Algorithms and logic as programming primers

Pia Niemelä[1], Antti Valmari[2] and Simo Ali-Löytty[3]

[1] Pervasive Computing, Tampere University of Technology, Tampere, Finland,
[2] University of Jyväskylä, Jyväskylä, Finland
[3] Mathematics, Tampere University of Technology, Tampere, Finland

Abstract. To adapt all-immersive digitalization, the Finnish National Curriculum 2014 (FNC-2014) 'digi-jumps' by integrating programming into elementary education. However, applying the change to mathematics teachers' everyday praxis is hindered by a too high-level specification. To elaborate FNC-2014 into more concrete learning targets, we review the computer science syllabi of countries that are well ahead, as well as the education recommendations set by computer science organizations, such as ACM and IEEE. The whole mathematics syllabus should be critically viewed in the light of these recommendations and feedback collected from software professionals and educators. The feedback reveals an imbalance between supply and demand, i.e., what is over-taught versus under-taught, from the point of the requirements of current working life. The surveyed software engineers criticize the 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, and logic: in short – discrete mathematics. The ground for discrete mathematics should be prepared early enough, started already from primary level and continued consistently throughout the secondary till tertiary education. This paper aims to contribute to the further refinement of the mathematics syllabus by proposing such a discrete mathematics subset that especially supports the needs of computer science education, the focus being on algorithms and data structures, and logic in particular.

Keywords: K–12 computer science education, programming in mathematics syllabus, digital skills gap, professional development of software professionals, effectiveness of education, continuous vs. discrete math, computational vs. specificational thinking

1 INTRODUCTION

Digitalization triggers pressure to change the current education system. Both domestic and multinational governing bodies have recognized the skills gap between 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 [39,6], highlighting the pervasive and ubiquitous nature of digitalization. The 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 re-employed more quickly than employees without digital skills [35].

The skills gap concerns not only the lack 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 [23,40]. One explanation is the skills mismatch. In compliance, employers point out the candidates' shortcomings, such as the incapability

of breaking down problems into manageable chunks and solving them, and the gaps in technical, data modeling, and analytical skills. In the US, for example, the skills related to data analysis, database skills, data management, and statistics outnumber other requested digital competencies of job advertisements [5].

The promotion of computer science (CS) education is global. In consequence, a number of countries all over the world have already introduced CS into their K–12 curricula. In line with others, FNC-2014 comprises algorithmic thinking and adapting good coding conventions as CS contents that are included in the mathematics syllabus [19], see Table. 1:

Table 1. Computing-related additions in FNC-2014. Typically, a student is 6–7 years old in Year 1.

	Years 1–2	**Years 3–6**	**Years 7–9**
Digital competence	using digital media, technological fluency	impact of technology, tech-integration	
Math	step-by-step instructions	visual programming	algorithmic thinking, good computing conventions
Crafts		robots, automation	embedded systems, own artifacts

In pursuit of consistent CS support, the entire mathematics syllabus should be reviewed along with these newly introduced additions. Thus, this study asks:

- RQ1: What elementary mathematics syllabus areas should be strengthened for the anticipated CS emphasis?
- RQ2: Are there mathematics syllabus areas that are currently overemphasized from this viewpoint?
- RQ3: According to the feedback from software engineers, is the current CS-supportive syllabus missing any crucial points?

First, Specifying the discipline of CS and relating it to mathematics chapter reviews the discourse of CS as a scientific discipline, its neighboring disciplines, and the learning targets of mathematics in anticipation of supporting CS. Related Work introduces already-existing directives and recommendations of institutions that aim at building flexible future software engineers, such as ACM/IEEE. There, we focus on suggested mathematics courses in particular. For an age-appropriate reality check, we reflect on the elementary-level mathematics and computing syllabi of current strong performers in CS, i.e., the UK and US. Results and Discussion cross-expose the recommendations with feedback from in-service software engineers by focusing on the mathematics topics that are the top-scorers in profitability. To sum up, Conclusions sketch a hypothetical learning trajectory for a CS support attached to the corresponding topics in the mathematics syllabus.

2 Specifying the discipline of CS and relating it to mathematics

Most natural sciences and engineering disciplines rely on calculus, differential equations, and linear algebra as a mathematical foundation appropriate for continuous phenomena [31]. 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 [33]. Unlike bridge load tests, testing a piece of software with typical and extreme values does not guarantee the 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 [9].

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 revealed. 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 profound learning of engineering topics. The basis is constructed 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 mechanisms suited for continuous phenomena, this mindset is not the same as that of, say, an electrical engineer.

The discussion about the educational needs in Finland suffers from a poor distinction between Computer Science (CS), Software Engineering (SWE), and Information and Communication Technology (ICT). For more than a decade, the Finnish mobile phone company Nokia was very successful and its educational needs had a significant 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, SWE and ICT were emphasized instead of CS, with SWE largely perceived analogous 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 to the mathematics 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 [34, p. 21]: physics belongs to the natural sciences, which target an understanding of a wide variety of phenomena, whereas electrical engineering is an engineering discipline striving to create useful artifacts. 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 mathematics, the latest specifications of Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE) explicate the similarity of required skills both in CS and SWE [3,4]. Even if CS is more scientific as a discipline and more deeply grounded in mathematics, SW engineers benefit from more theoretically-oriented CS education and discrete mathematics to be able to implement quality software. Hence, the conceptual difference does not diverge the required mathematics and CS fundamentals. Consequently, Meziane and Vadera concluded, '*There is very little difference between the SWE and CS programs currently offered in English Universities*' [30].

3 Related Work

3.1 ACM recommendations

The standards developed by the ACM promote CS as a discipline, and in compliance, provide normative recommendations for teaching CS at the tertiary level. The recommendations 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 in the considerations of advancing basics at the elementary school level.

CS Knowledge Areas of ACM ACM introduces Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (ACM-CS2013) [3]. 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 one or 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, where time allocation is thought to correlate with the importance of a topic:

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) by simultaneously strengthening 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 efficiency of algorithms based on their 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 hardware resources and memory, and, e.g., sequential and parallel execution.

From the list above, items 2 and 3 link closely with mathematics. 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. Algorithms and data structures are at the center of gravity of SDF, besides the introduction of the programming basics.

The most relevant mathematics to support CS ACM-CS2013 highlights the tight and mutual interdependence between mathematics and CS. However, ACM-CS2013 focuses on the common denominator part, instead of being prepared for a full range of different career options specifically. Thus, only directly relevant requirements are specified for the KA of DS, such as the elements of set theory, logic, and discrete probability. 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*' [3].

Thus, the recommendations make a distinction between such mathematics that is an important requirement for all students in the faculty, in distinction to mathematics that is relevant only to specific areas within CS, exemplified by 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*' [3].

If discrete mathematics – including logic – were emphasized in the elementary school mathematics curriculum, 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 taught discrete mathematics already for a longer period, see Section 3.3. In programming, logic is frequently employed, not only when implementing conditions in selections and iterations. Subsequently, university-level logic targets more sophisticated and far-reaching knowledge than these. Basic Logic of DS introduces such topics as normal forms, validity, inference rules, and quantification.

Although the domain of probability associates significantly weaker to the programming fundamentals than logic, for instance, it gives readiness for various prominent topics. These topics include the analysis of average-case running times, randomized algorithms, cryptography, information theory, as well as games.

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 [7]. Because of their direct mathematics linkage, we focus on both Chapters 13 and 14 of the guide in particular, i.e., Computing and Mathematical Foundations.

Computing Foundation comprises algorithms and data structures. The chapter classifies data structures based on following dichotomies: linear – nonlinear, homogeneous – heterogeneous, and stateful – stateless. For instance, linear structures organize items on one dimension (lists, stacks), in contrast to non-linear structures exemplified by trees and heaps. 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 foster algorithm development. When the effects of selected algorithms and data structures are combined, performance and memory consumption may range from poor to extremely efficient.

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

The following subtopics constitute the foundational KAs of mathematics. The topics are divided by us into continuous (c) and discrete (d) mathematics. The assumption is that the order implies their importance:

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/c)
7. Finite State Machines (d)
8. Grammars (d)
9. Numerical Precision, Accuracy, and Errors (c)
10. Number Theory (d)
11. Algebraic Structures (d)

Immediately, a notably smaller portion of continuous mathematics compared with traditional engineering education leaps out. In particular, calculus, differential equations, and linear algebra are conspicuous by their absence. 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, the 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, for instance, the resolution of measurements and impossibility of expressing most real numbers precisely.

3.3 CS-supportive mathematics syllabi of the UK and the US in K–12

Table 2. Mathematics syllabi (KS=key stage, G=grade, HS=high school. In the UK, each key stage covers several grades ranging from two to four; KS4 is followed by the GCSE exams.)

	UKNC	USCC
Algorithms	KS1: understand what algorithms are KS2: use logical reasoning to explain the functionality of simple algorithms KS3: key algorithms of searching and sorting, such as binary search and merge sort GCSE: data handling and algorithms visualized with flowcharts	G3: algorithms in problem solving heuristics, e.g., based on place value in addition/-subtraction G5: the same with multiplication G6: and division
Logic (in CS)	KS2: logical reasoning, e.g., in explaining the functionality of simple algorithms KS3: Boolean logic (AND/OR/NOT)	General: Construct viable arguments and critique the reasoning of others. Logical progression of statements
Sets	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 tables, trees 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
Stat Prob	Chart interpretations(K2), distributions/ rel. frequencies, bivariate data, P scale: 0–1, P of mutually exclusive events(K3); bigger samples, population, histograms, scatter and box plots, combined dependent events(K4)	Variability(G6), generalizations and random sampling (G7), bivariate data(G8)
Linear algebra	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

For comparison, we went through the National Curriculum (UKNC) and General Certificate of Secondary Education (UKGCSE) of the UK [16,22,8], and the Core Curriculum of the US (USCC) [11], see Table 2. In UKNC and USCC, all the suggested mathematics syllabus areas remain at the basic level, which is necessary taking into account elementary students' rudimentary abstraction skills. In addition to reducing its complexity, the new content should be carefully bridged with the prior knowledge by starting early enough, proceeding in spiral revisits, and by exploiting lots of different type of exercises including hands-on exercises and visual clues. To ensure mathematics-compatibility, a hypothetical learning trajectory is sketched and further extended to the proposed CS topics. Next, we will review potential discrete mathematics contents in UKNC and USCC starting from algorithms and data structures, logic, sets, statistics and probability, and ending up with linear algebra.

Algorithms and data structures In UKNC, the definition of algorithms in KS1 is followed by studying the behavior of selected algorithms in KS2, after which the key algorithms are digested. The CS syllabus of the GCSE sets a few learning targets for algorithms: at a minimum, binary search and merge sort are to be introduced [22]. In USCC, since CS is not compulsory, algorithms are not that pertinently present, however, computing strategies used in problem solving, are perceived as algorithms. Paper-and-pencil calculations provide affordances of applying such algorithms, for example, long division may be systematized as the strategy of 'divide, multiply, subtract, drop the next free number, repeat'. Because of all-immersive digitalization, many such educational sweet-spots are missed, e.g. phone books for demonstrating search algorithms.

Logic The basics of logic are present in UKNC. A comprehensive subset is provided, yet in the UK Boolean logic is currently included in the computing and not in the mathematics syllabus. However, Boolean logic would also fit well in the mathematics syllabus as a consistent continuum of studying inequalities and their truth values. A readily field-tested elementary syllabus is found in GCSE CS [22] as well. It contains the following topics:

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

In USCC, logic is substituted by logical thinking as part of critical thinking skills and consistency in arguments. In addition collaborative learning techniques are utilized, in particular sociomathematical norms [46]. Yackel and Cobb emphasize the need for a rationale and justification for a solution. It is a mathematics teacher's duty to challenge students to invent multiple alternatives ending up with the same result. Among the presented alternatives, the class should evaluate the most sophisticated and elegant method.

Sets, statistics, and probability Sets may be employed to 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 programming language selected. All the same, not every *int* is necessarily a *float*, in contradiction to the mathematics subset relation of $\mathbb{Z} \subset \mathbb{R}$. In mathematics, sets are a basic abstraction of containment, likewise in programming, they could be exploited as a cognitive scaffold that assists in understanding e.g. collections and their operations. The same operations are exploitable even if the collection type would change from a set to an array, a list, a vector or a matrix. Therefore, set theory would be useful in any mathematics syllabus designed to support CS. Currently, sets are a part of UKNC, but absent from USCC and FNC-2014. Sets prompt types in programming and they can be utilized in abstracting both primitives and collections. UKNC specifies the syllabus of sets containing the following topics:

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

Statistics and probability have only a small role in the conceptual core of CS, however they are useful tools for further studies, e.g., statistical analyses in STEM reports or probability exploited in game applications.

Linear algebra 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. However, linear algebra basics could be a beneficial addition, even if supported by ACM-CS2013 only as an elective mathematics topic. The need for matrices is increasing, because of topicality of their application areas. Hence many libraries, e.g., in Python exploit them extensively. As a topic, matrices and vectors (that can be handled as matrices) belong together. Matrix manipulations, such as transformations (scaling, translation, reflection and rotation) are especially applicable in many popular fields of graphics, animations and game engines. In addition, matrices are extensively exploited in machine learning, data analysis, and pattern recognition.

4 Method

This study complies with the scope of curriculum theory [37], and its key question of what knowledge is the most valuable and how the knowledge should be constructed in order to ensure consistent proceeding. 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 mathematics and compares FNC-2014 with UKNC and USCC [16,18,11] and with the recommendations given by the ACM and IEEE [3,7]. The comparison exploits content analysis in searching for the mathematics 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 [27,38,41,24]. The results of the previous studies are cross-correlated to confirm their validity in order to draw conclusions about the most profitable mathematics 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. Being informed of both the recommendations and criticisms of the current realization, we summarize the necessary mathematics syllabus content and bridge the learning trajectories from elementary to higher education mathematics.

5.1 Feedback from SW engineers

To evaluate the effectiveness of their education, SW engineers have scored the profitability of plenty of curriculum topics [27]. 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 the year 1997 and repeated in 1998. The differences between outcomes 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 British universities [24]. 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 [38] ($N = 212$). 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 competence 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

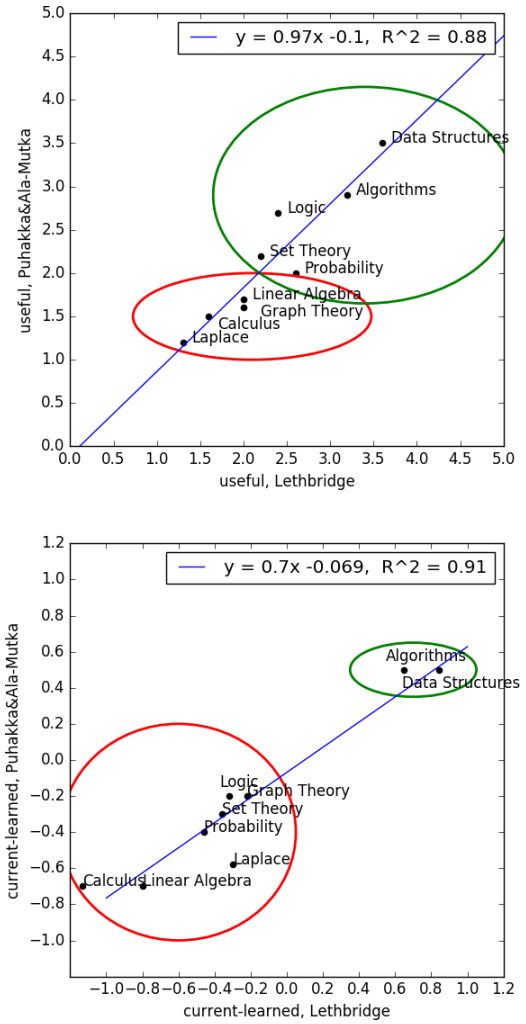

Fig. 1. The comparison of usefulness and adequacy of mathematics education evaluated by SW professionals [27,38] ($N = 181$; $N = 212$), originally in [31].

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.

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 mathematics compared with continuous mathematics is unchallenged, yet the bias has an academic flavor: discrete mathematics scores the highest among professors and lecturers (3.1).

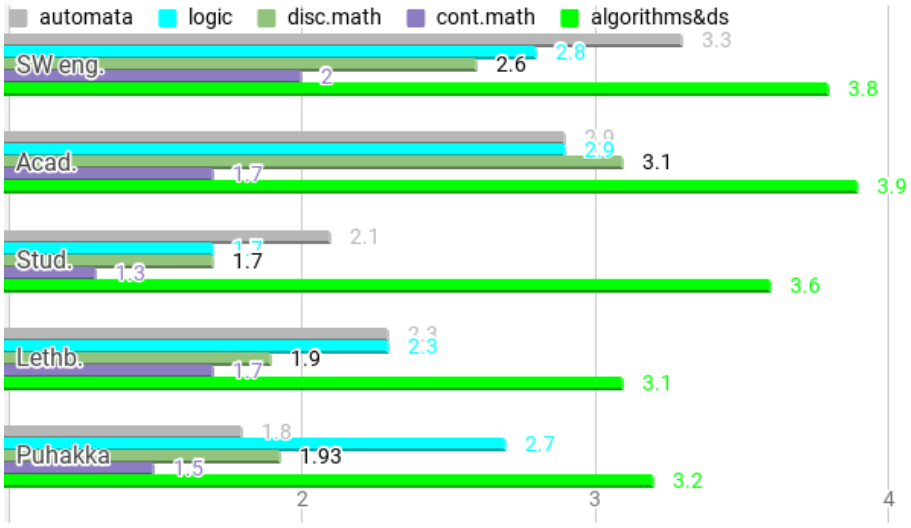

Fig. 2. The mathematics areas perceptions [1(not important), 4(very important)] of Surakka's engineers, academics, and students contrasted with Lethbridge and Puhakka et al.; $N = 11, 19, 24; 181; 212$ – respectively. Originally in [31].

5.2 TEK, Aarresaari

The society follows the effectiveness measures of higher education. In a yearly basis, the association of Academic Engineers and Architects in Finland, TEK, collects the feedback of university graduates. The latest survey, 2017 TEK graduate survey, is referred to update the current emphasis areas of the graduates [43, p.21].

Unfortunately, in regard to mathematics, the granularity of the survey is more coarse-grained than in the previous studies examined. The more general information addresses the importance of certain skills, such as problem solving and information retrieval skills. Only two aspects are studied more than anticipated by their importance, i.e., the difference of importance − learned in studies is negative. The two areas are 'knowledge of the research of the own field' and 'mathematics and natural sciences', that is, they are learned more than actually needed in working life. In comparison,

'practical application of theories' is one of the top-scorers. 'Problem solving' is regarded the most important skill among graduates, and after project management and oral communication skills, analytical thinking also scores highly. In Aarresaari data, learning skills in overall and self-regulatory skills in particular are valued the highest, exemplified by such skills as 'ability to learn' and 'self-steerability/initiative' [2]. These key areas intimately reflect the current requirement of a flexible workforce capable of recreating itself based on the current need. Also analytical thinking skills and problem solving score high. In contrast, 'theoretical skills' are valued much lower to their applicative counterparts, and 'theoretical knowledge of one's own domain' and 'mathematics and natural sciences' actually count among few topics over-taught, i.e., during their studies, the graduates learn them maybe too theoretically without exploiting the learned content in practice.

Fig. 3. TEK graduate survey in 2017; $N = 1985$

The TEK survey for university graduates confirms the findings of over-taught topics: 'knowledge of the research field' and 'mathematical and natural science', see Fig. 3. Even if the theoretical aspects of one's research field are over-emphasized, the practical application of theories is yet insufficient. Moreover, the resolution is partly lost in bundling mathematics and natural sciences together. Lethridge/Puhakka/Surakka studies suggest chemistry being constantly among the low-scorers, whereas parts of mathematics, especially those fostering algorithmic and logical thinking, are appreciated.

In the TEK and Aarresaari survey, high-valued self-regulatory skills are seconded by such practical aspects that can be situated under the broader umbrella concept of 'specificational thinking'. These skills comprise, e.g., communication and negotiation skills with a client as part of user-centered design. However, the specificational thinking in its entirety is more about modeling and abstracting data, which as topics were absent from these surveys. Specificational thinking will be treated more thoroughly in Ch. 5.5.

5.3 CS-supportive mathematics for primary and secondary education

In constructing a strong basis for CS, both ACM and SWEBOK emphasize discrete mathematics, confirmed by the feedback from the field. After programming basics, ACM values discrete systems as the second most prominent, and algorithms, data structures, and complexity as the third most prominent KAs, whereas the in-service SW engineers appreciate the latter more. In SWEBOK, nine out of eleven mathematics KAs comprise discrete mathematics. Spearheading in CS, the UK invests in discrete mathematics already at the elementary level and, in addition, provides CS as a subject of its own right that allows more in-depth topics.

Algorithmic thinking 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 [25]. Denning equates algorithmic and computational thinking [14], which he, in turn, associates with general problem solving [15]. When solving a problem, it is beneficial to start by decomposing it to smaller solvable subproblems that may be implemented as subroutines in a code. At its simplest, an algorithm may thus be understood as a subroutine, a sequence of commands that can be called repeatedly as many times as desired [13, for instance]. Fig. 4 suggests dividing the algorithms into the following sub-topics:

1. The introduction of the key algorithms of searches and sorts. These can be trained without computers as well.
2. Complexity considerations.
3. Data structures and corresponding operations.

Algorithmic thinking has been brought within reach of school or even pre-school children with multiple initiatives such as [28], and without utilizing computers, as demonstrated by the CS-unplugged movement [42], and algorithmic plays [21]. Puzzles and games can be very educative and thought-provoking, thus this approach is also exploited by a number of universities in familiarizing students with algorithms [26]. Unplugging removes the extra cognitive load of knowing the programming details. To be acquainted with well-known algorithms, binary-search and merge-sort are considered an age-appropriate start, backed up with the CS syllabus of UKNC.

In general, a student should learn to save resources, i.e., time and memory, as a part of good coding conventions. In algorithm development, it is

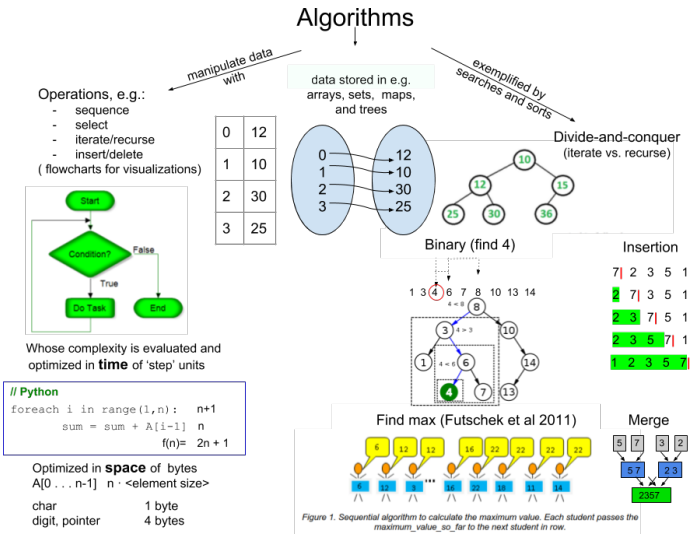

Fig. 4. Algorithms at elementary level

crucial to select an appropriate algorithm and data structure for a task. In consequence, a student must be aware of the consequences of poor choices, where estimating the complexity of the algorithm is educative. The combined effect of algorithm and data structure can be huge. In extending the data amounts, the differences highlight further. The same control structures are applicable as usual, i.e., sequencing, selection, iteration and recursion. In visualizing the control flow, a flow chart is educational. Iterations and recursions, especially if nested, can easily increase the number of iterations.

Data structures: sets and other representations In programming-oriented mathematics, data structures can be seen as an application of set theory that conceptualizes collections. 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.

In programming, collections are of various types: a set is an unordered collection of values where no value may occur twice, a list an ordered collection where the same value may occur multiple times, and a map a collection of values identified by keys; the map may also be interpreted as a representation of a mathematical function. Data structures should introduce the very basic structures, such as arrays, lists, maps and optionally also more sophisticated tree structures, and demonstrate the efficiency of each basic operation of adding, deleting, and selecting (searching). Sorting can be interpreted as an application of search in compliance with divide-and-conquer heuristics: after an item is found and in the right position, search is applied iteratively till all items are sorted.

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

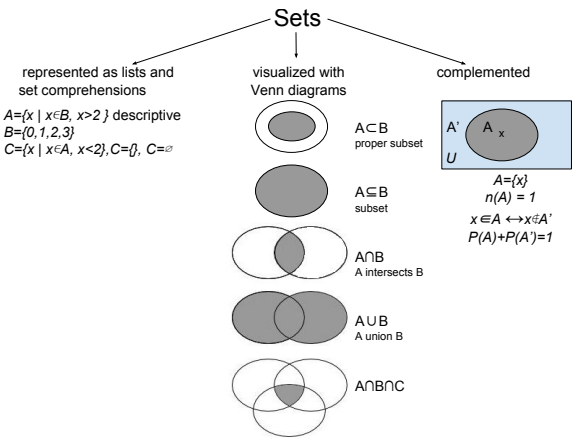

Fig. 5. Sets in UKNC

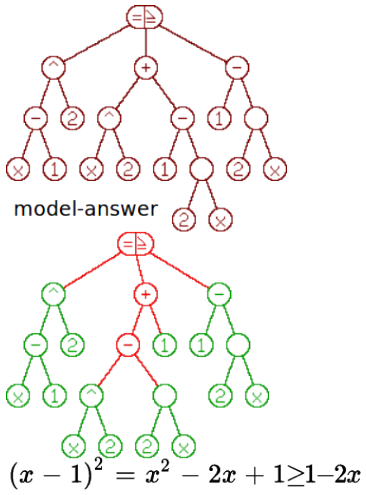

Fig. 6. A tree representation of an arithmetic relation chain, and a failed attempt to yield a similar tree [31].

indicates a deeper understanding of the concept
[29], 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 [45]. 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. 6 illustrates the use of the MathCheck learning tool [44] 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. 6 reveals that the relation operators ($=$ and $\geq$, and so on) are neither left- nor right-associative unlike arithmetic operators ($+$, $-$, and so on). Consequently, in $x = y \geq z$, the first comparison result is not passed as an argument to the second, but instead, a Boolean AND is performed on both. Thus, drawing $=$ as a child of $\geq$, or vice versa, would be misleading. Being even, the relation operators must share the root of a tree as Fig. 6 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 a syllabus area of HS mathematics [12,11]. It links to a broader pedagogical idea of using the open-ended problems of everyday life by combining skills from mathematics, statistics and technology, and ' *. . . and the 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. Specificational thinking is necessary for both SW engineers and their customers to reach a common vision.

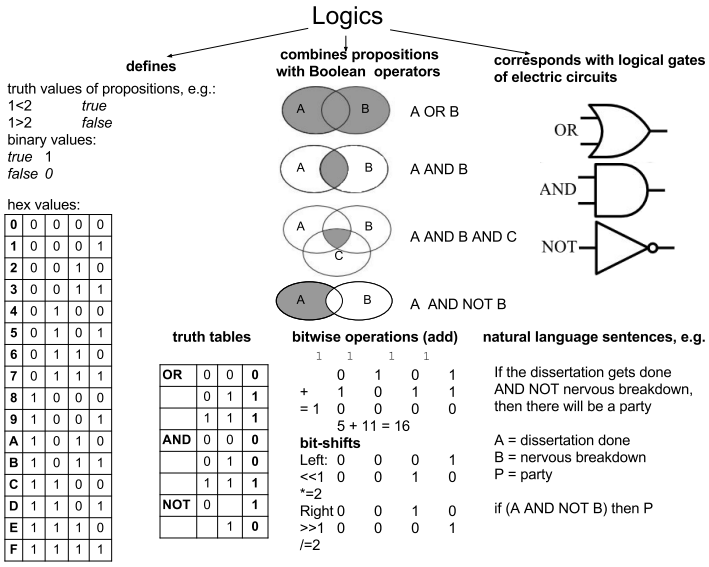

Fig. 7. Logic in UKNC [31]

Logic In formalizing CS, a formula, $CS = mathematics + logic$, proposed by Dijkstra, aims at describing its distinctiveness [17]. In accordance, he calls students to learn formal mathematics and logic to construct a well-grounded basis for CS. UKNC points out that already at the elementary level a novice programmer needs simple Boolean logic, at least the operators of AND, OR and NOT, and their combinations, see Fig. 7. In the same context, UKNC introduces logic gates in circuits, thereby creating a link between CS and physics (electronics).

To skim other logic uses, we reviewed ACM course descriptions. The logic applications were proofs, correctness, the combinational and sequential logic of state machines, and in addition to these, reasoning that targets translating natural language (e.g., English) sentences into predicate logic statements. Such a skill would stand out in specificational thinking in Ch. 5.5.

Statistics, probability The syllabus areas of statistics and probability are inter-related at the elementary level, justifying combining the topics under the same label. Building the knowledge base and gaining experience of these topics may be initiated, for instance, by collecting the 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 representing this information is to construct a cumulative frequency chart, in the UKNC subset visualized in Fig. 8, the left bottom corner. Ultimately, information could be reduced to a box-and-whiskers chart.

Venn diagrams and relative frequency charts prompt probability. The relative frequency of an event, e.g., which percentage of students are in the range of 140–150 cm, provides an obvious scaffold to investigate the probability of a randomly-selected student being 140–150 cm tall, and prepares for generalizations concerning bigger populations. 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., $P(A)+P(\overline{A}) = 1$.

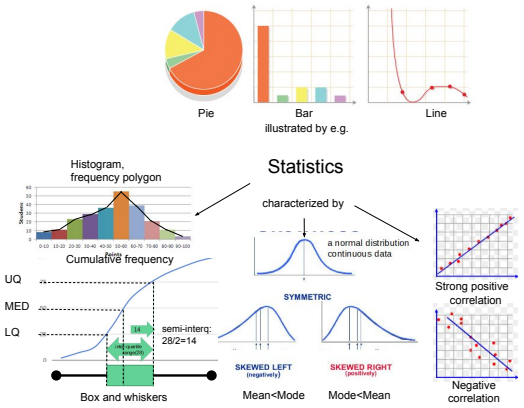

Fig. 8. Statistics in UKNC [31]

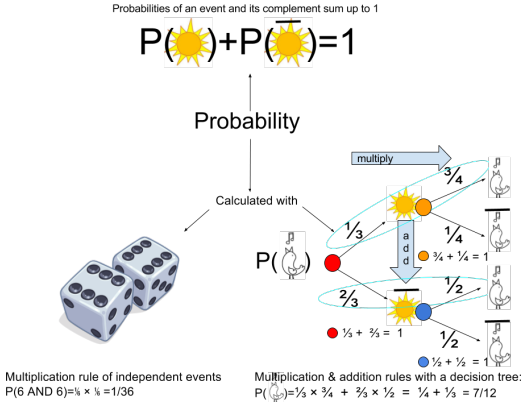

Fig. 9. Probability in UKNC [31]

In Finnish elementary mathematics, probability links closely with statistics in the described manner. UKNC progresses further by including the multiplication and addition rules, see Fig. 9. In the figure, a decision tree clearly demonstrates that the sun either is shining or not, no other options exist. As a consequence of the shining sun, a bird will sing more probably. Furthermore, the 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. For statistics, and probability, UKNC specifies a valid and deliberately planned mathematics syllabus for an elementary level that could be emulated as such in FNC-2014.

5.4 CS-supportive mathematics at high school

In the Finnish high school system, CS is not provided as a subject of its own. Instead, the cross-curriculum thread of digital competence started at elementary level continues as the utilization of CS and practicing technological fluency throughout the high school. After completing the first course of mathematics (MAY1), a student either chooses honors mathematics (the MAA* courses), or regular mathematics (the MAB* courses) [20]. The respective courses are listed in Table 3. This study concentrates on honors mathematics in particular, which comprises ten compulsory courses (MAY1-MAA10), and three optional courses (MAA11–MAA13), because of its partial support for discrete mathematics. The courses of discrete mathematics consist of 'Number theory and proofs' (MAA11) and 'Algorithms in mathematics' (MAA12) that are currently optional.

In HS, algorithms are introduced only in the elective courses of 'Number theory and proofs' (MAA11), and 'Algorithms in mathematics' (MAA12). Closest to logic is the elective MAA11 with conjunctives and truth values. Instead, we propose a compulsory status for these courses to ensure a proper support for CS. Due to the proposal, some compulsory courses ought to be converted as optional as a fair exchange. Maybe the advanced courses of derivation and integration could be potential candidates, since the SW engineers' feedback manifest the excess of continuous mathematics, such as calculus. Moreover, the electric version of matriculation examination will be taken into use in the year 2019 implying hands-on exercises with computers. The developing of these questions is currently proceeding. In anticipation of this change, mathematics teachers should prepare their students to master the required applications and programmable environments.

Table 3. High school mathematics divided in MAA* and MAB* courses and grouped by learning trajectories (stat. and prob., logic, algorithms, cont.mathematics, and geometry), see Fig. 11.

stat. and prob.		arithmetic(logic/algor. in yellow)		cont.mathematics		geometry	
course	description	*course*	description	*course*	description	*course*	description
MAB5	stat. and prob.	MAY1	numbers,sequences	MAB2	expression,equations	MAB3	geometry
(MAB8)	stat. and prob. II	MAB4	modeling,patterns	(MAB7)	mathematics.analysis		
		MAB6	commercial math				
MAA10	prob. and stat. combinatorics	MAY1	numbers,sequences	MAA2	polynomials	MAA3	geometry
		MAA8	root,log functions	MAA6	derivative	MAA4	vectors
		(MAA11)	number theory,proofs	MAA9	integral calculus	MAA5	analytic g.
		(MAA12)	algorithms in math	(MAA13)	adv.calculus	MAA7	trigonometry

5.5 Specificational thinking

In the SW engineers' feedback, specificational thinking and related skills become ever more pronounced. Specificational thinking shares certain analogies with computational thinking, and it could be described as its practical cousin. Taken that computational thinking comprises the theoretical basis and the acquaintance of a software process, at least in the abstract, specificational thinking extends to the real working life and project-based conditions, where the threads of modeling and user-centric design mix in, see Fig. 11. Modeling implies both data modeling and conceptual visualizations in order to describe the system. User-centered design attempts to ensure such products that respond to users' expectations and needs, which starts by specifying user requirements, often referred to as user stories in agile project management.

As a complementary part of the negotiation skills that were highly appreciated in the SW graduate surveys, capturing all the essentials in a specification benefits from an adequate amount of domain knowledge and observations as a typical practice of user-centered design. Translating all the information and observations as clearly-worded specifications – while minimizing the chances for misunderstandings – requires a sense of the nuances of a spoken language, preciseness, and the capacity for recognizing the sentences as implicit logical propositions. First, use cases and requirements are defined together with a customer. It is difficult to design SW so that it meets the needs of its end users. Indeed, [9] lists twelve common causes for SW project failures. Three of them are unrealistic or unarticulated project goals; badly defined system requirements; and poor communication among customers, developers, and users. We believe that *specificational thinking* alleviates these problems: to provide usable and user-friendly products requires a fair reflection on the actual needs and expectations of end users, and clothing them as precise specification text.

In writing a good specification, it is hard to anticipate all its consequences, especially if one is not a professional, which is illustrated by the next example. A man sitting in a wheelchair tried to buy winter boots at one supermarket, in Helsinki [1]. For the purpose, he had received a voucher worth at most 70 euro granted by social security authorities. However, the shoes did cost 74.50 euro. The remaining part, 4.50, the man would have paid himself. However, a cashperson refused and appealed to the instructions. After a while, a superior of the cashperson arrived, the next arrival being a safeguard. Finally, an outsider paid the winter boots from her own money to resolve the awkward stalemate. Afterwards, the supermarket analyzed what went wrong. Cashpersons had been given a written specification of how to process the social security vouchers: first, check the maximum value of a voucher, then, a purchase not exceeding that value. The cashperson obeyed the instructions literally. However, this was not the intention, but instead, to prevent from using the vouchers as worth of more than their maximum, i.e., they are no reduction coupons. The supermarket fixed the instructions and returned the money to the customer who had paid the boots.

At first glance, this incident may seem to have nothing in common with SW development, however, it illustrates such defects in specificational thinking that are a major source of problems throughout an SW process: it is very hard to see the unintended consequences in advance. Beforehand, the authors may think that they have written a decent specification; afterwards, it is self-evident to everybody that it allows some drastic and unintended interpretations. Commonly, a failure scenario is considered too improbable, too crazy, to worry about, until it does occur. Concerning software project failures more generally, the subtitle of [9] is '*We waste billions of dollars each year on entirely preventable mistakes*'. The publication lists many examples and argues, '*Even organizations that get burned by bad software experiences seem unable or unwilling to learn from their mistakes.*'

Another aspect of specificational thinking is the ability to choose an appropriate representation for the data and required operations. In bigger organizations, it is a duty of an SW architect to translate the specification into an architectural design, often illustrated as UML diagrams. To be capable of making efficient and unambiguous designs requires technical skills, awareness of the variety in possible data structures, and their implications for efficiency and required resources. For instance, Fig. 10 shows two fragments of Tampere region bus timetables:

Line 2 Rauhaniemi–Pyynikintori

05	10	36	48			
06	00	12	24	36	48	
07	00	14	26	38	50	
08–14	02	14	26	38	50	
15	02	07	14	26	38	50
16–18	02	14	26	38	50	

Line 55 Vesilahti–Tampere

A	B	C	D	E	F
06:20	06:35	-	06:50	06:59	07:25
07:30	07:45	08:02	08:05	08:15	08:41
08:40	08:55	-	09:07	09:17	09:39
-	09:45	-	09:57	10:07	10:29
-	-	-	10:40	10:50	11:12

Fig. 10. Two fragments of bus timetables (Monday–Friday)

On the left, full hours are shown in the first column, and the rest of each line lists departure times in minutes after the full hour. On the right, each line represents a route via bus stops from A to F: departure times are shown stop-by-stop; '-' denotes no visit. Students could be asked to discuss the advantages and disadvantages of these two representations, and possible justifications for each.

In avoidance of the unintended consequences and bad design, it is crucial to admit the very existence of these specificational problems, them necessitating specificational thinking to be taken seriously. To exercise it at elementary level, an educative and sufficient learning objective would be to deal with real and open-ended specification problems. For instance, a teacher could ask a student to write instructions for another student to follow. The instructions may describe, say, a location of an object hidden in the schoolyard. Afterwards, the students should discuss how the instructions were to be improved to find the object even quicker. By the same token as those 'follow-my-instructions' games, programming can provide epistemologically productive learning experiences. Papert claims that, *'...in teaching the computer how to think, children embark on an exploration about how they themselves think. The experience can be heady: Thinking about thinking turns the child into an epistemologist'* [32]. Sooner or later, a novice will notice that a computer functions differently from what he intended because it obeyed his instructions precisely. The situation is akin to the example of buying boots, where reaching a common understanding between humans was tricky, yet a computer is even more stubborn in its obedience.

The modest goal of these exercises is that in a future our students will be more discerning and resourceful in specifying and implementing SW projects than decision makers of today.

5.6 The learning trajectories bridged from elementary to higher education mathematics

To track the consistent proceeding in learning, we draft a hypothetical syllabus of CS-supportive mathematics by enhancing it with discrete mathematics. For learning proceedings, 'learning trajectory' is the selected theoretical framework rooted in Piaget's cognitive constructivism [36] and

active learning theories [10]. It targets a definition of a consistent path for a learner to follow. The path should consist of well-justified building blocks. Fig. 11 illustrates the learning trajectories as vertical dashed lines, dedicated to each topic proposed in the previous sections: algorithms and data structures that comprise sets, logic, statistics, and probability. The trajectories are crossing through four horizontal layers of Elementary mathematics, computational thinking, HS mathematics, and Tertiary mathematics. In Finland, only elementary education is compulsory, whereas continuing to high school or tertiary education is elective.

The learning trajectories of algorithms and data structures, and logic are marked with light green and blue to highlight their prominence. Currently, the elementary mathematics syllabus in FNC-2014 does not define any specific learning targets for the topics except that of 'algorithmic thinking' anticipated to start with problem solving and decomposition. In programming, the decomposition implies a program's division into subroutines. In algorithms, the introduction of the simplest sort and search algorithms would be a natural learning goal. Data structures are prompted by number sets of natural numbers ($\mathbb{N}$), integers ($\mathbb{Z}$), and reals ($\mathbb{R}$) that match with variable types (*unsigned*, *int*, *float*) in programming. In addition to simple primitives, types in CS can be more complex, such as primitive containers of arrays, lists, and vectors. Structuring data in various ways, modeling and visualizing it, assists in raising the abstraction level, thus ultimately in problem solving as well.

The second most prominent trajectory is logic. Like algorithmic thinking, in FNC-2014, logic is included only as a requirement of logical thinking. However, in programming, logic is highly exploitable in defining the conditions in selection and iteration structures. The logic subset in Fig. 7 proposes enhancements to mathematics, physics, and native language syllabi in Y7–9. To add further value to this age range, the UKNC syllabus areas of statistics and probability (Fig. 8 and 9) were worth considering in descending order of importance. However, due to time constraints, adding content to the mathematics syllabus is problematic. CS, as a separate subject, would solve the problem. Below elementary mathematics, the computational thinking (CT) layer illustrates the computing enhancement and how the process divides into abstraction, automation, and analysis phases. In this layer, the mathematics fundamentals have their CS counterparts. The schedule in mathematics Y7–9 implies an appropriate introduction order of the CS fundamentals as well.

In regard to the hypothesized trajectories, sets are unfortunately missing from the FNC-2014, both from elementary and high school education, whereas the situation of statistics and probability is much brighter. They start already at the elementary level, and in high school the following courses are allocated for the topic: MAB5, MAB8, and MAA10. However, high school is elective, and, regrettably, rigidly targets the matriculation examinations, whose importance has lately grown as a selection criterion for tertiary education. Tertiary mathematics elucidates the required skills for modern SWE by representing the most prominent topics only that can be considered as a continuum of the trajectories sketched in the figure.

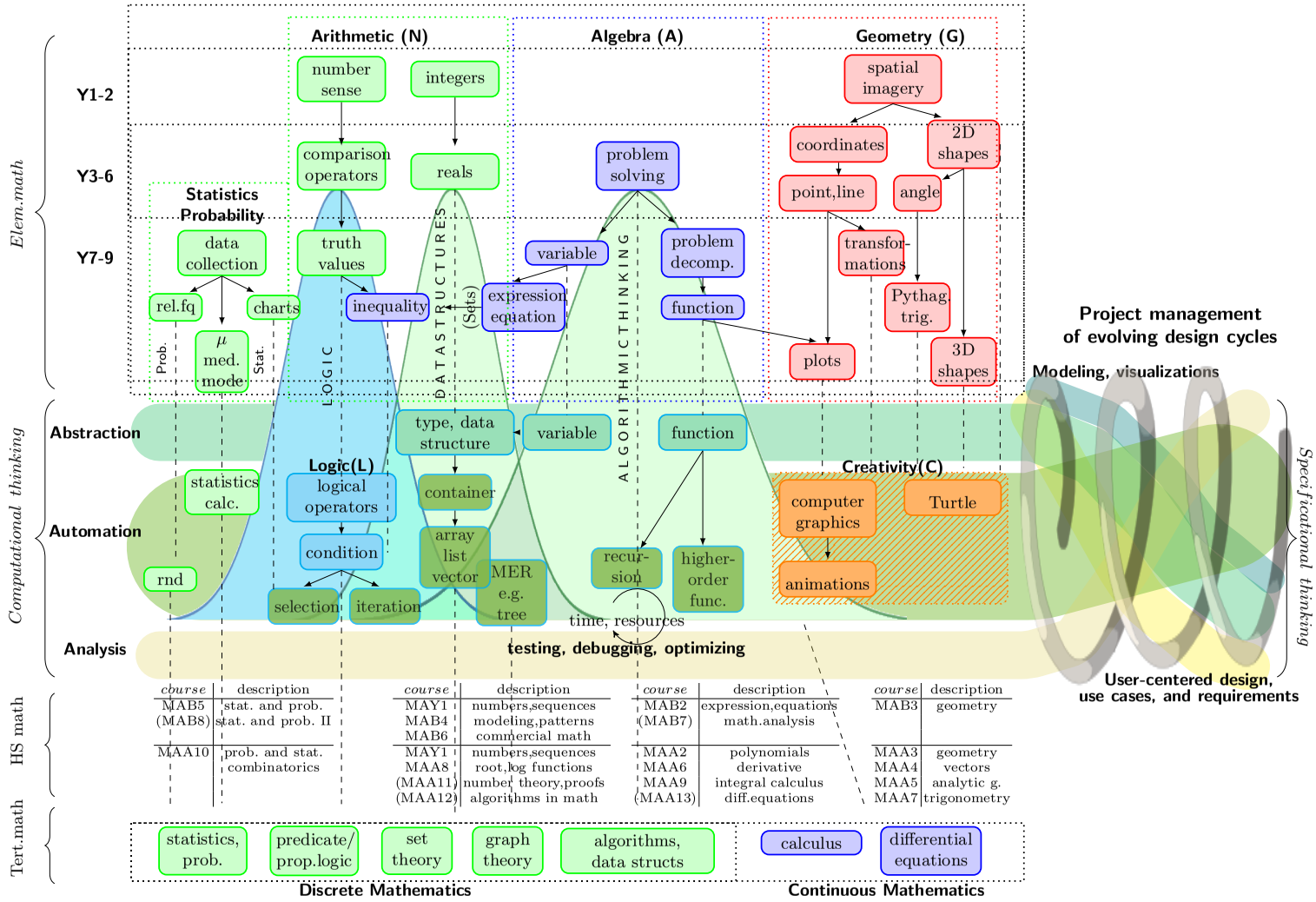

Fig. 11. Hypothetical learning trajectories bridged from the FNC-2014 elementary to higher-education mathematics

6 Conclusions

RQ1: Mathematics 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 primarily necessitates the inclusion/teaching of logic, and secondarily set theory, statistics, and probability. In increasing discrete mathematics, the UKNC mathematics and CS provide an exemplar to emulate in elementary education in Finland.

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

RQ2: The overemphasized mathematics 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 mathematics already at the elementary level. To get all the suggested content to fit in the mathematics syllabus is challenging, thus adding CS as a separate subject is a distinct option.

RQ3: Missing but crucial topics to support CS? The feedback from the SW engineers emphasized soft and practical skills more than hard and theoretical ones. For example, problem solving, and communication and negotiation skills, as well as other project managerial skills are valued high. In real projects, a freshman SW engineer faces the challenge of capturing the real needs and expectations of a customer and being capable of verbalizing them as unambiguous textual specifications and a design for an implementation team. For instance, USCC defines HS Modeling for structuring data, and the area could be subset age-appropriately for the elementary level. These skills constitute a substantial part of what we refer to as 'specificational thinking'.

However, mathematics is by far the only subject to practice specificational thinking. In essence, with such cross-curricula skills as good oral communication skills, appropriate observation and interview techniques, critical and analytical thinking that imply logical thinking, and conceptual modeling skills, students are more likely to excel in specificational thinking. Thus, achieving the goal should be a combined effort of the subjects of native language, mathematics, and why not social studies and philosophy, if available.

Further studies The emphasis shift from continuous to discrete mathematics must be executed in an evidence-based manner, i.e., the learning outcomes must be carefully evaluated in co-operation 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 mathematics in higher education, traditional '*Advanced Engineering Calculus*' would need its discrete mathematics counterparts, say, '*Programmers' Introduction to Automata and Formal Languages*' or '*Set Theory for Software Engineers*', which indisputably explicate the benefits of the renewed mathematics syllabus for the good of CS.

7 ACKNOWLEDGMENTS

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

References

1. Aamulehti: Pyörätuolissa istuvaa miestä nöyryytettiin Prisman kassalla – 'Olemme sydämestämme pahoillamme' (2017)
2. Aarresaari net: The ingredients of building a successful career in year of 2016 (2017)
3. ACM&IEEE: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, December 20, 2013. Tech. rep. (2013), https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf
4. Ardis, M., Budgen, D., Hislop, G., Offutt, J., Sebern, M., Visser, W.: Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Joint effort of the ACM and the IEEE-Computer Society (2014)
5. Beblavỳ, M., Fabo, B., Lenearts, K.: Demand for Digital Skills in the US Labour Market: The IT Skills Pyramid. CEPS Special Report No. 154/December 2016 (2016)
6. Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., Punie, Y.: Developing Computational Thinking: Approaches and Orientations in K-12 Education. In: EdMedia: World Conference on Educational Media and Technology. pp. 13–18. Association for the Advancement of Computing in Education (AACE) (2016)
7. Bourque, P., Fairley, R.E., et al.: Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press (2014), www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf
8. Brown, N.C., Sentance, S., Crick, T., Humphreys, S.: Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education (TOCE) 14(2), 9 (2014)
9. Charette, R.N.: Why Software Fails 42.9 (2005), https://spectrum.ieee.org/computing/software/why-software-fails
10. Clements, D.H.: Linking research and curriculum development. International research in mathematics education p. 599 (2002)
11. Core Standards Organization: Mathematics Standards — Common Core State Standards Initiative (2015), http://www.corestandards.org/wp-content/uploads/Math_Standards1.pdf
12. Core Standards Organization: High School: Modeling. http://www.corestandards.org/Math/Content/HSM/ (2017), http://www.corestandards.org/Math/Content/HSM/
13. CSTA: Computer science standards (2016), https://www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf
14. Denning, P.J.: The profession of IT Beyond computational thinking. Communications of the ACM 52(6), 28–30 (2009)
15. Denning, P.J.: Remaining trouble spots with computational thinking. Communications of the ACM 60(6), 33–39 (2017)
16. Department of Education: National Curriculum in England. Key stages 3 and 4 framework document (2014)
17. Dijkstra, E.W., et al.: On the cruelty of really teaching computing science. Communications of the ACM 32(12), 1398–1404 (1989)
18. English Department for Education: National Curriculum in England: Computing programmes of study (2013)
19. Finnish National Board of Education: Finnish National Curriculum 2014 (2014), http://www.oph.fi/download/163777_perusopetuksen_opetussuunnitelman_perusteet_2014.pdf

20. Finnish National Board of Education: Finnish National Core Curriculum for General Upper Secondary Education (2015), http://www.oph.fi/download/172124_lukion_opetussuunnitelman_perusteet_2015.pdf
21. Futschek, G., Moschitz, J.: 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) pp. 1–10 (2010)
22. GCSE: 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 (2015)
23. Harris, M.: The STEM shortage paradox. Physics World 27(10), 56 (2014)
24. Kitchenham, B., Budgen, D., Brereton, P., Woodall, P.: An investigation of software engineering curricula. Journal of Systems and Software 74(3), 325–335 (2005)
25. Knuth, D.E.: Algorithmic thinking and mathematical thinking. The American Mathematical Monthly 92(3), 170–181 (1985)
26. Lamagna, E.A.: Algorithmic thinking unplugged. Journal of Computing Sciences in Colleges 30(6), 45–52 (2015)
27. Lethbridge, T.C.: What knowledge is important to a software professional? IEEE Computer 33(5), 44–50 (2000)
28. Liukas, L.: Hello Ruby (2015), a childrens' book available in 22 languages
29. McGowen, M., DeMarois, P., Tall, D.: Using the function machine as a cognitive root. (2000)
30. Meziane, F., Vadera, S.: 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. pp. 65–70. IEEE Computer Society (2004)
31. Niemelä, P., Valmari, A.: Elementary math to close the digital skills gap. In: CSEDU 2018 Conference. vol. 10 (2018)
32. Papert, S.: Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc. (1980)
33. Parnas, D.L.: Software aspects of strategic defense systems. Commun. ACM 28(12), 1326–1335 (1985)
34. Parnas, D.L.: Software engineering programs are not computer science programs. IEEE Software 16(6), 19–30 (1999)
35. Peng, G.: Do computer skills affect worker employment? An empirical study from CPS surveys. Computers in Human Behavior 74, 26–34 (2017)
36. Piaget, J.: Piaget's theory of cognitive development. Childhood cognitive development: The essential readings pp. 33–47 (2000)
37. Pinar, W.F.: What is curriculum theory? Routledge (2012)
38. Puhakka, A., Ala-Mutka, K.: Survey on the knowledge and education needs of Finnish software professionals. Tampere University of Technology, Department of Software Systems (2009)
39. Redecker, C., Punie, Y.: European Framework for the Digital Competence of Educators: DigCompEdu. EUR - Scientific and Technical Research Reports, The European Commission's science and knowledge service (2017), http://publications.jrc.ec.europa.eu/repository/bitstream/JRC107466/pdf_digcomedu_a4_final.pdf
40. Smith, E., White, P.: 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 (2017)
41. Surakka, S.: What subjects and skills are important for software developers? Communications of the ACM 50(1), 73–78 (2007)
42. Taub, R., Armoni, M., Ben-Ari, M.: CS unplugged and middle-school students' views, attitudes, and intentions regarding CS. ACM Transactions on Computing Education (TOCE) 12(2), 8 (2012)
43. TEK: Tek graduate survey 2017 (2017), https://www.slideshare.net/ArttuPiri/tek-graduate-survey2017results
44. Valmari, A., Kaarakka, T.: MathCheck: a tool for checking math solutions in detail. In: SEFI 2016 Annual Conference Proceedings. pp. VK.1–VK.9. European Society for Engineering Education (2016)

45. Wilkie, K.J., Clarke, D.M.: Developing students' functional thinking in algebra through different visualisations of a growing pattern's structure. Mathematics Education Research Journal 28(2) (2016)
46. Yackel, E., Cobb, P.: Sociomathematical norms, argumentation, and autonomy in mathematics. Journal for research in mathematics education pp. 458–477 (1996)