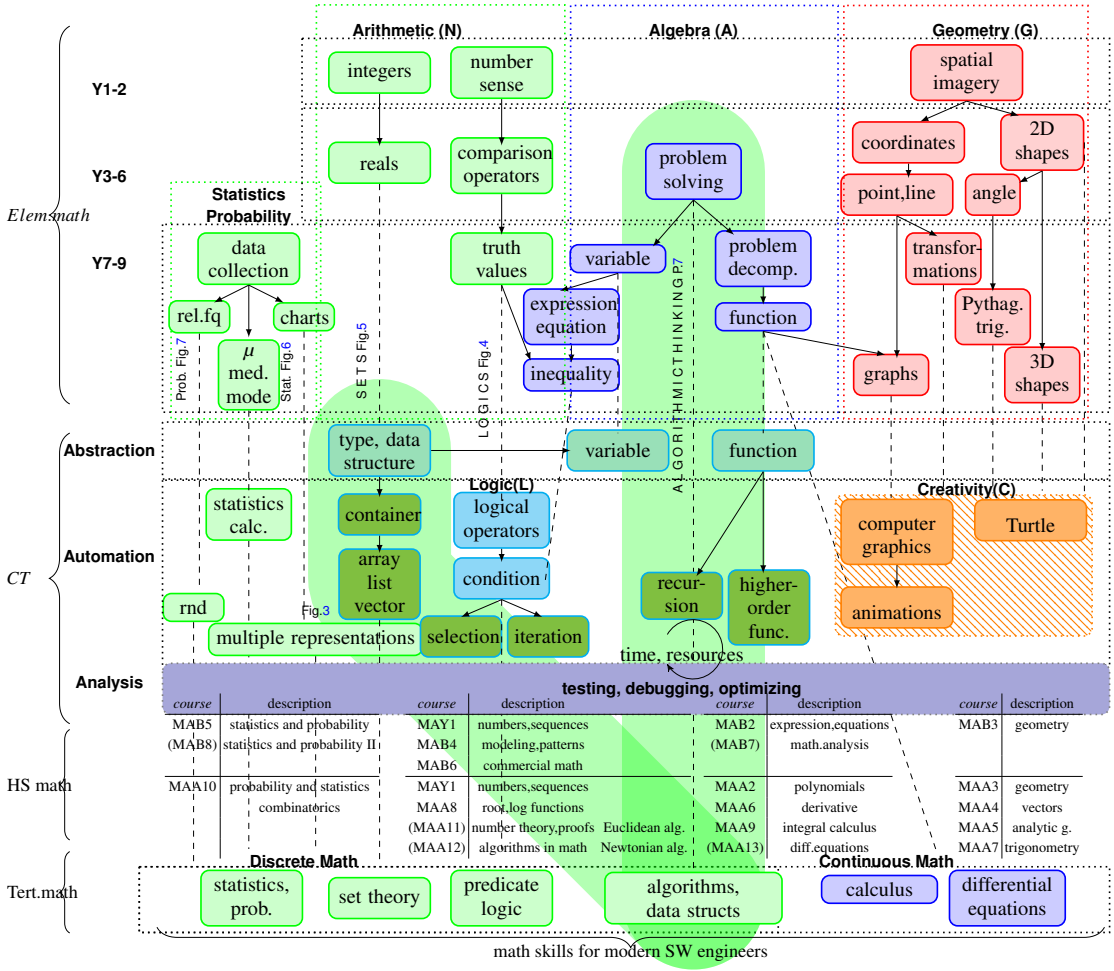

Figure 8: Hypothetical learning trajectories bridged from the FNC-2014 elementary to higher education math

logical thinking, except optionally in programming, where it is needed for the conditions of selection and iteration structures. The logic subset illustrated in Fig. 4 is proposed to partly enhance Y7–9 math, physics, and native language syllabi. To add further value to this age range, the UKNC syllabus areas of sets (Fig. 5), statistics and probability (Fig. 6 and 7) were worth considering in descending order of importance. However, due to time constraints, adding content to the math syllabus is problematic. CS, as a separate subject, would solve the problem. Below elementary math, the CT layer illustrates the computing enhancement and how the process divides into abstraction, automation, and analysis phases. In this layer, the math fundamentals have their computational counterparts. The math schedule (Y7–9) implies an appropriate introduction order of corresponding CS fundamentals.

The next layer of high school (HS) math is elective. It divides into A and B math: A is the major (ten compulsory MAA* courses, four electives), B being the minor subject (six compulsory MAB* courses, two electives) (Finnish National Board of Education, 2015). Regrettably, the HS math rigidly targets the matriculation exam, whose importance has lately grown as a selection criteria for tertiary education, thus extending the CT layer to cover the HS level is not topical. In HS, algorithms are introduced only in the elective courses of 'Number theory and proofs' (MAA11), and 'Algorithms in math' (MAA12). Closest to logic is the elective MAA11 with conjunctives and truth values. In regard to the remaining trajectories, sets are missing from the FNC-2014, both at the elementary and HS, whereas the situation of statistics and probability is brighter. They start already at the elementary, and HS allocates three courses to the topic: MAB5, MAB8, MAA10. Tertiary math elucidates the required skills for modern SWE by representing the most prominent topics only.

6 CONCLUSIONS

RQ1: Math syllabus areas to be strengthened? According to the reviewed studies, SW engineers need stronger algorithms and data structure skills. In accordance, fluency with multiple representations and modeling is considered beneficial in illustrating and structuring data, thus improving problem solving skills. To further strengthen the theoretical basis necessitates the inclusion/teaching of primarily logic, and secondarily set theory, statistics, and probability.

In increasing discrete math, the UKNC math and CS provide an exemplar to emulate in elementary education in Finland. USCC defines HS Modeling for structuring data, and the area could be subset age-appropriately for the elementary level. Modeling associates also with the use case/requirement specifications of SWE, which prompts the new term of 'specificational thinking'.

However, discrete math does not benefit only future SW engineers, but all students in becoming generally educated and acquainted with CS. Even though continuous and discrete math are posed as opposite, in practice, they are deeply interconnected and complement each other. Natural sciences continue to exploit continuous math as before, so continuous math must keep a significant role in the curriculum. However, to meet the challenges of digitalization, we believe that it is appropriate to move some emphasis from continuous to discrete math.

RQ2: The overemphasized math syllabus areas?

Curriculum planning is a zero-sum game. If the volume of discrete mathematics were increased, some areas ought to be decreased correspondingly. The proposal is to move some emphasis from continuous to discrete math already at the elementary level. For all the intended content the current time allocation is exiguous, which is why adding CS as a separate subject is a distinct option.

Further studies

The shifting of more emphasis to discrete math must be executed in an evidence-based manner, i.e., the learning outcomes must be carefully evaluated in cooperation with pedagogical experts both in elementary and higher education. To advance this approach further, the results should speak for themselves. To achieve the full potential of discrete math in higher-education, traditional '*Advanced Engineering Calculus*' would need its discrete math counterparts, say, '*Programmers' Introduction to Automata and Formal Languages*' or '*Set Theory for Software Engineers*', which indisputably explicate the benefits of the renewed math syllabus for the good of programming.

Is the time ripe for the next Lethbridge iteration to evaluate the current topics of the SWE curriculum?

ACKNOWLEDGEMENTS

Thanks to the Academy of Finland (grant number 303694; *Skills, education and the future of work*) for their financial support.

REFERENCES

ACM&IEEE (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, December 20, 2013. Technical report.

Ardis, M., Budgen, D., Hislop, G., Offutt, J., Sebern, M., and Visser, W. (2014). Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *Joint effort of the ACM and the IEEE-Computer Society*.

Beblavỳ, M., Fabo, B., and Lenearts, K. (2016). Demand for Digital Skills in the US Labour Market: The IT Skills Pyramid. CEPS Special Report No. 154/December 2016.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., and Punie, Y. (2016). Developing Computational Thinking: Approaches and Orientations in K-12 Education. In *EdMedia: World Conference on Educational Media and Technology*, pages 13–18. Association for the Advancement of Computing in Education (AACE).

Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.

Charette, R. N. (2005). Why software fails. *IEEE Spectrum*, 42:42–49.

Core Standards Organization (2015). Mathematics Standards — Common Core State Standards Initiative. `http://www.corestandards.org/wp-content/uploads/Math_Standards1.pdf`.

Core Standards Organization (2017). High School: Modeling. `http://www.corestandards.org/Math/Content/HSM/`.

CSTA (2016). Computer science standards. `https://www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf`.

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, 52(6):28–30.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6):33–39.

Department of Education (2014). National Curriculum in England. Key stages 3 and 4 framework document.

Dijkstra, E. W. et al. (1989). On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1398–1404.

English Department for Education (2013). National Curriculum in England Computing programmes of study.

Finnish National Board of Education (2014). Finnish National Curriculum 2014.

Finnish National Board of Education (2015). National core curriculum for general upper secondary education. http://www.oph.fi/download/172124_lukion_opetussuunnitelman_perusteet_2015.pdf.

Futschek, G. and Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. *Proceedings of the 2010 Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century (Constructionism 2010)*, pages 1–10.

GCSE (2015). GCSE subject content for computer science. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf.

Harris, M. (2014). The STEM shortage paradox. *Physics World*, 27(10):56.

Kitchenham, B., Budgen, D., Brereton, P., and Woodall, P. (2005). An investigation of software engineering curricula. *Journal of Systems and Software*, 74(3):325–335.

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, 92(3):170–181.

Lamagna, E. A. (2015). Algorithmic thinking unplugged. *Journal of Computing Sciences in Colleges*, 30(6):45–52.

Lethbridge, T. C. (2000). What knowledge is important to a software professional? *IEEE Computer*, 33(5):44–50.

Liukas, L. (2015). *Hello Ruby*. A childrens' book available in 22 languages.

McGowen, M., DeMarois, P., and Tall, D. (2000). Using the function machine as a cognitive root.

Meziane, F. and Vadera, S. (2004). A comparison of computer science and software engineering programmes in English universities. In *17th Conference on Software Engineering Education and Training (CSEE&T 2004), 1-3 March 2004, Norfolk, VA, USA*, pages 65–70. IEEE Computer Society.

Parnas, D. L. (1985). Software aspects of strategic defense systems. *Commun. ACM*, 28(12):1326–1335.

Parnas, D. L. (1999). Software engineering programs are not computer science programs. *IEEE Software*, 16(6):19–30.

Peng, G. (2017). Do computer skills affect worker employment? An empirical study from CPS surveys. *Computers in Human Behavior*, 74:26–34.

Pinar, W. F. (2012). *What is curriculum theory?* Routledge.

Puhakka, A. and Ala-Mutka, K. (2009). Survey on the knowledge and education needs of Finnish software professionals. *Tampere University of Technology, Department of Software Systems*.

Redecker, C. and Punie, Y. (2017). European Framework for the Digital Competence of Educators: DigCompEdu. EUR - Scientific and Technical Research Reports, The European Commission's science and knowledge service.

Smith, E. and White, P. (2017). A 'great way to get on'? The early career destinations of science, technology, engineering and mathematics graduates. *Research Papers in Education*, 32(2):231–253.

Surakka, S. (2007). What subjects and skills are important for software developers? *Communications of the ACM*, 50(1):73–78.

Taub, R., Armoni, M., and Ben-Ari, M. (2012). CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE)*, 12(2):8.

Valmari, A. and Kaarakka, T. (2016). MathCheck: a tool for checking math solutions in detail. In *SEFI 2016 Annual Conference Proceedings*, pages VK.1–VK.9. European Society for Engineering Education SEFI.

Wilkie, K. J. and Clarke, D. M. (2016; 2015). Developing students' functional thinking in algebra through different visualisations of a growing pattern's structure. *Mathematics Education Research Journal*, 28(2):223–243.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881):3717–3725.