Computational Thinking as an Emergent Learning Trajectory of Mathematics

Pia Niemelä
Pervasive Computing
Tampere University of Technology
Finland

Tiina Partanen
City of Tampere
Finland

Maarit Harsu
Pervasive Computing
Tampere University of Technology
Finland

Leo Leppänen
Computer Science
University of Helsinki
Finland

Petri Ihantola
Pervasive Computing
Tampere University of Technology
Finland

ABSTRACT

In the 21st century, the skills of computational thinking complement those of traditional math teaching. In order to gain the knowledge required to teach these skills, a cohort of math teachers participated in an in-service training scheme conducted as a massive open online course (MOOC). This paper analyses the success of this training scheme and uses the results of the study to focus on the skills of computational thinking, and to explore how math teachers expect to integrate computing into the K-12 math syllabus. The coursework and feedback from the MOOC course indicate that they readily associate computational thinking with problem solving in math. In addition, some of the teachers are inspired by the new opportunities to be creative in their teaching. However, the set of programming concepts they refer to in their essays is insubstantial and unfocused, so these concepts are consolidated here to form a hypothetical learning trajectory for computational thinking.

CCS CONCEPTS

• **Social and professional topics → Computational thinking**; *Computing education*; Employment issues;

KEYWORDS

Computational Thinking, Learning Trajectory, K-12 Computer Science Curriculum, Math-integrated Computing, In-Service Teacher Training

ACM Reference Format:
Pia Niemelä, Tiina Partanen, Maarit Harsu, Leo Leppänen, and Petri Ihantola. 2017. Computational Thinking as an Emergent Learning Trajectory of Mathematics. In *Proceedings of Koli Calling 2017, Koli, Finland, November 16–19, 2017,* 10 pages.
https://doi.org/10.1145/3141880.3141885

Koli Calling 2017, November 16–19, 2017, Koli, Finland

ACM ISBN 978-1-4503-5301-4/17/11…$15.00
https://doi.org/10.1145/3141880.3141885

1 INTRODUCTION

The rapid digitalization of society and the demand for a technologically fluent workforce for the 21st century means that our education system has had to adapt. Computational thinking (CT) skills comprise a significant portion of the new qualities that make up the resulting updated K-12 curriculum. Curricula, syllabi and learning trajectories are the essential components in making computational thinking accessible. The Finnish National Curriculum was modified in 2014 to include algorithmic thinking (a subset of computational thinking) and computing as the emergent parts of the math syllabus [13]. These changes were first introduced at the primary level and have been in effect since autumn 2016. However, exactly how computational thinking should be taught has still not been clearly defined, which has created an arena for various learning experiments, further research and speculation.

Educators need to agree on a clear theoretical perspective in order to establish the evaluation criteria for computational thinking. In addition, math teachers need to review the computing skills that they now require in order to implement CT in their courses. In order to respond to this need, in the autumn of 2015, a group of volunteer teachers informally launched the Code ABC MOOC with several tracks, one of which is the Racket track examined here. The Code ABC MOOC is aimed at providing teachers with the CT skills required by the new curriculum. In addition to introducing the basics of computing, it emphasizes creativity and the ability of teachers to integrate computing into their math lessons in a pedagogically justified manner.

The additions to the curriculum can be divided into two complementary parts: the basics of computing and computational thinking. In this article, we examine the views of the Racket MOOC participants by analyzing their essays (N=206). In this analysis, we focus on computational thinking and how the teachers expect to apply it in their teaching. The ideas and proposals in their essays are combined to form a learning trajectory for math that extends into the area of computational thinking. The main emphasis in this work is not on the basics of computing, but on how computational thinking is interwoven into teaching math. In the analysis, we focus on computational thinking and how the teachers expect to apply it in their teaching. The aim is to sketch out as smooth a learning trajectory as possible by streamlining the transfer between math and computing. More precisely, we seek to answer the following research questions:

- How do the teachers define computational thinking?
- How do they integrate computing with math?
- What kind of a learning trajectory for computational thinking can be constructed from the teachers' essays?

This article proceeds as follows. Section 2 reviews published work on computational thinking (CT) and learning trajectories (LT). Section 3 describes the research method. Section 4 provides the results: the teachers' views on both CT and computing are represented and generalized as a new enhanced LT of math that expands into the area of CT. Section 5 gives conclusions.

2 RELATED WORK

2.1 Definitions and models of CT

CT has emerged as a consequence of the increased prominence of computing as a new school subject. In particular, it refers to the skills that programmers need in their work. Wing introduced the term CT in 2006 in her seminal article [38]. Although there is still no absolute consensus on the definition of CT, most experts accept Wing's later description from 2010, that CT is, "The thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be carried out by an information-processing agent" [39]. Attempts to define what exactly constitutes CT can be traced back to 1996, when Papert *stated that, "Computer science develops students' computational and critical thinking skills and shows them how to create, not simply use, new technologies. This fundamental knowledge is needed to prepare students for the 21st century, regardless of their ultimate field of study or occupation"* [29]. Papert's observation that CT is a creative skill underpins much of the now accepted definition of the discipline.

The commonly accepted cornerstones of computational thinking include: data collection, data analysis and data representation, problem decomposition, abstraction, algorithms, automation, parallel code and simulation, as defined by Barr and Stephenson [5]. This model defines three classes for data, thus emphasizing its importance. In addition, it should be noted that parallel code and simulation are not commensurate with abstraction and automation, as the former tend to be more concerned with the implementation specifics.

Although a number of models enumerating the contents of CT have been proposed, see e.g. [5, 9, 36], in our opinion, the model introduced by Cuny, Snyder and Wing [10] encompasses all the essential components of CT and nothing superfluous, and it still has enough resolution power to categorize the Racket MOOC participants' views. It is capable of covering most of the teachers' CT characterizations under the following three categories:

- abstractions (e.g. pattern generalizations, symbol systems and representations, and structured problem decomposition e.g. as functions) that indicate the design-orientedness of a participant;
- automation (the control flow realized with the help of control structures and information processing); and,
- analysis (e.g. debugging and systematic error detection, optimizing performance and efficiency)

2.2 Integrating computing into K-12 curricula

A significant number of European countries have recently introduced computing as a new addition to their K-12 curricula [3, 16]. Although most of these countries have introduced computing as a separate subject, Finland has chosen to integrate CT into the curriculum mainly with math and crafts, see Table 1. Math provides a theoretical basis for the concept, while crafts gives the pupils an opportunity to apply their newly-learned skills by creating digital artifacts, such as robots. Compared with computing, math has a well-established learning trajectory that has endured the test of time, and has survived a number of regenerations, such as that inspired by the New Math movement [21]. *In Finland, the teaching of Craft has developed along with changes in technology, and has long included computing as one of its components.* Here, we aim to examine how best to exploit this synergy between the two topics.

Integrating computing with math is not risk-free. A recent OECD study [26] concluded that the more technology was merged with the math syllabus, the poorer were the results. Nevertheless, Hemmendinger [17] reminds us that algorithmic thinking is not anything new: the origin of the term "algorithm" lies in 12th-century Persia. Similarly, Tedre and Denning [37] states that the history of CT can easily be traced back to the 1950s. However, rather than enumerating the many advantages of CT, these authors prefer to explore the results of previous learning experiments with the subject, in order to avoid repeating the same mistakes again and again. Indeed, they question the transferability of algorithmic thinking, which has hardly ever been integrated successfully into other subjects, despite high expectations.

We proceed under the assumption that integrating computing into math will inevitably move the center of gravity of the math syllabus towards CT, but that this will merely strengthen the existing link between math and computing. Along with adapting appropriate thinking patterns, CT also requires a student to learn the necessary computing skills. Conceptually, the transfer between math and computing fits best with with the functional programming paradigm. In particular, it is claimed that learning the functions of algebra is easiest with functional languages [24, 35].

Math-integrated computing has a remarkably long history with the functional programming paradigm, starting with the LOGO learning environment [14, 23, 28], and continuing with the recent Racket and Haskell experiments [2]. Although it has been argued that Haskell has some pedagogical advantages over Racket, such as strong typing and symbolic notation closer to math, the Racket camp in the USA has consistently reported good, stable results [11, 12, 34, 35]. The successful experiments with Racket have focused on the transfer between computing and algebra, whereas the results with the LOGO experiments are harder to pin down [23].

Felleisen and Krishnamurthi [12] propose the paradigm of imaginative programming, by which they mean inventive exploitation of the media (image) rich Racket programming language. In contrast to other popular functional languages, Racket supports images as first-class values, which means that they can be inserted into text and manipulated in a similar fashion as numbers, e.g. in DrRacket editor. The authors note, however, that integrating computing into other subjects is fraught with difficulties, and they emphasise that the programming language should be as close to the language and

	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

Table 1: Computing-related additions to the Finnish Curriculum, 2014. (Typically a student is 6–7 years old, when starting Year 1.)

concepts of the school math syllabus as possible. This complies with the near transfer principle, which states that the more similar the topics are, the easier is the learning [32].

2.3 Co-constructing LT

Learning trajectories (LT) have made an important contribution to curriculum development and research. They are a part of a larger theoretical framework referred to as hierarchic interactionalism [33] , which synthesizes aspects of both Piagetian constructivism and Vygotsky's Zone of Proximal Development. The theory states that children actively and iteratively construct knowledge that is ordered as "hierarchic constructs", or mental structures. Although originally concerned with early education, hierarchic learning can also be applied to adult education, especially in such cases where any previous learning experiences are missing. For such adult learning developments, hierarchic interactionalism introduces the concept of non-genetic levels of cognitive development, in contrast to the traditional genetic levels of cognitive development ascribed to infants [8].

To ensure the smooth integration of CT, a well-grounded LT should determine consistent progress in the same way that the more established math syllabus does. In the context of computing and CT, the cohort of teachers in this Racket MOOC study have enough computing experience and understanding to reflect on what they have learned. It is their reflections on their experience of Racket MOOC that are elaborated on here in order to construct a hypothetical Learning Trajectory for the development of CT in the Finnish school curriculum. In this study, the test subjects (professional math teachers) are, on the whole, older than the participants in many other LT studies. According to Piagetian genetic epistemology, they are well above the age at which children begin handling formal operations, i.e. twelve and above [31].

Although adults can think more abstractly than children, the Piagetian cycles still apply to adult learning, even though some sensory-motoric cycles may be quicker, while others may have ceased to exist. In this particular study, it is also anticipated that the transfer will influence the learning: the closer the subjects, the easier is the transfer, and this seems to be true of the transfer between math and computing. However, there is little doubt that adult learners face different challenges than elementary school students. For instance, the brain's plasticity slows down in adulthood, which affects learning. In addition, although it may sound counter-intuitive, an adult learner's gained expertise may not always be an advantage, as a way of thinking that has become too entrenched can pose problems for the adult. As [4] points out, entrenched and therefore less sensitive mental structures may result in possible error signals failing to induce direct changes in the mental system.

On the other hand, as experts in both the pedagogy and substance of teaching math, math teachers are able to utilise a variety of strategies for efficient learning. A meaningful instructional set-up and well-justified LT facilitate explicit abstraction and transfer between prior knowledge and new concepts [32]. Given the various advantages and constraints, the math teachers who are the subject of this research can be regarded as valid representatives for the ultimate target group, elementary school students.

3 METHOD

3.1 Context of the Study

Up to 540 teachers participated in the Code ABC MOOC during the research period of autumn 2015 and spring 2016 [30]. One of the authors of this article was the instructor of the Racket track. The first design principle of the MOOC was to use multiple visually interesting image/Turtle/animation exercises to enable creativity in order to appeal to elementary school students. The second design principle was to prove the applicability of computing in the context of elementary school mathematics. Math teachers need to be convinced of the benefits of adopting CT and computing into their classes without feeling that time is diverted from math studies. Therefore, the programming exercises had a multitude of mathematical concepts woven in, such as geometrical shapes, angles and measures, the coordinate system, rounding decimals, and functions to calculate percentage/price/area/volume and to solve triangle problems, for instance, by utilizing Pythagoras' theorem.

The Code ABC MOOC consisted of six programming exercises and a pedagogical essay as the last item. The details of the course content and how it was organized can be found in [30]. To complete the course, 80 % of the coursework had to be accomplished, thus only a part (38 %) of the participants (N=130 in autumn 2015, N=76 in spring 2016, total of N=206) returned the final reflective essay. In the essays, the participants reflected on the curriculum, sketched out appropriate LTs for CT, and provided many instructive ideas and lesson plans. This study applies mixed methods: the essays written by the course participants are analyzed both qualitatively and quantitatively. In the qualitative analysis, the definition of CT and linking computing with math are extracted, and the most descriptive quotations are selected to give a voice to the teachers. The quantitative analysis synthesizes the teachers' views as statistical charts and finally as the crowd-sourced LTs of CT.

The teachers' CT views were categorized into three super-classes based on the model by Cuny et al. [10]: abstraction, automation, and

analysis. In order to examine the teachers' views about abstraction, the design-orientation (measured as the amount and level of detail related to the abstractions) of each teacher was estimated on a Likert scale (1-5). The score illustrates the structuredness of the computing process as a whole. The phases of planning, documenting and testing are counted as indications of design-orientation.

The Racket MOOC applied the staircase Design Recipe for Functions model [11] , which divides programming into the following steps:

(1) think what a function is supposed to do, specify the purpose
(2) name the function descriptively, figure out needed and returned info, specify the signature
(3) write the function stub, use descriptive parameter names and set a placeholder for a return value
(4) implement and run tests (check-expect) with concrete values
(5) lastly, implement the function body

It was mandatory to successfully complete the MOOC exercises and writing unit tests with check-expect (item four above).

The teachers' compliance with this recipe was one criterion used to arrive at the Likert-scale score of design-orientedness. The occurrence frequencies of computing concepts were recorded from the content, whereas the CT related topics found in essays were grouped to fit their respective category in the CT model. The most frequent topics are visible in the dendrogram (1b), such as decomposition, problem solving and functions as identifiers of abstraction. Even if the data itself is qualitative, it is quantitatively analyzed. Mixing qualitative and quantitative approaches within or across the stages of the research process is referred to as the mixed model [20].

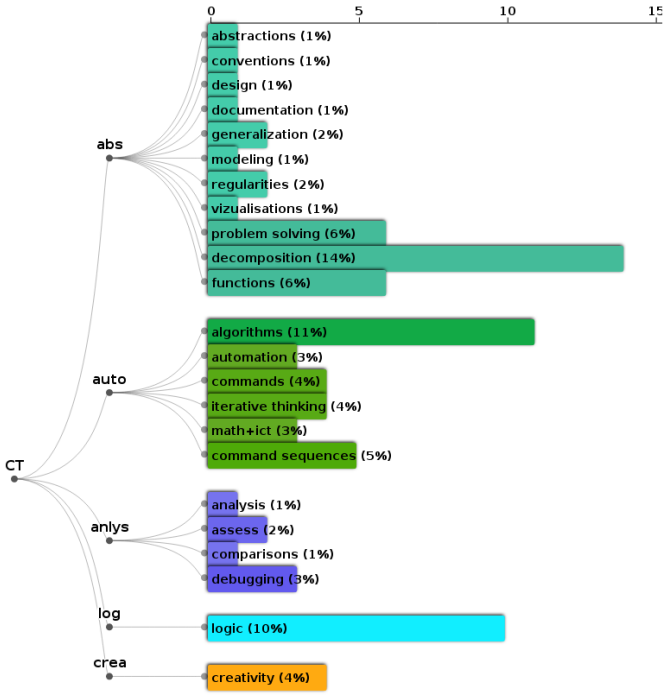
4 RESULTS AND DISCUSSION

This section introduces the results based on the pedagogical essays written by the teachers. In trying to integrate CT with math, the teachers were particularly concerned with the pedagogical viewpoints. We will examine how they perceive CT and decompose it as the general capability needed in programming. After the CT results, the affordances of those parts of the math syllabus which are most conducive to computing are investigated in more detail, as the math teachers describe which math areas, in their opinion, best suited for computational interventions. To make the results more generalisable, the teachers' views are combined into one crowd-sourced, math-integrated LT for CT.

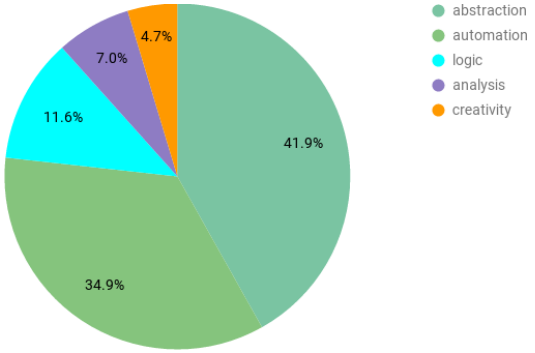
4.1 Components of the CT model

Overwhelmingly, the teachers showed that they had internalized the concept of CT, see Figure 1. All the needed components, abstractions (41,9 %), automation (34,9 %), and analysis (7,0 %), were present in proportion to their share in the MOOC content. In addition to the main components of the CT model, teachers emphasized such qualities as logic and creativity. Figure 1a lists the sub-items of each CT area with their percentages. The following subsections will illustrate the teachers' views with selected quotes.

4.1.1 Abstraction. The teachers described abstraction as: making generalizations and finding regularities; being able to make abstractions, design and model systems; writing documentation,



(a) CT categories decomposed



(b) Percentages of categories

Figure 1: CT key areas by math teachers as a decomposed dendrogram (a), and a pie chart (b). Percentages illustrate the relative frequencies of the concepts in the essays; however, values less than 1 % are omitted.

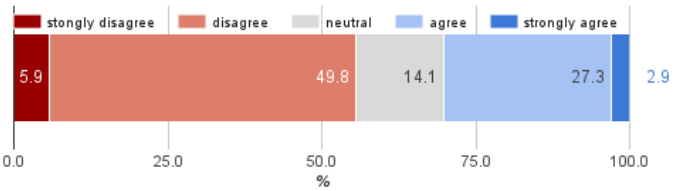


Figure 2: Computing should be taught by concentrating more on theory, concepts and design than creative hands-on experiments (N=206)

and following good coding conventions. While contemplating various aspects of CT, teachers reflected on the advantages of good problem solving skills in general, as the following response shows: *Highlighting problem solving skills is a welcome addition in any subject. Everybody benefits from decomposing problems into subproblems and solving them step-by-step.* In computing, like in math, problem-solving starts with decomposing the problem into smaller tasks, i.e., functions.

4.1.2 Automation. Within the automation category, the teachers regarded algorithmic thinking as the most important CT skill. Furthermore, designed functions need to be sequenced into separate commands. During function implementation, a student must employ iterations, conditional logic, and all the other syntactic means in order to accomplish the task. So, in addition to basic syntax, the control structures must have been internalized as well. *Algorithmic thinking produces such routines that facilitate and speed up our everyday actions.* Natural language provides an efficient tool in problem decomposition and deeper understanding: *In my teaching, I emphasize the path to the solution; the plain answer is nothing in lieu of intermediate steps to the solution and assessing the soundness of the answer. Computing supports the development of algorithmic thinking, which justifies its inclusion in the curriculum of the elementary school.*

4.1.3 Analysis. After the design and implementation, it is time to evaluate achievements. In math, the evaluation phase means e.g. ensuring that the result of a calculation is reasonable. In computing, the program must pass tests. If not, the functionality will be debugged and errors fixed until the tests are passed; as one teacher puts it: *Debugging separates the wheat from the chaff.*

At a more sophisticated level, the analysis covers aspects of efficiency and resource usage. The bottlenecks of execution may be determined by profiling the code. In algorithm development, the benchmarking of speed, for instance, enables comparisons of different solutions. From the angle of project management, this is the phase during which the quality of the product is assessed, i.e., whether a client is happy and there is completion of definition-of-done requirements.

4.1.4 Logic. Logic was mentioned the most frequently out of the uncategorized responses. In this context, logic is understood both as the skill of handling conditions and their truth values in iterations and selections, and as logical thinking skills. These skills comprise the clarity of abstractions, problem solving, seeing common patterns, and proceeding consistently step-by-step.

4.1.5 Creativity. Conceptions of teaching computing on the axis of creativity-vs-design-orientation varied remarkably, although on the whole they were more creativity-weighted, see Figure 2. The conceptions range from one extreme of seeing creativity in all computing phases to observing no creativity at all. For example: *I think computing is not creative at all! Not adhering strictly to the rules will be penalized. Creativity can not be taught by programming. Teaching programming may be reduced to merely teaching the theory.* Self-evidently, highlighting the design phase illustrates design-orientation: *It is crucial to learn the importance of planning. It is important that a student will be able to think about the program and its functionality even without knowing how to code. Thus, I consider design as the most important skill. Once the design is clear, it is easy to implement the program.*

The MOOC course emphasized planning functions beforehand and including unit tests and documentation as a part of the process. At the beginning, the need for documentation was questioned: *While coding, documentation seemed very stupid: of course you know what you are currently doing. Still afterwards, when writing more code, written comments started to feel precious. In addition, the proper naming of functions helped understanding.*

Some teachers favored experimental learning, expressing themselves as follows: *Playing and experimenting is well suited for learning programming. There is not only one correct way to solve the problem with code. Let us try, dare to fail, tolerate uncertainty and finally experience the joy of success, when the code works as expected.* And: *I enjoy such tasks the most that allow playing and experimenting. When starting with a completely new group, I would teach this way, not so much going through the pile of different concepts.* And one comment, where Dewey's view is well internalized: *Learning by doing!, Programming is 90 % creativity, 10 % theory.* In the middle of the creativity-vs-design continuum, we encountered opinions, such as: *creativity and theory, they go hand in hand*; *once basis and commands are clear and internalized, experiments / play are needed*; and *the lack of theoretical knowledge limits creativity.*

Some participants noted the two-sided nature of creativity: *In computing, creativity does not manifest itself in such richness that we are used to. On the contrary, finding the shortest and the most optimized way of writing code demonstrates creativity.* This teacher broadens the definition of creativity even further, that is, being able to prepare for faulty input and to step out of the current situation and anticipate easy maintenance in future: *Creativity is that your code works even if a user gives a faulty input. Moreover, creativity is writing such easy-to-read code that a person who modifies it gets the idea with ease.* Even though this teacher is capable of combining creativity with design-orientedness, the majority of the teachers echoed the opinion quoted at the start of this section, which contrasts creativity with design.

Another teacher became particularly inspired with the open-ended nature of programming tasks, and the opportunity to be creative: *Here is my owl. I wanted to include it here, because while doing it I was inspired like a child. The whole world of coding, its opportunities and creativity opened to me. I was capable of doing this and the result was unique!* Being creative equals tinkering, the philosophy behind which has also been referred to as having 'a maker mindset'.

4.1.6 Complemented CT Model. Figure 3 merges the CT model components of Cuny et al. [10] that were unambiguously present in the teachers' replies with the new CT complements of logic and creativity.

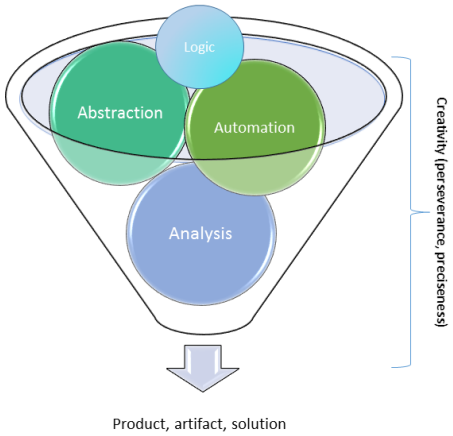


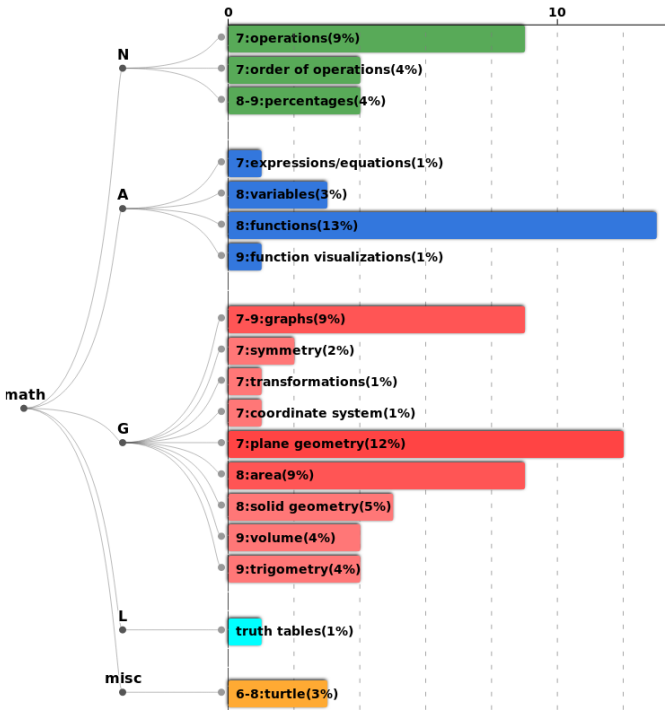
Figure 3: CT model enhanced with logic and creativity

In minor quantities, the teachers emphasize such personal characteristics as perseverance and preciseness. A number of them worry about their students' lack of motivation and perseverance regarding science-technology-engineering-maths (STEM) subjects that need hard work and an undaunted attitude in the face of difficulties. Being precise is tested, for instance, when a student is struggling with the syntax of textual programming languages, where adding a semi-colon or right indentation may do the trick. Many teachers proposed the students' own projects to prepare them for collaboration and working life. Project work necessitates paying attention to the schedule and the process in its entirety from the beginning of the design-phase to the very end of testing, documenting and finalizing the product.

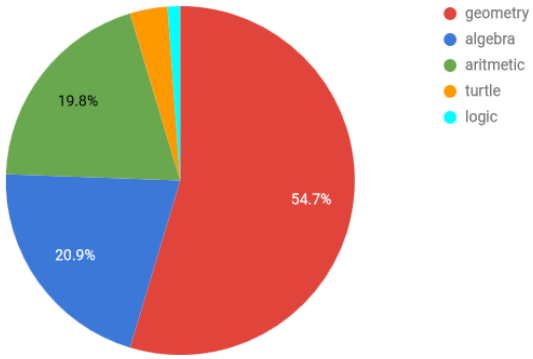
4.2 Math integration

In integrating computing into math, geometry was the most popular subject: the red slice of the pie (54.7 %) in Figure 4. The majority of the Racket MOOC participants sketched out geometry-oriented lesson plans. In addition, the teachers envisioned integrative projects with art and crafts: math-integrated computing would provide the needed design skills, which could be exploited in practice by implementing designs for posters, stencils, or 3D printing. Based on their answers, the serendipity of the outcome due to automation and iterations seemed to enthuse a number of teachers. In addition, Racket's capability of handling images as first-class values facilitates the programming of graphs and images with ease.

The prominence of geometry is still surprising, as concept-wise it is not central. It may rather be interpreted as an area where a student can apply computing skills. For example: a programmer may visualize both plane and solid geometric shapes and calculate their areas and volumes. Even though Turtle is not part of any specific math syllabus area, the teachers frequently mention it. Turtle is a movable figure that can be used as a drawing tool. For its part, Turtle scaffolds forthcoming steps of visualizations in geometry and functions of algebra, and fosters CT. It might also turn into a precursor to computing as one teacher points out - her students consider computing as *guiding some dude along a certain route.*



(a) Math categories decomposed



(b) Percentages of categories

Figure 4: Syllabus areas fit for computing (N=206). The percentages are based on the relative frequency of exercise proposals of the teachers. Percentages illustrate the relative frequencies of the math syllabus areas connected to the exercises. Values less than 1 % were omitted.

Figure 4 shows the most popular syllabus areas fit for computing: geometry, algebra, arithmetic, and logic. Algebra and arithmetic got clearly fewer votes even though they are more fundamental theory-wise in understanding programming basics: chronological and consistent progressing necessitates devising basic operations and the order of arithmetic operations before expressions and equations, followed consistently by algebraic fundamentals, variables and functions. In computing, statements are divided into the primitive assignments of variables, and function calls, which requires familiarity with these two fundamentals.

4.3 The learning trajectory of computational thinking

This section outlines the crowd-sourced LT as a means to generalize the teachers' views on CT. We merge the exercise proposals and syllabus ideas of the teachers' essays as LTs grouped under the corresponding syllabus areas. The majority of the proposals were highly compliant with the Finnish National Curriculum 2014, which forms the skeletal LT that is to be determined more in detail with exercise proposals and by linking selected computing concepts to corresponding math concepts.

According to the teachers, computing should be started already in primary school (Years 1–6) with a graphical environment, such as Scratch. Turtle is regarded as a good intermediate tool for bridging the gap between Scratch and textual programming, such as Racket. In addition, Turtle facilitates breaking the task down into smaller sub-tasks, for example, when constructing figures from simpler shapes. This is a kick-start to decomposing problems into smaller parts, hence it is good preparation for programming.

In the school grades (Years 7–9), students should preferably continue with textual programming. In Year 7, a student must learn how to execute basic mathematical operations. In algebra, expressions and equations support this topic as well, and built-in functions of the computational system demonstrate how to exploit functions. These calculations can be executed in the prompt as simple command line commands, so it is not necessary to write an actual program in this phase. In geometry, however, a student could start exercises in drawing various geometrical shapes. In order to modify and demonstrate the achievements, the results may be saved as programs. The teachers sketched the following examples:

- Turtle for examining shapes, angles, symmetry and mirroring that belong to the wider domain of transformations
- programming formulas
- quizzes for e.g. identifying geometric shapes

The teachers anticipate an easier engagement with visually appealing computer graphics than with calculations. In addition to static geometric exercises, the MOOC rehearsed animations as a dynamic extension. However, the animation exercises were not frequently referred to in the essays.

In Year 8, students start with percentages. These calculations are fit for functions, such as calculating reductions in prices. The algebraic fundamentals, variable and function, are introduced in this phase. In geometry, these algebraic fundamentals are exploited by defining functions for area and volume. The side length of a quadrangle implemented as a function parameter would enable easy experimenting. After plane geometry, drawings continue with the more advanced 3D shapes of cube, cone and cylinder. The teachers' exercises covered the following topics:

- equations and inequalities, formulas for e.g. percentages, areas, and other STEM subjects as well, in particular physics
- (simple) calculator application
- drawing plane and solid geometry shapes

In Year 9, percentages continue further and functions are visualized as graphs, which facilitates analyzing their behavior, such as finding solutions, and minima and maxima. In analyzing the data, visualization in general could be used in math and STEM. In this phase, the teachers were willing to gradually move to more complex tasks and to give more freedom to the students in topic selection:

- functions and simultaneous equations, solving and analyzing behavior
- problem solving, being able to decompose a bigger task into smaller functions
- own projects, learning to take responsibility

The teachers had mature and instructive opinions on how to apply CT to typical problem-solving in math. Practices such as problem decomposition, finding the optimal solution, analyzing the end result and representing the solution to others by verbalizing the phases, were categorised as CT. However, when moving on to actual computing, the teachers' views were more rudimentary, often being rather shallow in concept and concerned with minor details rather than striving for the bigger picture. Although most teachers were familiar with the computing requirements of the Finnish National Curriculum 2014, and tried to elaborate on them further to fill the gaps, there were surprisingly few totally original suggestions.

In addition to the National Curriculum requirements, the CS syllabus also covers the majority of computing fundamentals such as variables, functions, and statements, although type was rarely mentioned in the essays. The absence of type also reflects the MOOC content which is based on Racket's implicit typing. A couple of the more experienced computing teachers listed variables, function, selection and iteration as the target concepts, which, as a proper subset of gathered CS1 fundamentals, implies that consensus concepts might be found quite effortlessly. Table 2 shows that each of the syllabus areas received several exercise proposals.

The math teachers are remarkably faithful to the Finnish National Curriculum in following its guidelines and schedule. Hence, the curriculum sets the basis for the learning trajectories of each syllabus area. However, theory-wise only a few of these areas are closely linked to computing fundamentals. Figure 5 visualizes the connections between computing concepts extracted from the essays and the respective areas in the math syllabus. The upper part of Figure 5 depicts the LT of mathematics in Years 1–2, Years 3–6, and Years 7–9, where the solid arrows illustrate prerequisite relationships of math concepts.

The lower part of Figure 5 shows the necessary computing concepts and their prerequisite relationships. Computing concepts are clearly separated to avoid confusion. The concepts extracted from in the teachers' essays were validated against the basic computing concepts in Section 4.4. The concepts divide into abstraction, automation, and analysis. This grouping complies with the CT model explained in Section 4.1. We have not outlined the exact schedule for teaching these concepts. However, the dashed lines in Figure 5 extend LT into the area of CT, thus implying the timing to be followed provided that the corresponding concepts have been introduced in sync.

Type and data structure belong to abstraction because they refer to abstract data types. Even integers can be considered abstract, as their implementation is hidden. Variables are abstractions of real world items. Functions can be seen as command abstractions. As an abstraction tool, Design Recipe by Felleisen et al. facilitates

the planning of well-designed functions [11]. Recall from the list at the top of Page 2 that automation contains control flow, as the automation nodes of the Figure. Our analysis illustrates that the reflective part of the process complies with the test-driven emphasis of the Racket MOOC.

Concepts of geometry do not link to fundamental computing concepts (e.g. variable, function, and type) in the CT box below. Thus, geometry-related exercises do not limit or constrain the CT teaching schedule. However, various topics in geometry provide suitable applications to practice programming and, in particular, its automation role with Turtle and computer graphics. If affective aspects of learning are emphasized, these exercises seem to inspire a number of MOOC participants.

4.4 Validity considerations

In qualitative research, data, method and researcher triangulation are the main means of improving validity [22]. Although this article is based only on the data of essays, previous work which also utilized survey data produced similar results to the findings here. The mixed research model exploits both qualitative and quantitative phases: qualitative information is first coded or occurrences are counted, after which the data is quantitatively handled. Researcher triangulation would have improved the quality of categorizing of the CT components and coding of creativity vs. design-orientedness in Chapter 4.1.5. However, due to time pressures, only one researcher was available to read, categorize and code the essays.

Overall, the taught topics taught in the MOOC were reflected in the teachers' essays, which is to be expected. Thus, the extracted concepts do not spring from a vacuum, but are an echo of the course content. For example, algorithmic thinking was in focus instead of computational thinking, because of the wording of the Finnish National Curriculum. This may partly explain, why the concept of algorithm was so central (11%), see Figure 1b.

In order to ensure the validity of the concepts in the depicted LT, the teachers' concepts were compared with the concepts retrieved from other sources that define the central concepts at the higher education level. In the university course "Principles of Programming Languages", Harsu [15] rationalized the consistent approach of introducing the fundamental concepts. The priority of certain computing fundamentals was clear:

- Functions together with variables are the most essential concepts.
- Variables and function parameters may define a type. Data structures (e.g. containers: arrays, lists), i.e. advanced types, are elementary in e.g. search and sort algorithms, or more generally in filtering or accumulating the data
- Managing the control flow with selection and iteration provides the rest of the means for successful computing

The analysis of the first computer science courses (CS1) of Finnish universities and ACM computer science course requirements [1] gives a statistically-based rationale for opting for these very same concepts. The only exception is the prominence of the concept "algorithm". In frequency, it is comparable with the fundamentals of function and variable. In general, algorithms and data structures are of a significant importance [1][e.g. ACM-SDF, ACM-AL]. Here, the central role of data structures highlights the prominence of type concept. In contrast, type was not in focused on in the teachers' essays. Selected language and paradigm also warrants its own nuances for the concept set. E.g. if object-oriented, then object and class are among the top ten, but in the case of functional paradigms, recursion and higher-order functions become more important.

Software-engineering-wise, implanting a well-structured process of design-implementation-testing (the order is not fixed, as e.g. in test-driven development) as well as highlighting good coding conventions, such as modularity and appropriate naming, were also considered topical right from the beginning in Finnish CS1 courses.

5 CONCLUSIONS

How do the teachers define CT? When the teachers considered the skills and concepts that are the most important in learning computational thinking in Years 7–9, they mentioned topics that fit the categories of abstraction, automation, and analysis. In automation, algorithms were highlighted in particular. In addition, logic and creativity were frequently quoted; logic both as the competence of thinking consistently, and solving the truth values of conditions. Regarding the MOOC content, the CT part was especially well internalized, which is natural, since practices analogous to CT are applied in problem solving throughout the elementary school math syllabus.

How do they integrate computing with math?

The teachers regarded geometry as the syllabus area with the most potential due to options for creativity. Geometry was favored at the expense of the more conceptually-adjusted area of algebra (function, variable) and arithmetic (basic operations, the right order, condition primers). The visually educational, showy and sometimes serendipitous outcomes in geometry are found to be appealing. Controversially, a few teachers considered any math integration to be problematic in itself. Their reasoning was that math as a school subject has a reputation of being a hard subject, and its reputation for difficulty may readily taint any introduction to computing as well. This attitude was exemplified by the following quotation: *Current youth have no interest in math because of too much work (and complexity). Hence, first programming experiences should be as remote to math as possible.*

What kind of LT for CT can be depicted?

Our hypothetical LT, based on the MOOC participants' essays, is well rounded and contains all the essential fundamentals. In particular, variable and function were emphasized, although it must be recognised that type was hardly mentioned. The most common control structures, selection and iteration, were also well represented. However, higher-order functions and recursion as an emphasized iteration method of a functional paradigm were regarded as being significantly more complex and were thus seen as candidates for differentiation. The LT will give a consistent and solid base for assessing progress in CT and computing. However, in order to help teachers discern the similarities and differences between math and computing and in order to boost their confidence, it is clear that they need more in-service training and reinforcement of their knowledge of the theoretical basis of computing. Some of the most fundamental concepts in these two disciplines differ quite dramatically, as is the case for the concept of variable, for instance. A variable in computer science has a very complex nature

Table 2: Computing exercises that the teachers integrated in the math syllabus

Year	Area	Exercises for computational thinking and basic programming concepts
Y1–6	all	"unplugged" exercises, following instructions, hands-on experiments in graphical environment
Y7	N	basic operations, order of calculations
	A	expressions, equations
	G	drawing 2D shapes of plane geometry (triangle, square, circle), practising angles
Y8	N	percentages
	A	variables and functions
	G	calculating areas of basic shapes, Pythagoras, circle
Y9	N	percentages cont.
	A	visualizing and analyzing function behavior
	G	volume calculations, trigonometry, 3D shapes of solid geometry (cube, cone, cylinder)
Y7–9	L	logical thinking, Boolean values and operators, truth tables
Y6–8	Turtle	creative exercises related mainly to geometry

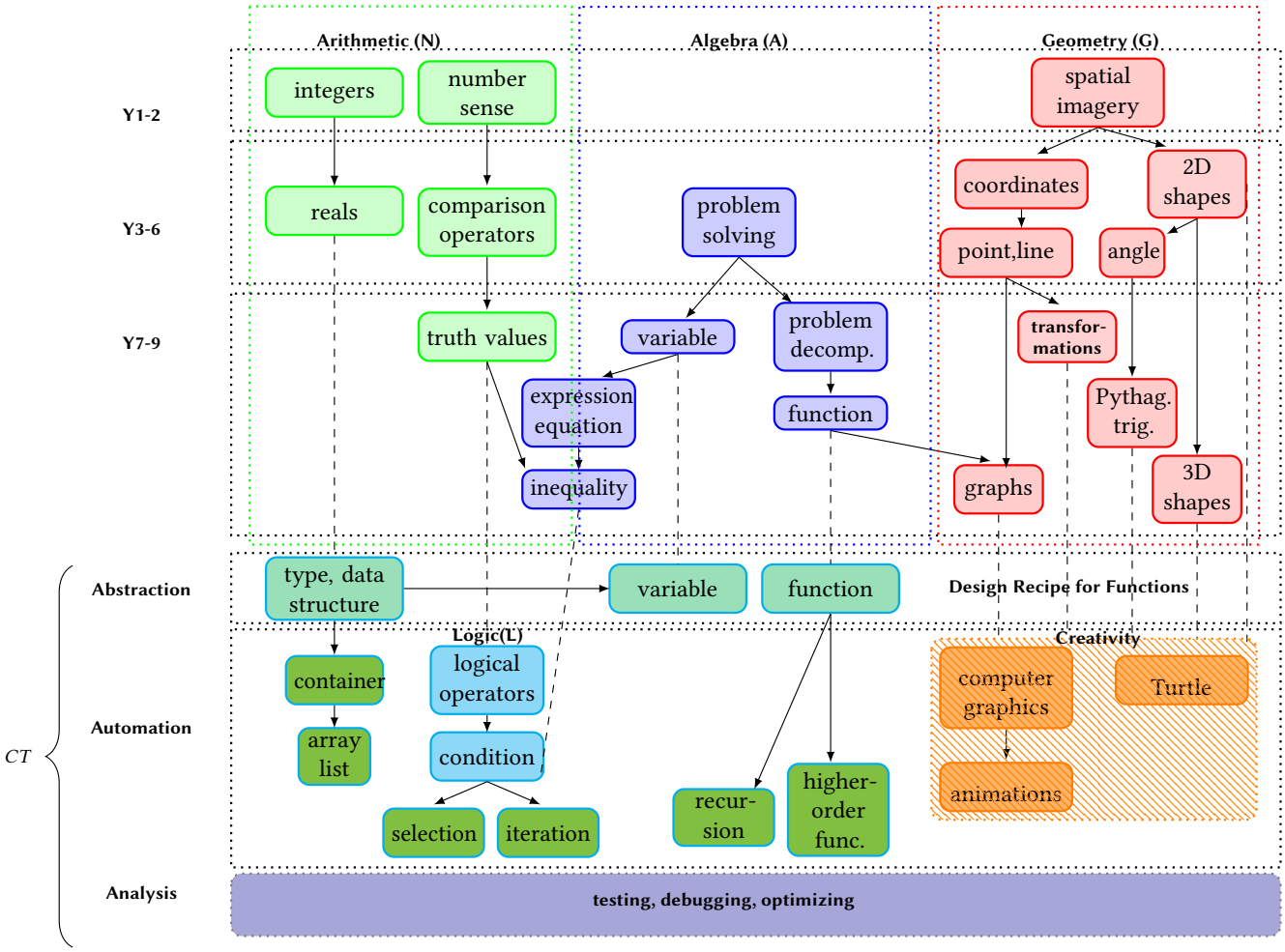


Figure 5: Hypothetical learning trajectories of CT

compared with its simplicity in math, being an entity of at least a name, value, type, location in the memory, scope and life time. The same applies to functions, e.g. the function in math outputs is always the same value for the same input, but this is not necessarily the case in computing, cf side-effects. If ignored, these fundamental differences can easily lead to misconceptions. However, at present it seems that probably only a few math teachers are aware of such details.

As a part of a wider range of thinking skills, CT emerges out of a reciprocal relationship between math and computing. Correspondingly, the math teachers easily transferred their problem-solving procedures to form a basis for CT. In addition, they were capable of sketching a number of exercise proposals even though they were missing some fundamental CS concepts. The math teachers' prior knowledge maps well with CT, although computing basics need more emphasis. However functional the linkage between math and computing might be, the curriculum should still reserve space for, e.g., philosophy, language, and art as alternate angles of approach to CT, and thinking skills in general.

Industry and educators have requested better CS-equipped students to fulfill the need of the future workforce [6, 7, 18, 19, 25, 27]. As an emergent new subject, computing provides novel opportunities to outfit future students with the required skills. In constructing computing knowledge, the Finnish National Curriculum needs further elaboration, since the 2014 version only gives relatively cursory guidelines for the teaching of CT. Regardless of the programming language or tools selected, the learned computational thinking skills and computing concepts should be the same for all students finishing elementary school, i.e. standardized. In refining the most crucial concepts, the Racket MOOC has made a valuable contribution towards this end. Raising the lower-end of the bar enables the learning targets at the top end of the educational bar to be raised as well, which is obviously the next step.

6 ACKNOWLEDGMENTS

Special thanks to the Finnish National Board of Education, Technology Industries of the Finland Centennial Foundation, and the Academy of Finland (grant number 303694; *Skills, education and the future of work*) for their financial support. In addition to the funders, we would like to express our gratitude to the Innokas network and Tarmo Toikkanen for co-ordinating the Code ABC MOOC, as well as to the Aalto University A+ and Rubyric teams for their efforts in continuously improving the MOOC platform and the tools needed to gather and maintain the data for this and other research.

REFERENCES

[1] ACM&IEEE. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, December 20, 2013.* Technical Report. http://www.acm.org/education/CS2013-final-report.pdf
[2] Fernando Alegre and Juana Moreno. 2015. Haskell in Middle and High School Mathematics, Vol. 1.
[3] A. Balanskat and K. Engelhart. 2014. Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe. (2014).
[4] Paul B. Baltes. 1987. Theoretical propositions of life-span developmental psychology: On the dynamics between growth and decline. *Developmental psychology* 23, 5 (1987), 611.
[5] Valerie Barr and Chris Stephenson. 2011. Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Inroads* 2, 1 (2011), 48–54.
[6] Jeanne M. Baugh. 2016. Beginning programming - why teach it and how to teach it? *Issues in Information Systems* 17, 3 (2016).
[7] T. Berger and C. Frey. 2016. Digitalization, Jobs, and convergence in Europe: strategies for closing the skills gap. (2016).
[8] Douglas H. Clements, Michael T. Battista, and Julie Sarama. 2001. Logo and geometry. *Journal for Research in Mathematics Education* 10 (2001), i–177.
[9] CSTA. 2016. Computer science standards. https://www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf. (2016).
[10] J Cuny, L Snyder, and J Wing. 2010. Computational Thinking: A Definition.(in press). (2010).
[11] M. Felleisen, R. Findler, M. Flatt, and S. Krishnamurthi. 2014. *How to Design Programs, Second Edition.* MIT-Press. http://www.ccs.neu.edu/home/matthias/HtDP2e/
[12] Matthias Felleisen and Shriram Krishnamurthi. 2009. Viewpoint Why computer science doesn't matter. *Commun. ACM* 52, 7 (2009), 37–40.
[13] Finnish National Board of Education. 2014. Finnish National Curriculum 2014. (2014). http://www.oph.fi/download/163777_perusopetuksen_opetussuunnitelman_perusteet_2014.pdf
[14] Gerald Futschek. 2006. Algorithmic thinking: the key for understanding computer science. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer, 159–168.
[15] Maarit Harsu. 2005. Programming Languages: principles, concepts, selection criteria (in Finnish). http://www.cs.tut.fi/~popl/nykyinen/Ohjelmointikielet-harsu.pdf. (2005).
[16] Fredrik Heintz, Linda Mannila, and Tommy Färnqvist. 2016. A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education. *Frontiers in Education* (2016).
[17] David Hemmendinger. 2010. A plea for modesty. *ACM Inroads* 1, 2 (2010), 4–7.
[18] House of Commons. 2016. Oral evidence: Digital skills gap. (2016).
[19] Incheon Declaration. 2015. Education 2030: Towards inclusive and equitable quality education and lifelong learning for all. In *World Education Forum.*
[20] R Burke Johnson and Anthony J Onwuegbuzie. 2004. Mixed methods research: A research paradigm whose time has come. *Educational researcher* 33, 7 (2004), 14–26.
[21] Jeremy Kilpatrick. 2012. The new math as an international phenomenon. *Zdm* 44, 4 (2012), 563–571.
[22] Simon C Kitto, Janice Chesters, and Carol Grbich. 2008. Quality in qualitative research. *Medical journal of Australia* 188, 4 (2008), 243.
[23] James A. Kulik. 1994. *Meta-analytic studies of findings on computer-based instruction.* Technology assessment in education and training, Vol. 1. Psychology Press, 9–34.
[24] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *ACM Inroads* 2, 1 (2011), 32–37.
[25] Jane Margolis and Joanna Goode. 2016. Ten Lessons for Computer Science for All. *ACM Inroads* 7, 4 (2016), 52–56.
[26] OECD. 2015. Students, Computers and Learning. (2015).
[27] OECD. 2016. Skills for a Digital World. (Jun 02 2016).
[28] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas.*
[29] Seymour Papert. 1996. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning* 1, 1 (1996), 95–123.
[30] T. Partanen, P. Niemelä, L. Mannila, and T. Poranen. 2017. Educating Computer Science Educators Online: A Racket MOOC for Elementary Math Teachers of Finland. In *Proceedings of the 9th International Conference on Computer Supported Education*, Vol. 1.
[31] Jean Piaget. 2000. Piaget's theory of cognitive development. *Childhood cognitive development: The essential readings* (2000), 33–47.
[32] Peter J. Rich, Keith R. Leatham, and Geoffrey A. Wright. 2013. Convergent cognition. *Instructional Science* 41, 2 (2013), 431–453.
[33] J. Sarama and D.H. Clements. 2009. *Early Childhood Mathematics Education Research: Learning Trajectories for Young Children.* Taylor & Francis.
[34] Emmanuel Schanzer, Kathi Fisler, Shriram Krishnamurthi, and Matthias Felleisen. 2015. Transferring skills at solving word problems from computing to algebra through Bootstrap. In *Proceedings of the 46th ACM Technical symposium on computer science education.* ACM, 616–621.
[35] Emmanuel Tanenbaum Schanzer. 2015. *Algebraic Functions, Computer Programming, and the Challenge of Transfer* (2015).
[36] Deborah Seehorn, Stephen Carey, Brian Fuschetto, Irene Lee, Daniel Moix, Dianne O'Grady-Cunniff, Barbara Boucher Owens, Chris Stephenson, and Anita Verno. 2011. CSTA K–12 Computer Science Standards: Revised 2011. (2011).
[37] Matti Tedre and Peter J. Denning. 2016. The long quest for computational thinking. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research.* ACM, 120–129.
[38] Jeannette M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.
[39] Jeannette M Wing. 2010. Computational Thinking: What and Why? Link Magazine. (2010).