



TAMPEREEN TEKNILLINEN YLIOPISTO  
TAMPERE UNIVERSITY OF TECHNOLOGY

Pia Niemelä

## **From Legos and Logos to Lambda**

A Hypothetical Learning Trajectory for Computational Thinking



Julkaisu 1565 • Publication 1565

Tampere 2018

Tampereen teknillinen yliopisto. Julkaisu 1565  
Tampere University of Technology. Publication 1565

Pia Niemelä

**From Legos and Logos to Lambda**  
A Hypothetical Learning Trajectory for Computational Thinking

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Festia Building, Auditorium Pieni sali 1, at Tampere University of Technology, on the 14<sup>th</sup> of September 2018, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology  
Tampere 2018

Doctoral candidate: Pia Niemelä  
Pervasive Computing  
Computing and Electrical Engineering  
Tampere University of Technology  
Finland

Supervisor: Professor Hannu-Matti Järvinen  
Pervasive Computing  
Computing and Electrical Engineering  
Tampere University of Technology  
Finland

Instructors: Professor Petri Ihantola  
Big Data Learning Analytics  
Department of Education  
University of Helsinki  
Finland

Professor Petri Nokelainen  
Industrial and Information Management  
Business and Built Environment  
Tampere University of Technology  
Finland

Pre-examiners: Professor Mike Joy  
University of Warwick  
The United Kingdom

Professor Arnold Pears  
KTH Royal Institute of Technology  
Sweden

Opponent: Professor Erkki Sutinen  
Interaction Design Department of Information Technology  
University of Turku  
Finland

# Abstract

This thesis utilizes design-based research to examine the integration of computational thinking and computer science into the Finnish elementary mathematics syllabus. Although its focus is on elementary mathematics, its scope includes the perspectives of students, teachers and curriculum planners at all levels of the Finnish school curriculum. The studied artifacts are the 2014 Finnish National Curriculum and respective learning solutions for computer science education. The design-based research (DBR) mandates educators, developers and researchers to be involved in the cyclic development of these learning solutions. Much of the work is based on an in-service training MOOC for Finnish mathematics teachers, which was developed in close operation with the instructors and researchers. During the study period, the MOOC has been through several iterative design cycles, while the enactment and analysis stages of the 2014 Finnish National Curriculum are still proceeding.

The original contributions of this thesis lie in the proposed model for teaching computational thinking (CT), and the clarification of the most crucial concepts in computer science (CS) and their integration into a school mathematics syllabus. The CT model comprises the successive phases of abstraction, automation and analysis interleaved with the threads of algorithmic and logical thinking as well as creativity. Abstraction implies modeling and dividing the problem into smaller sub-problems, and automation making the actual implementation. Preferably, the process iterates in cycles, i.e., the analysis feeds back such data that assists in optimizing and evaluating the efficiency and elegance of the solution. Thus, the process largely resembles the DBR design cycles. Test-driven development is also recommended in order to instill good coding practices.

The CS fundamentals are function, variable, and type. In addition, the control flow of execution necessitates control structures, such as selection and iteration. These structures are positioned in the learning trajectories of the corresponding mathematics syllabus areas of algebra, arithmetic, or geometry. During the transition phase to the new syllabus, in-service mathematics teachers can utilize their prior mathematical knowledge to reap the benefits of ‘near transfer’. Successful transfer requires close conceptual analogies, such as those that exist between algebra and the functional programming paradigm.

However, the integration with mathematics and the utilization of the functional paradigm are far from being the only approaches to teaching computing, and it might turn out that they are perhaps too exclusive. Instead of the grounded mathematics metaphor, computing may be perceived as basic literacy for the 21st century, and as such it could be taught as a separate subject in its own right.



# Preface

My family used to play a lot together, especially games involving strategy. In addition to games, my Mom felt compelled to foster our analytical thinking by providing us with ‘developing toys’, such as puzzles and lego. Mom was also a determined fan of mathematics, and we children almost grew tired of hearing about all of its benefits. Nevertheless, she still managed to sow the seed of interest. Computing grew from another source; my first contact with a computer was due to my uncle and his brand-new computer, a Spectrum. I can well remember sitting in front of a tv display screen with my cousin overwhelmed about opening scenes where computers would change the world, as they inexorably did.

After graduating from the Department of Technical Physics ( $\Phi$ ) in Helsinki’s Aalto University, I started working for Nokia as a software engineer, Java being my ‘logo’, the ultimate seedbed of computational thinking. The shift from natural sciences towards software profession had begun, yet the road ahead was going to be bumpy. After Nokia laid off thousands of engineers, including myself, I had once-in-a-lifetime chance to fulfill my other calling: pedagogy. The transition from being a scientific positivist to a relativistic humanist was not easy, but I eventually graduated as a class- and mathematics teacher in 2015 from University of Tampere. I have worked as a progressively inquiring teacher ever since, alternating between teaching and research.

This cross-disciplinary thesis synthesizes the accumulated experiences engraved on the palms of my hands, if not quite on my heart. The majority of the research was carried out at Tampere University of Technology under the research project ‘Skills, Education and Future of Work’ funded by the Academy of Finland. I would like to express my sincere gratitude to my professors, Hannu-Matti and the two Petris. Hannu-Matti, you restored my faith in human kind (read: professor kind). It is easy to work for a person that you respect. The thesis was greatly improved by the review comments received from the pre-examiners, Professors Joy and Pears, who pointed out the many vague rambling sections which required more concrete argumentation. During the process, I have shared my troubles with Kati and Martti, my fellow wanderers, and Tiina O., Irina and Katariina. Thank you all for the in-depth discussions and ‘think-tanking’. My thanks also go to Maarit, Maria, Antti J., Antti V., Ville I., Charis, Ekaterina, Chelsea, Adrian, MOT, Google Translator and all the co-authors and reviewers of my publications, but none more so than to Tiina Partanen. In addition, my family, Rikun Ruusut and PEO2017, thanks for providing alternative ideas to ponder when I was simply fed-up with my thesis.

Darling Petteri, you are my rock! You gave me the space and time to construct all these models, while keeping the house going in the meantime. Thank You! Last spring, wonder of wonders, we became ‘gamma’ ( $\Gamma$ ) and ‘taata’ ( $\Theta$ ). Once the remaining lambdas are finished, there are still plenty of letters for us to explore before the Final  $\Omega$ .

Yours, Pia

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>Acronyms</b>	<b>ix</b>
<b>List of Publications</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Objectives of the thesis . . . . .	2
1.2 Outline and contributions . . . . .	3
<b>2 Literature review</b>	<b>5</b>
2.1 Computational thinking (CT) positioned . . . . .	5
2.2 Didactic research of mathematics in resonance with CT . . . . .	12
2.3 The anticipated benefits of mathematics-CS integration . . . . .	16
2.4 The transition of mathematics teachers to CS . . . . .	17
<b>3 Research methods and theoretical frameworks</b>	<b>21</b>
3.1 Socio-constructivism as an underlying epistemology of FNC-2014 . . . . .	22
3.2 Design-based research . . . . .	26
3.3 Qualitative and quantitative data . . . . .	28
3.4 Analyses conducted . . . . .	28
3.5 Method summary . . . . .	29
<b>4 Overview and relevance of the publications</b>	<b>31</b>
4.1 Publication I: Significance of pedagogy- and context awareness . . . . .	32
4.2 Publication II: A holistic view of CS education . . . . .	33
4.3 Publication III: A problematic switch from visual to textual . . . . .	34
4.4 Publication IV: CS curriculum comparison: UK, US, FI . . . . .	35
4.5 Publication V: Transfer of mathematics teachers' prior knowledge . . . . .	36
4.6 Publication VI: CT/CS integrated in mathematics syllabus . . . . .	37
4.7 Publication VII: Necessary but under-taught discrete mathematics . . . . .	38
4.8 Publication VIII: Paradigms compared in mathematics-suitability . . . . .	39
<b>5 Results and discussion</b>	<b>41</b>
5.1 CT as an embedded commodity of mathematics . . . . .	41
5.2 Mathematics and CS concept overlap . . . . .	45
5.3 Mathematics teachers' professional development . . . . .	48

<b>6</b>	<b>Conclusions</b>	<b>51</b>
6.1	Implications for FNC-2014 . . . . .	51
6.2	Conclusive CT model . . . . .	53
6.3	Further research . . . . .	56
	<b>Bibliography</b>	<b>57</b>
	<b>Publications</b>	<b>69</b>





# Acronyms

ACM	Association for Computing Machinery
CAS	computer algebra system
CAS	Computing-At-School teacher coalition (in the UK)
CS	computer science
CSTA	computer science teacher association (in the US)
CT	computational thinking
DBR	design-based research
FNC	Finnish National Curriculum
ICT	information and communication technology
IEEE	Institute of Electrical and Electronics Engineers
IMRD	introduction-method-results-discussion style of structuring articles
ISTE	International Society for Technology in Education
K–12	school years from Kindergarten (K) to Year 12, in the Finnish system the elementary and high school (upper secondary)
MBA	master of business administration
MER	multiple external representations
MOOC	massive open on-line course
OECD	The Organisation for Economic Co-operation and Development
PISA	Programme for International Student Assessment
SDT	self-determination theory
SNA	social network analysis
SNS	social networking site
STEM	science-technology-engineering-mathematics
SW	software
SWE	software engineering
SWEBOK	software engineering body of knowledge
TIMSS	Trends in International Mathematics and Science Study
TPACK	technological pedagogical content knowledge
TPD	teacher’s professional development
TUT	Tampere University of Technology
UKNC	UK National Curriculum
UML	unified modeling language
USCC	US Core Curriculum
$Y_n$	year at school, numbering in sequence starting from 1, e.g., Y1



# List of Publications

- I Niemelä, P., Isomöttönen, V., and Lipponen, L., “Successful design of learning solutions being situation aware,” *Education and Information Technologies*, vol. 21, no. 1, pp. 105–122, 2016.
- Personal contributions:** The whole resource group participated in the content analysis phase. The situation awareness model was co-created with the second author. The author wrote the article, which was then reviewed by the co-authors.
- II Niemelä, P., Di Flora, C., Helevirta, M., and Isomöttönen, V., “Educating future coders with a holistic ICT curriculum and new learning solutions,” *Journal of Systemics*, vol. 14, no. 2, pp. 19–23, 2016.
- Personal contributions:** The author wrote the original article while the co-authors reviewed and rewrote selected parts of the original draft. Cristiano di Flora organized the survey in Rovio.
- III Niemelä, P., “All Rosy in Scratch Lessons: no Bugs but Guts with Visual Programming,” in *Frontiers in Education Conference Proceedings (FIE)*, 2017.
- Personal contributions:** The author wrote the article by herself. The interview data was collected by the author, but Scratch coursework was granted by the computing teacher. The initial results were reflected both with the students and the teacher, which led to the improved version of the motivation category model.
- IV Niemelä, P. S. and Helevirta, M., “K–12 curriculum research: The chicken and the egg of math-aided ICT teaching,” *International Journal of Modern Education and Computer Science*, vol. 9, no. 1, p. 1, 2017.
- Personal contributions:** Both authors participated in writing and reviewing, however, the first author’s contribution was more prominent. The background discussions were essential in shaping up the end results and conclusions.
- V Partanen, T., Niemelä, P., Mannila, L., and Poranen, T., “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, 2017.
- Personal contributions:** Partanen wrote the initial draft, which Niemelä re-structured to follow the introduction-method-results-discussion (IMRD) style. All authors contributed in the review phase.

- VI Niemelä, P., Partanen, T., Harsu, M., Leppänen, L., and Ihantola, P., “Computational thinking as an emergent learning trajectory of mathematics,” in *Proceedings of Koli Calling International Conference on Computing Education Research*, vol. 17, no. 1, 2017.

**Personal contributions:** The teachers’ essays were reviewed together, however, the contribution of the first and second author were the most prominent. The draft was mainly written by the first author, but the second author’s input was significant. The learning trajectories were sketched in a group among the first, second and third author. In the end, Harsu streamlined and shortened the draft as a conference paper. Leppänen’s contributions were proof-reading and several proposals for improving the draft in the review phase. In addition, Professor Ihantola made his remarks in order to improve the scientific quality, e.g., concerning methodology.

- VII Niemelä, P., and Valmari, A., “Elementary math to close the digital skills gap,” in *Proceedings of the 10th International Conference on Computer Supported Education*, vol. 1, 2018.

**Personal contributions:** The section that concerned terminology (CS, SWE, and ICT) and the development of the CS as a discipline was written by the second author. The first author wrote the remaining sections. Background discussions were influential in interpreting the results and wording them. The review process was short but efficient and aligned the controversial issues. The paper was granted the best student paper award.

- VIII Niemelä, P., Partanen, T., Mannila, L., Poranen, T., and Järvinen, H.-M., “Code ABC MOOC for math teachers,” *Revised Selected Papers*. Vol. 865. Springer, 2018.

**Personal contributions:** The article was written as an extension of Publication V. The scope, however, was widened to cover the tracks of Code ABC MOOC that target the secondary level, e.g., Python and Racket. Partanen and Poranen reviewed the Racket track feedback while the first author concentrated on the Python side. The comparison highlighted the underlying paradigms and the results were illustrated as a table with the adjacent columns of Python and Racket. Professor Järvinen was consulted in the paradigm issues in particular, whereas Mannila, as the expert of CT, reviewed the entirety and balanced the functional paradigm preferences with valid points about Python’s general usefulness.

# 1 Introduction

The 21st century society is digitizing at a rapid pace and the job descriptions of current professions are changing accordingly (Frey and Osborne, 2017). In addition to the changes in existing professions, new, previously unseen occupations are emerging, such as bloggers, community managers, and data analysts - or even ‘full-stack jedis’. Both domestic and multinational governing bodies have recognized the skills gap of computer science and the growth in the 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. The UK House of Commons has recently published two reports: The Digital Skills Crisis (Blackwood, 2016) and The Digital Skills Gap (House of Commons, 2016). These reports quantified the price of the shortage of skilled CS personnel and claimed that the digital skills gap costs the British economy £63 billion a year in lost GDP. The Cognizant Center for the Future of Work conducted a survey of over 2000 executives and 150 MBA students to summarize the digital viewpoints of European businesses; three-quarters of the respondents worried about the development of the right skills sets for the workforce in 2020 as the shift to digitalization accelerates (Davis, 2017). Globally, this shift is also seen as a more dynamic allocation of new digital talents and as an increase of a freelance workforce, referred to as a liquid workforce (Gupta, 2017) or liquid modernity (Nicolaidis and Marsick, 2016).

The discussion of the role of computer science in education is global, since a number of countries all over the world have introduced computational thinking, programming or computer science into their K–12 curricula. The literacy of the 21st century includes computing, which is comparable to basic skills such as reading and writing. Curricula and syllabi are at the heart of making computational thinking accessible for K–12 students. The Finnish National Curriculum 2014 (FNC-2014) integrates computational thinking and programming as parts of the mathematics syllabus. These changes have been in effect since autumn 2016. However, computational thinking and targeted computer science (CS) concepts must be determined more meticulously; the current description leaves space for speculation, various learning experiments and initiatives, and further research.

Computing in FNC-2014 divides into two complementary parts: the right mindset, i.e., computational thinking, and then the actual computing, that is, the programming basics of the selected programming language. Integrating computing into elementary education is a significant change. Currently, Finnish teacher training has not fully adapted to the change and is in the middle of a transition phase. Both pre- and in-service teachers need to learn to compute and to obtain a core understanding about the fundamental CS concepts. In teacher training, the CS basics need to reflect a clear theoretical perspective, define the exact fundamental concepts and their integration into mathematics to streamline the learning trajectory between mathematics and computing by explicating their conceptual similarities.

	<b>Years 1–2</b>	<b>Years 3–6</b>	<b>Years 7–9</b>
Digital competence	using digital media	impact of technology, tech-integration	
Mathematics	step-by-step instructions	visual programming	algorithmic thinking, coding conventions
Crafts		robotics, automation	embedded systems, own artifacts

**Table 1.1.** Computing-related topics in Finnish National Board of Education (2014)

FNC-2014 has been applied since August 2016 and it emphasizes the importance of digital competence as a part of general education throughout the school years; see Table 1.1. Digital competence is set as a cross-curricular aim, so that searching for, handling, and presenting information should utilize the latest information technology. Mathematics will provide a theoretical base for CS and teach the required programming skills, whereas crafts can provide new opportunities for self-expression in applying these new skills.

## 1.1 Objectives of the thesis

The main objective of this thesis is to contribute to the computing syllabus of FNC-2014 by clarifying the most crucial CS concepts and the application of computational thinking in a mathematics-proof manner.

The primary research questions are:

- RQ1. How to integrate computational thinking into the mathematics syllabus?
- RQ2. Which are the computer science fundamentals that suit mathematics education best?
- RQ3. How to train in-service mathematics teachers as computing teachers?

The main goal of this research was to examine the anticipated approach to computational thinking and CS basics in elementary mathematics, and to execute the research by following appropriate ethical guidelines and practices. The achieved outcome is a clarification of the practices of computational thinking and the current computing syllabus with regard to its most essential content, i.e. its fundamental concepts. Previously, Marttala (2017) has identified the need to amplify the learning goals, and several other sources, such as Tulivuori (2018), have demanded more resources for in-service teacher training.

In addition to the integration of CS into mathematics, this study also examines the process by which mathematics teachers are transformed into computing teachers. The target of this thesis is relevant because of the topicality and strategic importance of computing education. FNC-2014 is paving the way for the integration of computing into the Finnish school curriculum. Although previous waves of CS integration have left traces on the teacher population, the emphasis which FNC-2014 brings to the issue is relatively new. The literature review focuses on state of the art of the research on math-CS integration. The review reveals that the Finnish approach to math-CS integration is globally unique, which adds to the novelty of this study.

## 1.2 Outline and contributions

This thesis is divided into six chapters. The contents of each chapter are summarized below. Chapter 1 is an introduction to the field of CS education at the elementary level. The background and motivation for the study are given, followed by the research questions and the document outline. Chapter 2 reviews the relevant computational thinking research. Chapter 3 introduces the methods used, which are a blend of both qualitative and quantitative data, making this a mixed-method approach. The results contribute to the development of in-service training and curriculum planning, which are developed in iterative cycles, thus complying with design-based research practices. Chapter 4 gives an overview of each of the nine publications on which this thesis is based. Chapter 5 synthesizes the main findings as answers to the research questions. Chapter 6 crystallizes the implications of the research for embedding CS into mathematics and speculates on the implications for other school syllabi as well. The final model presented in Chapter 6 is a synthesis of the computational thinking models used in the study, and the thesis concludes with suggestions for Further research which emphasize the need for in-field testing to review the effects of the integration on learning outcomes, and to aid in the preparation of suitable learning material.

The main contributions are as follows:

- a learning trajectory for computational thinking,
- specification of the most fundamental CS concepts and integrating them into the mathematics syllabus, and
- analysis of the feedback obtained from mathematics teachers who participated in the in-service training MOOCs. From this analysis, and the comparison of computing conventions backed by their respective learning theories, the best-suited programming paradigm, i.e., the one least prone to misconceptions in the context of mathematics, can be inferred.





## 2 Literature review

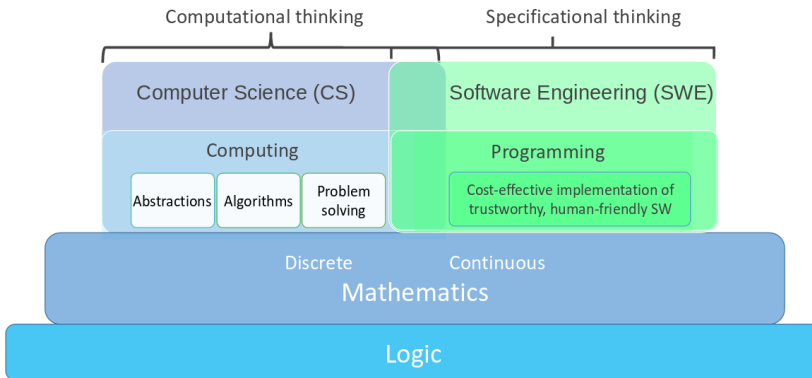
FNC-2014 introduces the integration of computational thinking and CS into the mathematics syllabus. Although previous initiatives to introduce CS into schools have left some traces on current teaching practice, FNC-2014 emphasizes its importance more strongly than ever before. During the literature review, the latest state-of-the-art research revealed that the Finnish approach of integrating computing into mathematics is relatively rare, so this study is breaking new ground.

### 2.1 Computational thinking (CT) positioned

Using mechanical tools to assist in calculations is an ancient practice. Tedre (2014) regards Quipus of the Incas or Chinese counting rods as early types of computing. Modern computing, as it is understood today, started to develop as a consequence of many theoretical and technological advances in the 1930s. The first steps were connecting Boolean logic to digital circuits, and finding a formal definition of an algorithm. This continued in the 1940s with the formalization of computable functions, storing data in memory (Tedre, 2014), and the invention of von Neumann architecture (Von Neuman, 1945).

In this thesis, computing is regarded as holistic machine-based problem solving. Its supra-conceptual discipline, CS, aims to formalize the discipline as a science. However, CS's position at the intersection of science, engineering, and mathematics has always complicated its categorization. This implies that it must have a robust theoretical basis and acceptable means and methods to examine, test and prove theories in order to reach conclusions that are of indisputable scientific significance (Denning, 2005). According to Dijkstra (1974), these methods comprise, e.g., proofs, program verification, and discrete mathematical and algebraic extensions which are specifically designed to formalize computing. However, because certain aspects of CS and Software Engineering (SWE) often mix and merge, CS as a discipline is still hard to define.

This thesis addresses the underlying division between theoretical CS and the more industry-oriented SWE, as can be inferred by the metaphor associated with computing education in Fig. 2.1. In addition to theoretical elaborations, CS aims to formalize CS practices in order to provide as error-free, and high-quality software as possible. SWE, on the other hand, is more prosaic and aims to implement computer programs in order to produce a desired outcome. Programming thus encompasses the whole process from defining user requirements to offering well-tested deliverables to the consumer. Although computing and programming may be seen as siblings: computing resides more in a more theoretical, ideal world, whereas programming falls into the realms of the practical world, with its tight schedules and budgets. Under this definition, programming benefits



**Figure 2.1.** CT situated in the landscape of CS and its neighboring disciplines (Denning, 2009; Dijkstra, 1974; Niemelä and Valmari, 2018, , as a combination of these)

from specificational thinking, user-centered design, process management skills, and the reflective procedures of iterative development. In contrast, CT aims to instill the desired mindset in young (and older) learners, and this is the central concept of this dissertation. This concept is analyzed in more detail below.

In the earlier publications on which this thesis is based, the accepted terminology had not yet been established so the terms computing and programming have been used interchangeably; Publication II and Publication IV even employ the term ‘information and communication technology’ (ICT). However, the terms used in the later publications are more precise, with computing and CS coming to the fore. In an attempt to explain the nuances between all these concepts in more in detail, Publication VII clarifies the development of the Finnish SW industry, which has had an influence on the accepted terminology now in use. To widen the perspective to cover specificational thinking would require the inclusion of other school subjects in addition to mathematics, such as English, which would enhance the realization of the ‘human-friendly’ element of the topic. In the end, the terms ‘computing’ and ‘CS’ are regarded as being closest to the procedures embodied in teaching CT/CS integrated with mathematics at elementary-school level. To distinguish between CT and CS, CT is more abstract thinking skill and a convention of achieving things exemplified by algorithmic and logical thinking, whereas CS refers to certain fundamental concepts and procedures, and is a more substantial topic.

### 2.1.1 Papert promotes Turtle to scaffold CT

Recent school curriculum enhancements utilizing programming have given added impetus to CT in Finland, although Papert, the CS education luminary, had in fact heralded the importance of CT notably earlier, started from late sixties. The next quotation is from the year 1996: ‘*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.*’ Therefore, CS is not merely using computers, but it also means using them to create digital artifacts and carrying out authentic projects that provide options for self-expression. By identifying themselves as potential creators, CS students can experience a feeling of empowerment.

Papert applied the fundamental ideas of child psychologists and constructivists, such as Piaget and Bruner, to computer science education. In *Mindstorms*, Papert (1996)

develops the theoretical basis of Piaget's genetic epistemology and children's cognitive development, in an attempt to combine them with the affective side of learning. By providing plenty of computational stimuli, such as turtles – whether floor, screen or dyna turtles – a child's spontaneous sense of geometry and logic is stimulated in order to develop its understanding of CT, and other powerful concepts, such as the laws of physics. On the basis of his observations of children playing with turtles and elaborating on their instructional language, LOGO, Papert goes so far as to see parallels between the Turtle system's computational geometry, and Euclid's axiomatic and Descartes's analytic geometry. In the search for appropriate algorithms, children are encouraged to fully experience the concrete operations with 'body-tonic' movements by imagining themselves in the place of the commanded turtle. This turns '*a stereotypically disembodied mathematics to activities engaging a full range of human sensitivities*' (Papert, 1980). Papert thus aims to foster in children the propensity to consider how they themselves can assist a computer in solving problems. According to him, thinking about thinking turns a child into an epistemologist as they work out how to get the computer to act correctly.

To promote the development of the child's CT, Papert anticipates that the shift from straightforward triadic and square turns to more challenging circular ones will make them receptive to more advanced mathematical ideas. Instead of the simpler actions needed for straight-line turns, i.e. moving forward and then taking a new direction, circular turns are more demanding. To introduce the concept of circular motions, Papert recommends: move-a-little, turn-a-little. However, this procedure only approximates the smooth arc of a circle. To achieve the ideal outcome, each step ought to be ever shorter, ultimately being squeezed into infinitesimal steps approaching zero. Such an exercise may provide the dawn of 'computational Turtle geometry' in a learner, whose required differential thinking anticipates the skills needed in mathematical analysis and calculus. For Papert himself, playing with gears was his 'Turtle experience' and the catalyst for CT.

The Turtle exercises not only engage students but are Papert promotes a more playful, or – using his vocabulary – a more bricoleur way of nurturing computing basics and mathematical practices in sync. According to his interpretation, computing is applied mathematics, and playing with turtles provides a gentle way of practicing it and strengthening a child's self-efficacy. For formal mathematics lessons, Papert is concerned about the increasing number of math-phobic students that label themselves as too stupid to learn. He hypothesizes that this trend is a consequence of an over-rigid methods of problem-solving which pressurize the learner into getting it right on the first attempt, or in his words, the 'technology of grading'. Because of this phobia, students appear to be mathematical under-performers.

To counteract this trend, Papert sees exploration and debugging as integral parts of the incremental and flexible problem-solving practices required in computing, and suggests applying an analogous mindset to mathematics. Papert emphasizes that, in the context of CS, the question is not whether a solution is right or wrong, but whether it is fixable. Moreover, debugging need not be limited merely to problem solving, but can be extended to encompass self-reflective practices required in any learning process, stretching into the realms of meta-cognitive skills. To further exploit the early programming experience, the gained experiences should be explicitly abstracted. This abstraction will foster progress in the more cognizant phase of formal operations that takes place about the age when children switch from primary to secondary school (Piaget, 1972). Schooling and literacy naturally affect a learner's rate of development. In mathematics, the abstraction could mean e.g. noting the regularities of triangles, squares, and circles, inducting these observations to

such time-proof laws and lemmas, prototyped by Papert's powerful ideas.

### 2.1.2 The second wave of CT

In her seminal article, Wing (2006) re-emphasizes the importance of CT by proclaiming its pivotal role in computing education, yet fails to provide a comprehensive definition. Even if no absolute consensus on the definition has been reached – and it is questionable, whether it can be reached given the current diversity of definitions – the majority of the experts in the field are content with Wing's later description (2010): '*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.*' An exemplar for such thought processes are small children, referred to as epistemologists by Papert, who guide turtles to draw geometrical shapes: they learn the basics of computing and simultaneously unfold the regularities of Euclidean geometry.

In summary, defining CT comprehensively is a challenge. To complicate the situation further, the paradigms, languages and tools used each bring their own flavor to the learning experience, which the analogous 'divide-and-conquer' approaches tend to conceal.

#### CT decomposed

Barr and Stephenson (2011) define data collection, its analysis and representation, problem decomposition, abstraction, algorithms, automation, parallel code and simulation as the cornerstones of CT. However, parallel code and simulation, for instance, are not commensurable with the all-encompassing principles of abstraction and automation, but more concerned with minor formalities and implementation details. In addition to researchers, many teacher associations and education organizations have participated in efforts to clarify the practices of teaching CT. These organizations include the International Society for Technology in Education (ISTE), the Computer Science Teachers Association (CSTA), and Computing-at-School in the United Kingdom (CAS-UK). ISTE is a not-for-profit organization dedicated to supporting the use of information technology in teaching K–12 students as well as supporting their teachers in including CS education in their syllabi. The ISTE definition (ISTE, 2015) complies more or less with the model constructed by Barr and Stephenson (2011).

Like ISTE, CSTA promotes computer science education both in America and worldwide by empowering K–12 CS teachers. Although mainly prominent in America, CSTA consulted with the co-located ISTE in defining CT in an elementary-education-proof manner. The outcome of this highlights problem-solving as the core proficiency, which involves, e.g., abstraction and analysis skills (CSTA, 2016). In addition, the character of the ideal student was described, which comprises such qualities as confidence, persistence, and tolerance of ambiguity (Seehorn et al., 2011); skills that are especially needed when problem solving is open-ended. In 2016, CSTA reorganized the previous standard strands from 2011 into the new concept sets of computing systems: data & analysis, networks & Internet, algorithms & programming, and finally, the impacts of computing (CSTA, 2016). Compared with the previous strands, data & analysis and network & Internet were new, reflecting the newly-realized importance of these areas. In essence, the renewed strands approach the ISTE model.

In contrast to ISTE and CSTA, who represent CS issues concerned with the American school curriculum, Computing-at-School (CAS-UK) reflects the UK's more European view of CT. Regarding problem solving, CAS-UK divides the solution phases into more abstract,

and more practical solutions (Csizmadia et al., 2015). The former consist of problem solving, and the latter of the application of technical skills in order to solve the problem computationally. As elements of CT, CAS-UK lists logic, algorithms, decomposition, finding patterns, abstraction and evaluation.

The last model introduced here is from Grover and Pea (2013). The model manages to capture all the essential super-classes of abstraction, algorithms and assessment, which are defined in more detail below:

- abstractions as pattern generalizations, and as a key to dealing with complexity,
- algorithms as structured problem decomposition, the algorithmic notions of flow of control, and systematic processing of information,
- assessment to evaluate students' understanding and use of abstraction, conditional logic, and algorithmic thinking. Instead of teachers assessing their students, which demonstrates the 'technology of grading', the goal should be self-reflecting students familiar with the conventions of debugging and systematic error detection, and with optimizing performance and efficiency.

### 2.1.3 CT problem-solving heuristics

In abstracting and systematizing problem solving with algorithms, Papert's most prominent exemplar comes from the mathematics side, namely, the eminent Hungarian didact of mathematical education, Pólya (1887–1985). Papert introduces the Pólyan heuristic strategy of problem-solving to achieve more perceptive learning (Pólya, 1945). The strategy includes principles such as the decomposition of problems (divide-and-conquer), the recognition of analogous patterns, generalization, and specialization, the echoes of which are carried far into the discourse of CT, as heuristic means of systematizing problem-solving.

These systematics and means of abstraction are not restricted to mathematics alone, but may be found in other science-technology-engineering-mathematics (STEM) subjects as well, in particular physics. Thus, other templates, besides that of Pólya, may be introduced. Most Finnish didactics and pathfinders, e.g. Kurki-Suonio and Kurki-Suonio (2000), regard mathematics as an abstraction primer, especially in geometry (proportionality and symmetry), where the very same principles are transferable to more advanced topics, even up to the domain of modern physics. Later, quantifications – dividing, multiplying, adding and reducing implemented in the form of thought experiments for approaching zero or infinity – provide a means for the mental assessment of the anticipated rules and relations between the examined quantities. In summary, the means of abstraction introduced in CT are not epistemologically unique but rather shared between all the subjects that exploit mathematical problem-solving methods, e.g. other natural sciences.

### 2.1.4 Computing curricula abroad

The majority of the European countries surveyed (17 out of 21) introduce computing as an emergent new addition in their K–12 curricula (Balanskat and Engelhart, 2014; Heintz et al., 2016). Starting from the primary education, various approaches have been suggested by different projects and stakeholders. In the beginning no computer is even necessary. For instance, in the TACCLE3 project, the CAS-UK teachers ( $N = 357$ ) employ unplugged, contextualized activities, such as playing with robots and lots of

hands-on practice with digital artifacts (García-Peñalvo et al., 2016). In addition, the CS Unplugged project has assembled an inspirational exercise package to be used in a school context without computers to learn about binary numbers, trees as data structures, and basic algorithms for searching, such as binary search (Rodriguez et al., 2017). In compliance, CAS-Barefoot defines its own model to teach CT, the prominent subtopics being algorithms and logic (Barefoot, 2014). In the UK, computing as a school subject comprises a more holistic and system-wide view of computer systems, networks, and architectures, as defined by the General Certificate of Secondary Education (GCSE, 2015). In addition, the GCSE strongly emphasizes starting the strand of security and ethics already in the early school years. To foster computational creativity, students have to implement digital artifacts once they have gained the required skills.

In their review, Heintz et al. (2016) examine the curricula of Australia, New Zealand, Estonia, Finland, Sweden, Norway, the UK, South Korea, and the USA in detail. In Australia (2015) and New Zealand the subject is called Digital Technologies (DT), and has a strong focus on CT, as well as the development of both digital literacy and programming skills. In Australia, the subject is mandatory at junior level (K–Y6), and is optional thereafter (Y7–12). In New Zealand, DT is only taught in high school (Y10–12), and covers programming and, albeit only cursorily, a wide range of CS topics including algorithms, human-computer interaction, artificial intelligence and computer graphics.

Estonia has plunged straight into the issue, linking programming with tangible digital exercises such as robotics and electronics. In contrast, much of Scandinavia is lagging behind. In Finland, Sweden, and Norway, the whole education system is in turmoil due to rapid changes and inadequate resourcing. If the goal is to be riding the crest of a digital education wave, the vocational education of teachers is a bottleneck to realizing this ambition. For instance, Swedish schools have provided elective computer science courses since the 1970s, originally called informatics, and later changed to information technology. As in Finland, Sweden introduces programming as part of the mathematics and crafts curriculum, e.g. the algebra section examines how algorithms are created and the problem-solving section analyzes algorithms by actually implementing programs and testing them out. In crafts, students study what materials can be enhanced with digital technology and construct two- and three-dimensional diagrams, models, and patterns, both with and without digital tools. Moreover, these models are backward-compatible with mathematics calculations.

In addition to mathematics and craft, in 2017 Sweden added technology as a core subject in the curriculum (Skolverket, 2017). To aid progress towards this goal, Sweden's innovation agency Vinnova funds several in-service training projects for Swedish teachers, such as 'Computational thinking for all' started in 2016 (Heintz and Mannila, 2018). This addition to the curriculum aims to give students an insight into how computing is intertwined with industrial and scientific practices, and how it can be applied in various contexts. The 'comments' appendix highlights the link between problem-solving in mathematics and computing (Skolverket, 2017): *'problem solving consists of modeling it, i.e., translating a situation into mathematical language of symbols. A general model can be expressed as an algorithm that is created based on a mathematical or everyday function and can solve various kinds of problems, such as sorting large amounts of data. Students should therefore meet the content how algorithms can be created when programming for mathematical problem solving. When students use programming to solve mathematics problems, they also have the opportunity to create, test and improve the algorithms.'*

In Norway, educators trumpet the success of their primary school students in the Trends

in International Mathematics and Science Study (TIMSS). However, according to the PISA tables, the situation for lower-secondary school students is not as encouraging. In 2015, Norway created the initiative ‘Science for the Future’, whose main goals were to increase students’ interest in mathematics-science-technology (MST), to strengthen the pre-service teacher training, and to decrease gender bias (Norwegian Ministry of Education and Research, 2010). Programming is introduced as part of the Mathematics curriculum, starting from giving step-by-step instructions as the basis for programming in Years 1–3. In Years 4–6, students learn how to use algorithms for programming by utilizing sequences, repetition and abstraction. The algorithms are then created, tested and improved as part of programming for mathematical problem solving. Another Norwegian initiative, *Lær Kidsa Koding*, lobbies government, schools, and politicians to achieve a more established position for computing in the school curriculum (LLC, 2017). Overall, Norwegian educators desire the curriculum to be constructively more aligned and concise by providing more in-depth learning focusing only on the core competencies.

The early birds of computer science education were South Korea, the UK and the US. However, in South Korea enthusiasm for the topic waned between 2004 and 2012 in favor of subjects that ensure easier access to higher education, such as mathematics. By renewing the informatics curriculum in 2018, South Korea is attempting to recover the initiative. In comparison, the UK and the US are performing much more strongly. The UK has added compulsory courses of CS and offers GCSE exams for the qualification. In addition, it provides strong assistance to teachers. For example, the CAS-UK community freely delivers useful material and training in CS. In the US, computer science is still an elective subject. However, the vision of the government has been clear. For example, in 2013 President Obama promoted the project Hour-of-Code by Code.org (Partovi, 2014; Wilson, 2015). In addition, there are many strong actors, such as Google, Microsoft, and several organizations such as CSTA and ISTE that are realizing this vision with parallel projects, such as CS4All (Vogel et al., 2017), and to balance the gender bias, CS4All-G (Marghitu et al., 2014). These new fancy initiatives, tools and extra-curricular hack clubs have managed to reverse the trend of falling enrollments on the CS courses.

In short, the main current dilemmas for school curriculum planners seem to be whether or not computing requires a syllabus of its own, i.e. should it be taught as a separate subject, whether that subject should be optional, and what fundamental concepts should be covered. If there is no formal qualification for teaching CS, the quality of teaching will vary and depend on ‘good luck in the teacher lottery’. In addition, if CS is to be integrated into other subjects, it is more challenging to target the learning outcomes in a formal and standardized way. Heintz et al. (2016) noted that many vocal proponents advocate teaching CS as its own separate subject, purely from the perspective of the subject itself, and it seems likely that this may well be the best approach.

### Critical views

Integrating mathematics with computing is not risk-free, and for this approach to be implemented efficiently, the CS elements of the syllabus need to be developed with reflective feedback loops. For instance, a recent OECD report (OECD, 2015) demonstrated that the greater the extent to which technology was merged with the mathematics syllabus, the poorer were the results. In addition, motivational aspects should be taken into account. For instance, South Korea suffers from falling enrollments in computing courses, apparently because the students’ attitudes towards the subject became more negative due to the increase of computer science lessons in elementary school. The reasons identified



were the absence of an appropriate policy and comprehensive evaluation methods (Choi et al., 2015). As a remedy, the authors of this Korean study recommend determining a robust policy, goal clarification, formal qualifications, and adequate resourcing in teacher training.

In addition, there are a remarkable number of groundless promises associated with CT. For example, according to Mark Guzdial, a professor in the School of Interactive Computing at Georgia Tech, ‘*There is no reliable research showing that computing makes one more creative or more able to problem-solve. It won’t make you better at something unless that something is explicitly taught*’, and he continues, ‘*You can’t prove a negative, but in decades of research no one has found that skills automatically transfer*’ (Pappano, 2017). In addition to Guzdial, Hemmendinger (2010) pleads for caution and reminds readers that algorithmic thinking is anything but new. As he points out, the term ‘algorithm’ has its origins in 9th-century Persia (Rocker, 2006). The author does, however, list scalability, feasibility and optimizing resources as the integral characteristics of computing. Similarly, Tedre and Denning (2016) recall the long history of CT, which they trace back to the 1950s. Instead of exaggerating the advantages, the authors would prefer to explore the results of previous learning experiments in order to avoid repeating the same mistakes over and over again. Furthermore, the authors question the transferability of algorithmic thinking, which, according to them has hardly ever transferred to the benefit of other subjects, despite the high expectations it arouses.

## 2.2 Didactic research of mathematics in resonance with CT

The novelty value of learning programming basics integrated with elementary mathematics has not yet worn off. In mathematical thinking, the exemption from mechanical calculations affords an opportunity to concentrate on higher-level operations, such as the phases of abstraction, algorithmic thinking, and analysis. The emergence of symbolic calculators and computer algebra systems (CAS) technology in the nineties gave a foretaste of what was to come and led researchers to start talking about the instrumentalization of mathematics. The current emergence of computers in mathematics teaching demands a theoretical extension to deal with programming languages as instruments.

### 2.2.1 Abstraction as moving from procedural to conceptual knowledge

Abstraction means leveraging one’s thoughts above the concrete towards more abstract and general ideas, i.e., from procedures to concepts, or, to give it a more didactic wording and flavor, from structural to functional knowledge. Procedures cover the routines of rote calculations, whereas conceptual learning comprises internalization of central concepts and seeing how these concepts interact with previously-learned knowledge. In mathematics, the threads of procedural and conceptual approaches are interwoven in the praxis of mathematics lessons. Hibert and Lefevre (1986) introduced this dichotomy of the procedural versus the conceptual in their book; the edition was renewed and completed in 2013. According to the authors, similar overlapping dichotomies commonly exist in the discipline of knowledge building. For instance, Piaget (1972) based his theory of genetic epistemology on the transfer from concrete to formal operations, where formal operations comprise abstractions developed by hands-on experiments. Anderson (1990) distinguished between procedural and declarative knowledge, where declarative knowledge is substituted

for conceptual. Brownell and Chazal (1935) emphasized the need for the nexus of isolated skills to be connected to existing knowledge in order to enhance conceptual understanding.

There are several parallels in mathematics that highlight the different nature of these types of knowledge, for instance: procedural vs. conceptual knowledge (Tall et al., 2001), syntactic vs. semantic (Resnick et al., 2009), skills vs. principles (Gelman and Gallistel, 1978), or structural vs. functional (Cai et al., 2011). Cai et al. argue that in mathematics routines *‘naked equations and [emphasizes] procedures for solving equations are all hallmarks of a structural focus’* which is in contrast to the functional approach, involving a much higher level of conceptual emphasis.

In moving from procedural to conceptual/functional, a supportive educational framework is crucial. Instead of an excess of routine calculations, a mathematics teacher should provide puzzles and open-ended problems. These should be meaningful for the students, enhance their understanding of the problem area and foster algorithmic thinking. Such open-ended problems should have their origin in real-life and should require the handling of large amounts of data and extensive calculations so that the solution can only be found with the help of calculators and computers, such as many statistical calculation, e.g., getting smooth bell-shape curves from a normal distribution.

Although most research emphasizes the importance of conceptual over procedural knowledge, nowadays the bidirectional nature of their interaction is noted to be beneficial for both: *‘two forms of knowledge are treated as distinct, but linked in critical, mutually beneficial ways’* (Artigue, 2002; Hiebert, 2013, for instance). To exemplify this inter-relatedness, Hiebert (2013) describes the conceptual bridge of a place-value. Even if the place-value procedure has been correctly executed by the learners when doing subtractions, i.e., borrowing from the next decade succeeds, they will only reach a full conceptual understanding of the principle by internalizing the geometrically increasing magnitudes of ten-base blocks, and seeing place as an indicator of this magnitude. For instance, to fully understand subtraction, a student must internalize the next place as being a ten-block store from which one can borrow.

In Hungarian mathematics, for example, different tangible manipulatives scaffold cognitive bridges of this kind (Tikkanen, 2008; Varga, 1988): place-value exercises are carried out with various appliances, such as place-value charts, number lines (in paper, with tape on a floor), abacuses, and construction series. Hungarian mathematics, more specifically the Varga-Neményi method (Kurvinen et al., 2014), is not restricted to the decimal number system only, but the exercises cover different number systems (e.g. binary). In addition to place-values, commonly known conceptual bridges include, e.g., ideas of the common denominator and the relative sizes of quantities (Hiebert, 2013).

### 2.2.2 Conceptual abstraction leveraged with algebra

The transition from arithmetic to algebra exemplifies the process of abstraction in stepping from the procedural to the conceptual in that students must transfer from number mathematics to letter mathematics. Vygotsky (1980) asserted that, *‘the student who has mastered algebra attains a new higher plane of thought, a level of abstraction and generalization that transforms the meaning of the lower (arithmetic) level’*. Abstract thinking consists of the expression of generality, whereas algebraic thinking utilizes a learner’s natural ability to make mathematical sense. The expression of generality in increasingly systematic, conventional symbol systems is, according to Kaput (2008), one

of the core aspects of algebraic reasoning, the other being syntactically guided actions within organized systems of symbols.

The pitfalls of the transition into algebra have been well-documented (Fong et al., 2014; Schanzer, 2015). Various experimental approaches have been utilized to ease the threshold of transition, such as early algebraization (Kieran, 2011) and fostering functional thinking at the elementary level (Wilkie, 2016a). Both the above approaches aim to move the transition phase to an earlier stage in the learner's education, from the secondary to primary level, using age-appropriate content, of course. As pointed out by Carraher et al. (2008) however, early algebra does not mean algebra early. For instance, Kieran (2004) thinks that algebraic thinking can be taught without the use of the letter-symbolic, and can be built up by identifying numerical and geometric patterns and by trying to describe them with alternative means, by the learners inventing their own systems of notation, for instance. She also focuses on seeing expressions and equations with 'algebra eyes'.

In algebra, the most fundamental concepts are variables and functions. According to Küchemann (1978), the concept of a variable evolves through six progressive stages starting from being a single, irrelevant value, then being recognized as a label or an object, then as a specific unknown, a generalized number, and finally as a functional relation. According to Wilkie's epistemological view, internalizing algebra requires deepening levels of objectification, which she called arithmetic, factual, and contextual generalizations (Wilkie, 2016a). As an intermediate phase before the symbolic one, she also notes the value of pro-numerals (e.g., *number\_of\_articles*). In word problems, pro-numerals associate effortlessly with an unknown in the problem description. In particular, Wilkie focused on growing patterns and their role in developing functional thinking. In generalizing the patterns, a student should develop a recursive solution that requires consecutive calculations of the next steps. Explicit generalization should capture the direct rule and relational correspondence, i.e., a rule for the  $n^{\text{th}}$  term.

### 2.2.3 Algorithmic thinking in mathematics

An algorithm is a streamlined sequence of steps required to solve a problem (Doleck et al., 2017). At its simplest, a cooking recipe or driving directions represent such a sequence (Yadav et al., 2016). Streamlining means the optimization of time and resources in problem solving. In optimization, computing has to take into account the limits of the concrete world, such as scalability, feasibility, the optimization of computer resources and performance, and the processes of interpretation and compilation (Hemmeldinger, 2010). In mathematics, streamlining the solution has a different flavor. Facets such as elegance, simplicity, intricacy and logical approach are appropriate stream-lineage measures (Dreyfus and Eisenberg, 1996; Halmos, 1968).

Research demonstrates that even if children do not know how to express their thoughts in symbolic language, they are endowed with intuitive problem-solving capabilities. If the instructional framework is well designed, it can foster the development of children's skills in more advanced algorithmic competencies such as sorting and searching (Baroody, 2004; Clements and Sarama, 2007). To get to the very essence of algorithmic thinking, it must be understood that the iterative and recursive processes are substantive, and inherently more frequent in discrete, numerical methods, and computing. In classical, calculus-heavy mathematics, such iterations are not as frequent. Mathematics praxis that use an iterative approach are, for instance, sums, products, recursions, e.g. factorial, bracketing of roots, approximations and theory derivations, where the exact solution is approached inductively, or in tiny increments. Discrete mathematics provides a number

of abstraction tools for algorithmic development in computing, such as set and graph theory, probability and combinatorics, and the methods of formal logic. Sets, relations and graphs are helpful in presenting much of the discrete data intrinsic to computing.

#### 2.2.4 Multiple external representations to facilitate abstraction

Multiple external representations (MERs) illustrate the same topic from different perspectives. For example, a function may appear as a relation of  $x$  and  $y$ , a graph, a map from argument set to image set, or as a metaphor for a function machine. Flexibility in moving from one representation to another indicates deeper understanding (McGowen et al., 2000). Wilkie and Clarke (2016) describe representational flexibility as a resilience with the order of operations, and fluency with distributive laws and the equivalence of expressions. In terms of practicing algebraic abstractions, pre-algebra exercises are a good approach, using such pedagogic devices as growing patterns (Wilkie, 2016a,b; Wilkie and Clarke, 2016), pictorial equations in Singaporean mathematics (Cai et al., 2011), and games. Some startling examples from DragonBox Algebra demonstrate that even a five-year old child is capable of solving algebra problems if the presentation is age-appropriate (Liu, 2012). Overall, much younger students than was previously expected are capable of learning and presenting their algebraic thinking (Brizuela et al., 2015). Although a lack of experience often hinders the growth of abstract thinking and intuition, (Jurdak and Mouhayar, 2014), McGowen and Tall (2010) posit ‘met-befores’ as the foundation of mathematical intuition.

#### 2.2.5 Analysis in CT associates with sociomathematical norms

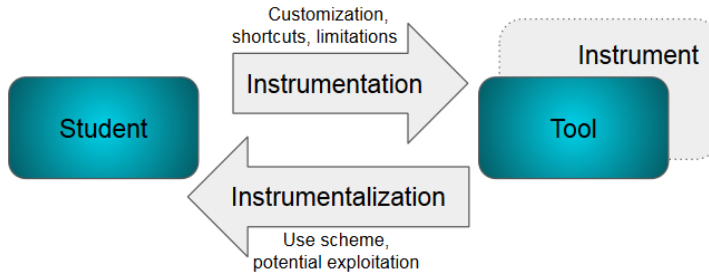
Yackel and Cobb (1996) study the paradigm of sociomathematical norms, which Stephan (2014) defines as normative criteria according to which students of a class create and justify their mathematical work. These norms are applied in negotiations of the criteria of different, efficient, or sophisticated mathematical solutions and the criteria for an acceptable mathematical explanation. A mathematics teacher challenges students to invent multiple alternatives ending up with the same result. Among the presented alternatives, the class should evaluate the most sophisticated and elegant solution. In addition, the authors emphasize the need for a rationale and justification. This approach is seconded by Izsák (2011), who sees that teachers elicit students’ thoughts by engaging them in classroom conversations to explicitly compare different approaches, thereby encouraging the emergence of more powerful algebraic representations. In terms of teaching algebra, Koellner et al. (2011) also challenge teachers to pose Socratic questions to push students forward in their thinking. Argumentation provides a means of capitalizing on a student’s contribution and making it accessible to the whole class.

To conclude, the common practices of mathematics, such as using everyday problems as material, breaking down problems into smaller sub-issues (Pólya, 1945), refining their stages (Joutsenlahti, 2003), optimizing the solution, and thinking of the rationality of the results and alternatives for potentially more optimal solutions (Yackel and Cobb, 1996) are good ways to practice the principles of CT in mathematics, and this is nothing new.

#### 2.2.6 Instrumentalization

The emergence of Computer Algebra System (CAS) calculators in mathematics in the 1990s triggered a number of studies that aimed at revealing these tools’ influence on learning. Compared to the plain old pen-and-paper method, a student was equipped

with an additional instrument to assist in solving more complex and computing-intensive problems. In illustrating the effect of calculators in a mathematics class, Trouche and Drijvers (2010) suggest the analogy of a single musical instrument in an orchestra. As with a violin player, a CAS player first needs to master his instrument. The authors specify the entity of an instrument with the following equation:  $instrument = tool + use\ scheme$ . The tool becomes an instrument only when students know how to ‘play’ it. This process of taking possession is called ‘instrumental genesis’. In order to capture this genesis, Drijvers and Trouche (2008) have coined terms for the two counter-directional sub-processes involved, instrumentation and instrumentalization; see Fig. 2.2.



**Figure 2.2.** Instrumental genesis: a student takes a tool into possession and starts to lean on it.

Briefly put, instrumentation is the user’s engraving on a tool in order to customize it for their use, whereas instrumentalization is when the tool etches its marks on the user’s activities and schemes. During instrumentation, a student customizes the tool, e.g., chooses suitable themes and shortcut keys, and defines scripts for certain tasks to be automatically executed. In other words, the student redefines the tool to suit their own purposes. During instrumentalization, in turn, a student adapts the tool for different purposes and begins to think and solve new problems with the tool. In this way, the tool leaves its trace on the action schemes of a student. Later, the same group of researchers revisited the idea of instrumental genesis and enhanced it with documentational genesis, where a mathematics teacher transforms resources into documents for their own use in teaching (Gueudet and Trouche, 2009). In school math lessons, CAS calculators have now been largely superseded by computers. With computers, instrumentalization may be interpreted as the internalization of the first programming language as a fixed starting point, although the path should eventually lead to different languages and paradigms.

### 2.3 The anticipated benefits of mathematics-CS integration

Mathematics has been chosen as the basis for CS as they both require algebraic, logic and problem-solving skills. Compared with CS, mathematics has a well-established learning trajectory which has evolved gradually into its current form since the very beginning of the modern school system. Despite the fact that certain areas have been dropped and some reintroduced, the core content of the school math syllabus has remained largely the same for decades. There have been new initiatives in teaching math. For instance, due to the so-called ‘New Mathematics Movement’, set theory has experienced a kind of yo-yo effect: first, having been pushed into Finnish elementary schools in the seventies, after which it gradually vanished from the syllabus (Bernack-Schüler et al., 2015; Pehkonen, 2001). New Mathematics aimed at bringing school mathematics as closely in line as possible with higher-level, scientific mathematics rooted in set theory. However, the approach was too theoretical and the learning outcomes suffered.

A school curriculum reflects the society in which it exists, so as society changes, it becomes necessary to review the content of the curriculum and the weighting of certain syllabus topics. For example, the current shortage of software engineers has shifted the educational imperative in Finland away from the more traditional natural sciences towards CS. According to the feedback from professional software engineers in the field, if elementary math teaching in schools is to be more supportive of CS, then the emphasis should be shifted away from continuous mathematics and towards discrete mathematics. The FNC-2014 integration of CS into the school curriculum is the biggest revision of the Finnish school syllabus for a long while. It is, however, justified by recent research, and is based on the obvious inter-relatedness of mathematics and CS. For instance, the practices of mathematical thinking are deeply interwoven into CT, in particular with regard to problem solving (Wing, 2008). This can be divided into several sub-skills, depending on the categorization used. For example, abstraction – in particular in conjunction with algebra (Susac et al., 2014) – and analytic and critical-thinking skills (Elliott et al., 2001) are frequently mentioned as sub-skills of mathematical thinking that overlap with CT.

Since FNC-2014 has already stipulated that CS be integrated with mathematics, there is a positive bi-directional synergy between the two subjects that can be exploited. Even though the transfer from mathematics to CS has already been recognized (Lent et al., 1991; Zeldin and Pajares, 2000), the transfer in the opposite direction, from CS to mathematics, may not be that obvious. However, certain topics, such as algebraic variables, functions, and logic can be taught either directly or indirectly through programming and familiarization with the basic concepts of CS. If the benefits of integration, especially improved learning outcomes, could be reliably demonstrated, it would be a powerful selling point for the mathematics teachers who now have to implement the changes.

## 2.4 The transition of mathematics teachers to CS

The requirement of teaching programming basics integrated into mathematics lessons has challenged and accelerated Finnish math teachers' professional development (TPD). Successful TPD increases a teacher's perception of self-efficacy. The self-reinforcement and self-efficacy theories of Bandura (2006) provide a view on motivational factors, where self-efficacy is a more accurate predictor of successful professional development than the teacher's actual achievements. Depending on the achieved self-efficacy level, Kennedy (2016) discusses the enactment problems of bringing new skills, in this context programming skills, into a classroom context. Accordingly, the 'whole teacher' framework for TPD recommends that the focus is not only on skills and knowledge, but also on attitudes and practices (Chen and McCray, 2012). In measuring the effectiveness of a learning intervention, both the participants' content knowledge and technological pedagogical content knowledge (TPACK) should be evaluated (Voogt et al., 2013), where content knowledge represents skills and knowledge and TPACK is a more holistic view of the efficiency with which teachers exploit technology in their teaching.

Lavonen et al. (2012) examine in-service teachers' adoption of new technology and reflect on the process through the theoretical lens of technological diffusion (Rogers, 2003). The teacher will accept an innovation gratefully, provided that it is easy to use and it adds value to the subject. The adoption is further promoted when the society also notices the benefits (visibility). A well-planned technological innovation also meets the demands of self-determination theory (Gagné and Deci, 2005; Lavonen, 2008):

- A user desires his autonomy, competence and group cohesion to be strengthened through use of new technologies;
- A user is involved because he finds the activity interesting and derives spontaneous satisfaction from the activity itself;
- A functional innovation has a high level of availability, with the following features: easily learned and remembered, efficient, faultless and pleasant to use.

Sinclair et al. (2011) pay special attention to the teacher's attitude. They state that a teacher's negative attitude is reflected in the attitudes of his students. The more constructive and exploratory the attitude of the teacher, the more the students are challenged to enter into discussion, particularly in an open atmosphere during mathematics lessons. Consequently, researchers emphasize the importance of the teacher's attitude in successful technology education. Sinclair et al. (2011) recommend collegiality and co-teaching as one means for teachers to reach the required level of self-efficacy. In addition to these affective factors, the situational aspects of the teacher's learning also have to be taken into account. The school and classroom context, the available resources, e.g. computers and technical support, and in-service training options all have an effect on learning. In Finland, the curriculum change was applied suddenly, before the introduction of appropriate in-service training opportunities. Therefore, several of the publications for this thesis examine the Code ABC MOOC, an in-service training MOOC for Finnish mathematics teachers.

This MOOC aims to build on the existing, well-functioning mathematics syllabus by exploiting and transferring this knowledge for programming skills. The exploitation of prior knowledge is expected to create positive feelings of self-efficacy from the very beginning. Consequently, the TPACK model has been exploited in an attempt to occupy the newly-created space between mathematics and computing. The aim of the MOOC was to increase both content and pedagogical knowledge. In particular, the MOOC focuses on a smooth transfer between the two disciplines by highlighting the similarities of content knowledge and providing stimulating exercises which encourage the teachers to reflect on the FNC-2014 curriculum enhancement. The change in the teachers' perceived self-efficacy is one metric for assessing the MOOC course learning outcomes. Kennedy (2016) talks about enactment problems in bringing new programming skills into the classroom context after attending a professional development course. She highlights the gap between the course set-up and the actual teaching context of a real classroom. For practicing in-service mathematics teachers, good self-efficacy in mathematics is assumed to lower the transfer threshold and facilitate the transfer of mathematical knowledge to computing. However, further research will be needed to analyze the long-term effects of the MOOC, e.g., a follow-up study on how many participants actually started using the learned material and skills in the classroom would be extremely pertinent. As Kennedy (2016) points out, real enactment in the school context is the final test.

The MOOC has a Q&A discussion forum that employs functionalities of a social networking site (SNS) for math teachers, and this is why it can be regarded as an interactive value creation forum. In any SNS, the dominant users create relevant content to appeal to a mass audience, but a more typical user assumes an information-seeking profile (Bechmann and Lomborg, 2013). Trust et al. (2016) refer to professional learning networks. The authors state that the shifting technological landscape requires new knowledge, skills and attitudes. Professional learning networks offer a resource for teachers who wish to satisfy

their diverse, interconnected, and holistic TPD needs. Many previous TPD studies have emphasized that prolonged interventions, combining reflective practices with the learning process, follow-ups on a regular basis, and co-learning with other teachers definitely have a favorable effect on learning outcomes and their long-term effects (Avalos, 2011).

### 2.4.1 Metaphors and paradigms shape perceptions

In determining the role of computer science in education, various metaphors are used, e.g. computer science as literacy, a maker mind-set, or grounded mathematics (Burke, 2016). If the literacy metaphor is used, then programming as digital literacy emphasizes the same logical skills as those applied in constructing linguistically correct sentences, such as utilizing and/or/not in order to express the internal logic of a sentence. From a ‘maker mindset’ perspective, the programming language should be as productive as possible, with a low learning curve, and a ‘low floor and high ceiling’, which implies the use of visual programming languages such as Scratch. Other studies, however, have questioned the benefits of Scratch in developing problem-solving skills and good programming practices (Gülbahar and Kalelioglu, 2014; Meerbaum-Salant et al., 2011).

In addition to metaphors, programming paradigms are essential in defining the angle of approach in teaching programming. Each paradigm has its own command set and programming technique, which leads to different kinds of implementations and programming styles. Each paradigm also has its own strengths; some problems are easy to solve with one paradigm, but another paradigm may be more efficient or flexible in other contexts. Consequently, there are already regular arguments about ‘the right paradigm for the job’. In order to make sufficiently informed decisions about which language and paradigm to select, the decision-makers should have an adequate understanding about the alternatives available, and their implications.

The division of programming languages into different paradigms is not easy, and multi-paradigm languages further blur the categorizations. Wegner (1989) divides languages simply into two fundamental categories of imperative and declarative languages. In this division, the imperative paradigm is largely comprised of procedural, object-oriented and distributed (parallel) languages, whereas the declarative one consists of functional, logical, and database languages. However, a distributed, parallel paradigm does not fit with, for example, an object-oriented paradigm, which may well also be implemented in parallel. Furthermore, Bal and Grune (1994) propose the former sub-categories of procedural, object-oriented, functional and logic paradigms as the main categories.

The categorization is further challenged by multi-paradigm languages constantly increasing in number. Therefore, instead of paradigms, languages may be categorized based on the supported features. Jordan et al. (2015) recommend a feature model, arguing that the current categorization is too vague to help software engineers (and educators) assess the suitability of a language for a particular project and purpose. The feature model has thus been developed based on an analysis of the actor, the agent, the function, the objective, and the procedural programming languages. These features encompass type systems, mutability/immutability, input/output systems, the declarativeness of expressions, metaprogramming, and considerations of concurrency and modularity. Much like Jordan, Van-Roy and Haridi (2004) categorize languages based on their declarativeness and expressiveness. The more fine-grained feature model of Van Roy (2009) defines declarativeness as a horizontal axis and adds features such as procedure, state, closure, port and thread in ascending order of complexity and descending order of declarativeness.



### 2.4.2 Transfer between mathematics and CS

To foster successful transfer, a teacher should emphasize the common underlying conceptual bases (Jarvis and Pavlenko, 2008), in this case those of mathematics and CS. In general, a successful transfer correlates with already acquired expertise: the more knowledgeable the learner, the more well-rounded their skills and the more flexible their mental models, the more readily they will adopt the new knowledge (Bransford et al., 2000). An expert finds analogies by exploiting their previously-constructed knowledge. Without too much effort, an expert is capable of identifying the significant points of the new material, and can thus learn faster and cope better with open-ended problems, these being the cognitive hallmarks of expertise (Billett, 2001). A novice, on the other hand, can become bogged down by the amount of data and may concentrate on irrelevancies. In defining the concept of expertise, the Gestalt psychologists (e.g. Köhler, 1970) refer to the insight experience that helps learners find the right solutions intuitively and enables them to predict outcomes in new situations.

A transfer may happen either near or far, laterally or vertically (Gagné, 1965), or by the low road or the high road (Perkins and Salomon, 1988), all of which imply a certain hierarchy of learning. As the terms suggest, ‘near’ and ‘low road’ transfer occur semi-automatically in similar contexts, whereas ‘far’ and ‘high-road’ transfer only occur once the similarity has become clear after a process of abstraction. Sometimes far transfer also suggests an element of innovation (Butterfield and Nelson, 1991, inventive transfer) or creativity (Haskell, 2000, creative transfer), especially when the transfer engenders new concepts. In addition, Rich et al. (2013) state that one of the complementary subjects tends to be interpreted in learners’ minds in a more abstract manner while the other encourages the learner to focus more on the application. In most comparisons, mathematics is regarded as being more abstract than computing, which is regarded as being a type of applied mathematics (Dijkstra, 1982). In mathematics, educators have long talked about procedural and conceptual knowledge (Gray and Tall, 1994). Procedural knowledge consists of well-internalized mathematical routines, ‘processes’, and these may form concepts if they are explicitly abstracted. Conceptual knowledge, on the other hand, comprises possession of the relevant concepts and their relationships. It is assumed that the practice of both mathematical routines and concepts can provide appropriate bridges for programming learning interventions by exploiting transfer mechanisms.

Transfer between mathematics and computing is streamlined by bridging the gap between corresponding concepts in mathematics and CS. Bridging includes fostering the transfer by clearly explaining the similarities between the concepts. Convergent cognition exploits the synergy of teaching two complementary disciplines in sync: the bridge between them is supported with trusses, such as an instructional framework that highlights the link. Rich et al. (2013), however, claim that the convergent-cognition approach requires an adequate intellectual maturity. Similarly, the method of deliberate practice proposed by Ericsson et al. (2006) is intentionally aimed at elaborating the content to seek analogies and to build connective cognitive links. According to Lehtinen et al. (2014), this implies ‘*conscious effort, a great deal of thinking, problem solving, and reflection for analyzing, and conceptualizing*’. Explicit abstraction raises the level of perception in order to recognize the analogies despite minor deviations in details (Perkins and Salomon, 1988). In this thesis, the transferability of prior knowledge is anticipated to help in-service trained mathematics teachers to learn computing, catalyzed by the similarity between mathematics and functional programming.

## 3 Research methods and theoretical frameworks

The chapter reports the methods exploited, but it first addresses the broader epistemological underpinnings of FNC-2014, such as socio-constructivism, that are also inherently embodied in the studied learning solutions, e.g., in in-service trainings, where the integration with mathematics teaching is also influential. In overall, the publications employ mixed methods and design-based research. The methods used will be introduced under the relevant research questions in a more detail. The method- and data triangulation of the included publications add to the reliability of the results. Moreover, the research findings are in compliance, thereby confirming each other.

RQ1:CT, ‘How to integrate CT into the mathematics syllabus?’, examines the integration of CT by exploiting the shared practices in mathematics. The data collection comprises a survey, interviews, essays, a questionnaire and the specification texts for the CS curricula and syllabi in secondary and higher education. The informants ranged from school students (Y10) to teachers and SW engineers. Taken together, the literature reviews in the publications provide an in-depth overview of the variations in the definitions of CT. Only a few of the curricula mentioned CT by name, so the associated skills, such as algorithmic and logical thinking, are referred to instead. In-service trained teachers and educators are seldom CS experts, so their reflections mainly echo the learned content, and unique and original ideas are rare. However, if they have already gained first-hand teaching experience, some of the course topics might sound impractical, whereas the verisimilitude of other topics to their own experience affirms the topic’s value, and this is manifested in their reflective essays. The qualitative data of the essays is further elaborated as a hypothetical learning trajectory of computing, where algorithmic thinking in particular comes into focus.

The views of the students (PIII), teachers (PV,VI,VIII), and engineers (PII,VII) contribute to the research findings. Their interviews are transcribed, and these and other survey data are analyzed with the content analysis means of classification and thematization. The analysis ultimately targets finding the entities and relations of meanings and combining it as a common narrative (Vilkkä, 2005), embodied as a learning trajectory of CT in this thesis. Hsieh and Shannon (2005) divide content analysis orientations into conventional, summative, and directed one in an ascending order of the eminence of theory. The orientation here is summative: the results are reflected on a dialog with previous CT models, still remaining sensitive to meanings proposed by the informants, such as engagement through creativity and authentic self-expression, and the prominence of specificational thinking concerning engineers. Reliability is pursued by reviewing the results against computing syllabi abroad and the recommendations set by ACM/IEEE.

To respond to RQ2:CS, ‘Which are the CS fundamentals that suit mathematics education best?’, the K–12 syllabi of different countries are compared, and the comparison is stretched to higher education by taking into account the recommendations of the ACM and the IEEE. In addition, the evaluation of the most useful and suitable topics from practicing SW engineers and mathematics teachers enriches the analysis. Finding the most prominent CS fundamentals is significantly easier than the definition of a thorough CT model. The CS concepts and teaching practices are already established. Technical universities share a more or less common view of how the basics of CS ought to be taught, even if the languages and paradigms vary. The prominence of the CS concepts are evaluated simply by their frequency of occurrence in the quantitative data analysis. In the qualitative analysis, the teachers’ authentic voices are heard by selecting any especially enlightening quotations. Even though the order of prominence of the CS fundamentals is clear, in the MOOC, the geometry-related, creative exercises rank surprisingly high, because of the motivational boost they are capable of providing.

RQ3:TCH, ‘How to train in-service mathematics teachers as computing teachers?’, approaches the question of the appropriate means of training teachers from various angles. For instance, how effective is the MOOC for in-service training? How can we assess the value of the informal learning which takes place online? The MOOC feedback, surveys and essays provide a plethora of data to be analyzed. The data illustrates the teachers’ sentiments and concerns regarding the changes in their job descriptions, as well as the programming languages and paradigms they have to deal with. In evaluating the suitability of the taught material, and a paradigm selection, easily transferable knowledge that complements teachers’ prior knowledge is considered a beneficial approach.

Each individual teacher functions as an independent agent of their own learning. However, in knowledge building and professional development, it is not just the individual’s personal objectives which are of importance, but the more general requirements of society as a whole must also be taken into consideration, such as employability issues. The background theories used to evaluate the effectiveness of the in-service training are based on those of ‘teacher professional development’, ‘technological and pedagogical content knowledge’, and ‘adaptive expertise’. In addition, the data analyses presented here try to capture what is worth learning by observing the data through the overall theoretical lens of curriculum theory and research. The following sections introduce epistemological underpinnings implied by FNC-2014 as a research target, the underlying methodology of design-based research, including the research data, the analyses conducted, and a summary table of the publications referred to and the corresponding methods.

### 3.1 Socio-constructivism as an underlying epistemology of FNC-2014

The studied and developed learning solutions in this thesis are directly or indirectly connected to the current curriculum, hence its epistemological rudiments are influential. As in the previous curriculum, FNC-2004, the dominant learning theory in FNC-2014 is constructivism, in essence socio-constructivism, which underpins a large family of related theories (Kimonen and Nevalainen, 2005; Kuusisaari et al., 2016). In the current version constructivism is only implicitly present. The specification text describes an ideal active learner without naming any particular learning theory. However, Kuusisaari et al. (2016) interpret the curriculum as still mainly relying on socio-constructivism, socio-cultural, and humanistic learning perceptions.

These theories emphasize an individuals' own work in active information seeking and its structuring as consistent constructions, i.e., as schemas and other elaborations. In addition, an active learner ought to be aware of the best means of learning for him, the whole learning process being supported by proper meta-cognitive and self-regulation skills (Piaget and Duckworth, 1970). The cognitive constructivism of Piaget and Duckworth (1970) highlights in particular the process of building schemata and the increasing complexity of these mental structures. As a development psychologist, Piaget studied children's cognitive development for the whole of his career and described learning as the construction of schemata by employing two main mechanisms: when a child faces a new concept, 'assimilation' strengthens the schema; a new concept snaps smoothly into the sketch, whereas 'accommodation' deals with the inconsistencies in a schemata that mandate its reconstruction. In particular, Piaget's genetic epistemology emphasizes reflective abstraction as the way to deliberately construct such conceptual knowledge. In summary, cognitive development can be characterized as changes in one's schemata.

In contrast to Piaget's rigid age ranges for children's cognitive development, numerous later research studies have demonstrated that many of Piaget's views to be underestimations, such as his claim of poor conservation (Donaldson, 1978), or false-belief tasks that are manifested as a child's ability to perceive rather sophisticated mental operations, for instance, of settling in a seeker's position when in a play hidden items are moved (Goldman et al., 2012). In essence, challenging Piaget's developmental phases has become so popular among developmental researchers that it led to the dedicated term of 'Piaget bashing', and a canonized procedure: choose one of Piaget's claims about what 6- to 8-year-old children cannot do, design a child-friendlier method, and demonstrate earlier ability (Doherty, 2008). This child-friendly approaches aspires towards more visual and dynamic study set-ups instead of ones that may confuse a child by being too formal or adult-compliant. In mathematics, children's early abstraction skills have been proved, e.g., with visually-aided algebra exercises that they are capable of solving earlier than expected, see Ch. 2.2.1.

Socio-constructivism, however, spotlights the social and cultural aspects of learning more than the internal cognitive processes, which, according to Piaget, happen endogenously while a child grows. In development, Vygotsky and his seminal work depict the prominence of language, connections with peers, and master/apprentice relations as a means of learning, where the zone of proximal development is thought to be especially fruitful (Vygotsky, 1980). As the latest extension of active learning, FNC-2014 introduces phenomenon-based learning in conjunction with other experimental and active learning approaches, such as 'learning by doing' (Dewey, 1902) and 'learning by making' (Papert, 1980). Both emphasize the affective side of learning and aim first at motivating students with engaging exercises. In Finnish educational discourse, Papert has enjoyed a solid reputation from the late sixties, when LOGO was introduced. Indeed, it is only recently that his reputation has started to fade. However, the programming addition in FNC-2014 has revived his reputation. According to Papert's interpretation, computing is applied mathematics, thus these are mutually supportive. Piaget has his say also in the didactics of mathematics by bringing a more 'bricoleur' learning approach, engaging students with a strong hands-on emphasis, and promoting computers as excellent tools for doing this. Papert's interpretation of constructivism is called constructionism.

The spiral curriculum takes the learning content and splits it into suitable and age-appropriate portions for each school year, with each revisit deepening the review (Bruner, 2009). Bruner aims to solidify the learning of even the most complex known issues by iterative revisits *'until the student has grasped the full formal apparatus that goes with*

them’. Similarly, the theory of a ‘learning trajectory’ has its roots in constructivism and active learning theories (Clements, 2002), and shares the idea of a consistent path for a learner to follow. The path consists of well-justified building blocks specified as the result of learning experiments and the selection of the most functional approach. In both approaches, well-designed exercises scaffold the way to abstraction for students.

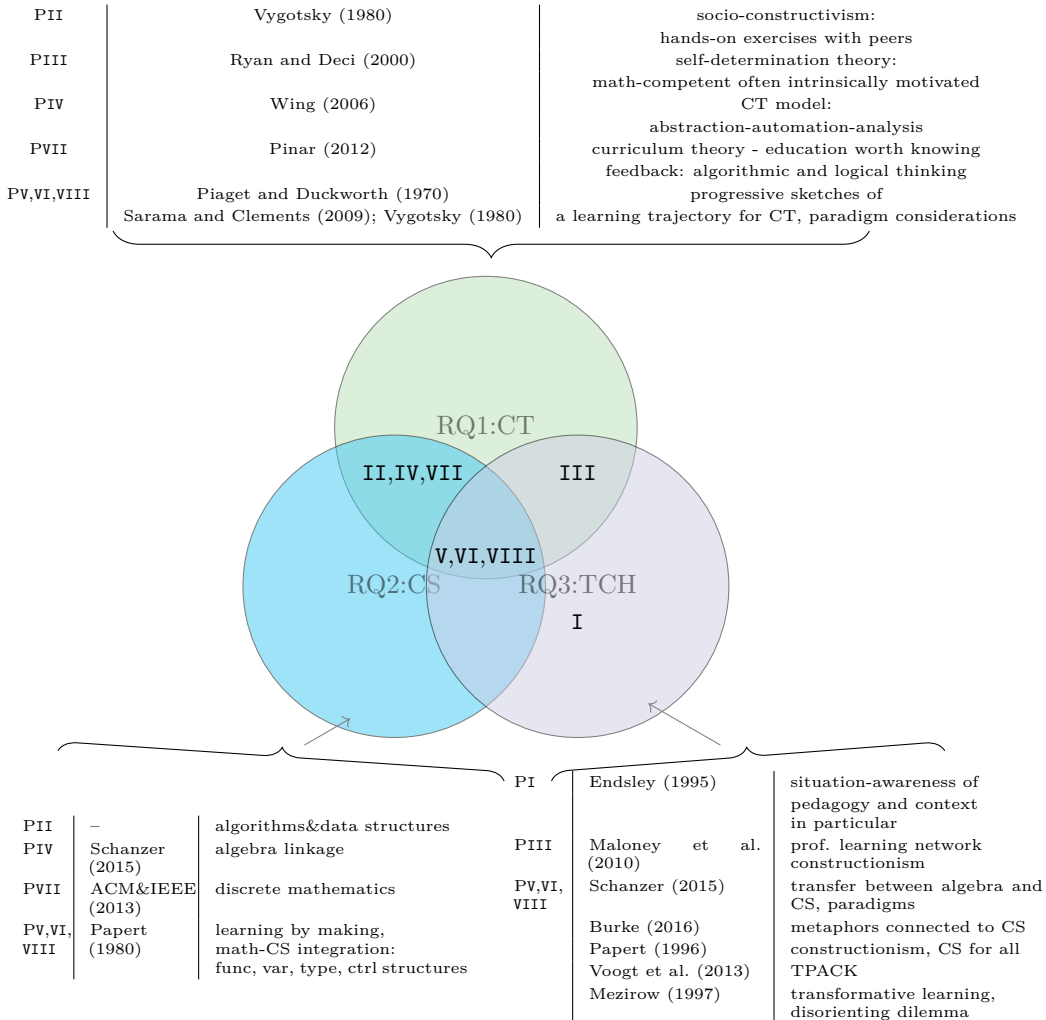


Figure 3.1. Research questions, contributions, and referenced theories

This thesis complies with the learning theories of constructivism, e.g., the theories of Piaget, Vygotsky and constructionism of Papert in Fig. 3.1. Even if the underlying epistemological background has not been stated in each publication, the overall research topic, FNC-2014 and its programming enhancement, implies it. In reviewing previous CT models, Papert is referenced in the publications V, VI, VIII, as sharing the same goals of integrating CS and mathematics. Several of his viewpoints are influential, e.g., considering the affective side of learning, and enriching mathematics with real-life problems solved with peers. A learning trajectory was selected as the means of elaboration, and the most central CS concepts were positioned beside the corresponding concepts in mathematics to be introduced in sync. The hardest work has already been done in specifying the mathematics syllabus and dividing the content into suitable portions for school periods,

which provides attachment points to construct a learning trajectory for CT as well.

### 3.1.1 Mathematics and CS thrive in tandem through transfer

From the educational point of view, algebra is ideally positioned in the overlapping parts of the syllabi of mathematics and computing, and near transfer is the educational mechanism to exploit that synergy (Schanzer, 2015). Transfer happens more easily with no conceptual contradictions and with as clear and explicit a ‘notational machine’ as possible, which in Schanzer’s dissertation implies the construction of robust knowledge structures. In comparing paradigms, math-suitability is assessed as compliance with mathematical rules. Assignment is particularly problematic as a source of ‘mind bugs’: what is  $i = i + 1$  but a false sentence in the mathematical sense. Its full explanation necessitates a comprehensive review of memory issues, even if the intention is to stay at a higher abstraction level. Moreover, out-of-scope assignments of global variables cause side-effects. If the global variables are exploited in Boolean conditions, the vertical line test of function will fail by returning varying values for one input.

The literature supports the hypothesis of conceptual challenges being caused by the assignment. Multiple common misconceptions in computing are related to data handling implying difficulties with assignment. As common misconceptions after the first CS course in higher education, literature itemizes issues, such as storing input and reading data in BASIC (Bayman and Mayer, 1983), instance variable initiating, actual vs. formal parameters in Java (Fleury, 2000), and the memory model, references and pointers, primitive and reference type variables (Kaczmarczyk et al., 2010).

Knowing multiple paradigms is educative. However, the preferred order and, especially, the selection of the very first programming language are divisive issues. The popularity of the language is one justification for its selection, avoiding misconceptions in the context of math is another, and providing forward-compatibility for continuation of CS studies is yet another. With regard to this last consideration, the transition from a more restricted to a looser system is considered easier than the other way around. In particular, removing an assignment from the tool set is later felt to be an inconvenience. A functional paradigm contains a concise command set that is easily mastered throughout, yet it possesses a considerable power of expression. Without side-effects, the purely functional paradigm is the closest to mathematics, or to be specific, a coherent part of it, the paradigm being derived from lambda calculus.

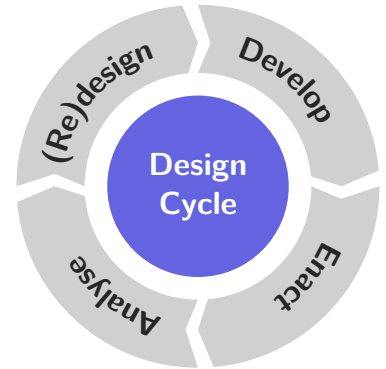
Comparing learning outcomes between, for example, imperative and functional paradigms is cumbersome. The test set-up should not be biased in either direction, thus an extensive exploitation of either variables or functions should be avoided. Algebra – or more broadly mathematics – provides an objective test to measure the outcomes of the integration. As a subject, mathematics has a solid reputation: a number of countries utilize it as an entry criterion in many fields of higher education, such as technological and medical faculties. For example, in South Korea, CS has lost in popularity but mathematics has gained because of its anticipated profitability in a later career (Choi et al., 2015). In mathematics, the progress is more measurable and the assessment criteria are well-known, thus riding on its reputation is considered a viable strategy in establishing the position of CS as well. Counter-intuitively, mathematics teachers were measured to be more successful in teaching CS for youngsters than CS teachers themselves (Schanzer, 2015). Teaching CS is such a new thing that neither proper qualifications for teachers nor distinct learning outcomes have been specified. Besides, mathematics teachers have a head start: they are more experienced in pedagogy and teaching abstract thinking skills analogous to CT.

## 3.2 Design-based research

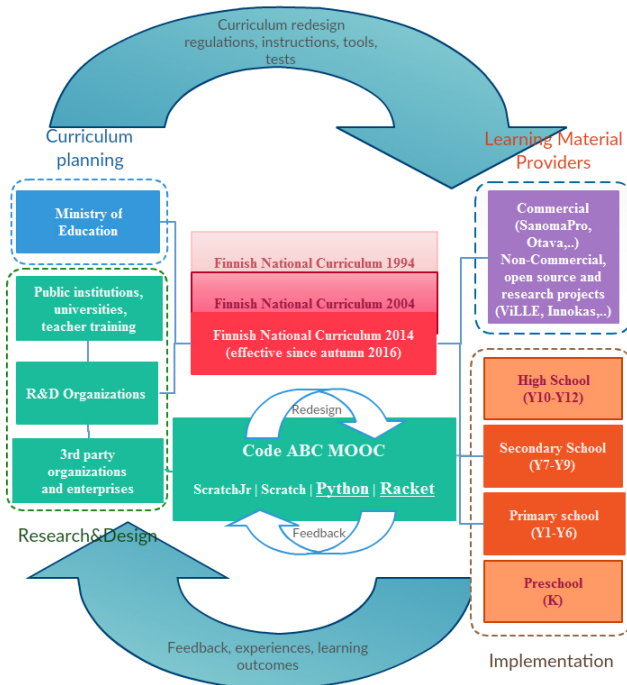
The overall methodology is mainly based on design-based research (DBR), which implies the theory-driven development of educational solutions combined with their empirical proofs (Anderson and Shattuck, 2012; Ørngreen, 2015; Wang and Hannafin, 2005). Suhonen et al. (2012) trace the development of design-based research over the last 60 years, starting with design science, which systematizes the design process (Fuller, 1957), through design research, which investigates the design of man-made artifacts (Archer, 1965), and ending up with design-based research, which is delimited to educational learning artifacts only (The Design-Based Research Collective, 2003).

Design-based research consists of consecutive iterations of design, development, enactment, and analysis, see Fig. 3.2. In education, the design and development phases involve both designers and teachers; the teachers help define the problem, and the designers come up with a solution. The solution is then enacted by the teachers in a real school context, and the feedback from the teachers is analyzed and used as the basis for further development.

It is a continuous feedback loop in which, preferably, the same group of designers and developers have a chance to observe, analyze and re-design the solution based on feedback and research findings. However, in the case of large-scale, long-term projects, the analysis phase is often executed by third-party researchers who are remote from the original designers. Unless these third-party researchers' results are well disseminated, the benefits of the continuous design cycle and the information it provides are largely lost.



**Figure 3.2.** DBR cycles (Collins, 1992; The Design-Based Research Collective, 2003)



**Figure 3.3.** Nested design cycles of curriculum updates (update/10 years) and Code ABC MOOC tracks (2 updates/year)

Involving teachers from early on in the design process ensures contextually fitting learning solutions that address real classroom problems, support teachers' pedagogical practices and add value to the end-users' learning. Thus, the practices of user-centered design may facilitate the enactment of the learning solution.

Fig. 3.3 illustrates two nested DBR development cycles: the outer cycle of curriculum development, which may be interpreted as an instructional artifact, is released in approximately ten-year cycles (Finnish National Board of Education, 1994, 2004, 2014). The inner cycle specifies one particular implementation of a MOOC providing in-service training for mathematics teachers, as this is examined in

several publications of this study. It is iterated in half-year cycles. The outer cycle comprises the phases in the design and publication of the new curriculum by the Finnish National Agency for Education. This includes the development of new learning material by commercial and non-commercial providers, the enactment and implementation of the curriculum in the school context by teachers and principals, the collection of data about the learning outcomes, the analysis and interpretation of the results, i.e. the learning outcomes, and the incremental improvement and fine-tuning of the new curriculum. Thus, the design and development process is a large-scale effort involving a great number of government officers, educators, and scholars. The development of FNC-2014 is understood as a narrative for the whole nation, which can be analyzed and evaluated by all its citizens and organizations. This study, however, is mainly concerned with the analysis of new curriculum requirements, and to a lesser extent the development of learning material and the enactment of the curriculum changes; the subsequent redesign phase will take place in the future. Nevertheless, the results of this study will be disseminated to the respective stakeholders, who hopefully will use this and other completed research to fine-tune the next curriculum.

The inner cycle represents a smaller-scale effort by a handful of volunteers who have created the Code ABC MOOC for Finnish mathematics teachers who are keen to learn the CS basics. This MOOC has been executed once a semester since the autumn of 2015, totaling four iterations during the research period of this thesis. The MOOC was designed to closely follow the Finnish ministry of Education's written specifications for FNC-2014. FNC-2014 is based on the constructivist theory of learning, i.e. the aim is for the learner to construct their own knowledge base. This theory underpins a large family of related theories, all of which emphasize consistent knowledge building by an individual whose knowledge is modeled on schemata and other elaborations. In the MOOC, the feedback is collected after each topic in the MOOC syllabus, and in compliance with the principles of DBR, the iterative feedback is analyzed in order to make further improvements to the content.

The articles about the Code ABC MOOC, Publications V, VI, and VIII, explicitly exploit design-based research, yet the method is implicitly present throughout the thesis. The MOOC comprises four tracks, Scratch Junior, Scratch, Python, and Racket. This thesis mainly concentrates on the Python and Racket tracks, which are designed for secondary schools. However, this study acknowledges and emphasizes the value of Scratch as a primer, especially in the way it improves the self-efficacy of Finnish primary school students.

Most Finnish primary schools make great use of Scratch, which, if necessary, can usefully be extended to the secondary level as well. Publication III discusses the motivational aspects of Scratch, which offers many ready-made learning examples. Besides engaging young children's interest, Scratch is designed as a social networking site which facilitates teachers' professional development by enabling them to network and exploit other teachers' experiences. In addition to FNC-2014, the Code ABC MOOC and Scratch, Publication I studies twelve learning solutions developed by incubating start-ups and small and medium-sized enterprises. In the research project, the design-based approach was prominent: observation data and feedback from the teachers were used to redesign the learning solutions. The entrepreneurial narratives revealed the values and visions that guide the strategy and determine the success of the product.



### 3.3 Qualitative and quantitative data

This is a longitudinal study conducted over a variety of sites, which is reflected in the variety of research data. Not all the data were new, nor specifically collected for this study. For instance, Publication IV compares the computing curriculum specifications of the UK, US, and Finland, and Publication VII exploits the quantitative data of previous mathematics topic evaluations among SW engineers. The other studies, however, utilize interviews and surveys from learning solution developers, computing students, and practicing mathematics teachers in order to collect data specifically for this thesis.

### 3.4 Analyses conducted

The analyses of the study combine both qualitative and quantitative research approaches, which is referred to as mixed-method (Denscombe, 2008). The choice of method depends mainly on the nature of the data, e.g., plenty of numeric data implies a quantitative method, whereas open-ended questions with textual input imply a qualitative approach. Two good examples of quantitative research are Publication V, which analyses the teachers' Likert-scale feedback from the Code ABC track topics, and Publication VII, which uses a numeric evaluation of mathematics topics by SW engineers to cross-correlate and confirm the validity of previous research.

All the interviews were recorded and transcribed. This data, and the answers to the many open-ended questions were examined further using the content analysis methodology. ATLAS.ti, a workbench for the qualitative analysis of textual and audio data, was utilized once in the beginning, while in other cases, the categorizations were either manual or the occurrence frequencies were calculated with the help of Python scripts or Excel functions.

The validity of the main findings of this study is further increased through the use of triangulation. As described in the previous section, the mixture of methodological approaches implies a method triangulation. In addition, the Code ABC MOOC is explored using both numerical data and essays, and by comparing various tracks and exploring free-form discussions on social media. The basic conventions of Finnish curriculum planning are also analyzed with reference to teaching in a foreign culture, in the international school in Cambodia (Publication III). Distancing oneself from the usual conventions may also be interpreted as one type of data triangulation. The fact that different researchers evaluated the same data in comparing the Code ABC MOOC track and analyzing the social media data is further evidence of researcher triangulation.

In some studies, the sample size is too modest for the results to be generalized. For instance, in Publication II,  $N = 7$ , the informants describe the close connection between mathematics and CS and the need for more emphasis on teaching algorithms and data structures. However, with so few informants, the results, although a useful indicator, cannot be generalized to larger populations. On the other hand, in Publication VII these results were confirmed with more comprehensive studies with the statistically significant sample sizes of  $N = 181$  and  $N = 212$ . The same mathematics-CS association was found in Publication IV,  $N = 6$ ,  $N = 16$ , and  $N = 54$ ; the correlation between intrinsic motivation in CS and competence in mathematics seems clear even if the sample size limits the validity of too wide a generalization. With such limited sample sizes, being competent in mathematics is only one possible explanation for students' intrinsic motivation.

The sheer range of methods, data, and informants used for this thesis do manage to capture a rich variety of different viewpoints, and the data and methodological triangulations

---

provide a comprehensive picture of the current situation in Finnish schools, which increases the reliability of the main findings.

### **3.5 Method summary**

Table 3.1 itemizes the publications, the methods, and how each publication contributes to the research questions. RQ1, which handles integrating CT and mathematics, involves a few paradigm-related issues as well. RQ2 extends into the area of discrete mathematics and supports the CS theory basis, as well as pointing out the most prominent CS fundamentals. RQ3, regarding teachers' in-service training mainly exploits the MOOC and SNS feedback, although the first publication also summarizes the views gained from the entrepreneurial narratives of the learning solution developers.

Table 3.1. Summary of the methods used in the cited publications (Python abbreviated as P, Racket as R)

	Data	N	Sample	Method/theory	RQ1:CT	RQ2:CS	RQ3:TCH
I	interviews, success approximated by turnover	12	enterprises	entrepreneurial narrative, content analysis/grounded theory, situation-awareness			pedagogy- and context awareness prominent in success
II	survey	7	SW eng.	content analysis/holistic ICT model	abstraction, user-centered design, modeling	algorithms and data structures	
III	survey, focus group interview, Scratch coursework, grades	6,16,54	Y10 students	survey, focus group interview, comparison of school vs. rnd MIT Scratch projts/SDT theory, intrinsic motivation	motivation, self-efficacy, authenticity		prof.development through ready-made examples and remixing
IV	CS syllabi	3	UKNC, USCC, FNC	content analysis, mind mapping/curriculum research, transfer, paradigms, CT	CT:abstraction-automation-analysis	algebra: var/func UKNC disc. math: logic, sets, matrices	
V	survey	137	math teachers	DBR/transfer, explicit abstraction	algorithmic thinking, problem solving, creativity, geometry exercises engage	function as a main abstraction, recursion challenging	in-service training MOOC, TPD, TPACK, exploitable content knowledge of mathematics
VI	essays, content analysis	206	math teachers	DBR/transfer learning trajectory	abstraction-automation-analysis, logic, creativity	mathematics concepts -> CS fundamentals: func, var, type; basic ctrl structs	in-service training, curriculum reflections, self-made exercises, transfer, metaphors
VII	CS education recommendations and curricula, questionnaires	11, 19, 24	SW profs. 181 212	content analysis, correlation, $R^2$ /curriculum theory	algorithmic, logical and specificational thinking, modeling	algorithms and data structures, multiple representations, logic, sets, stat., prob.	
VIII	course material, MOOC topic feedback	320(P) 137(R)	math teachers	DBR, content analysis, occurrence fq of CS fundamentals/curriculum research, transfer	abstraction: flow chart, filter pattern, functions automation: iteration/recursion analysis: debugging, testing	P: var R: func	in-service training, course feedback, transfer, paradigms, metaphors

## 4 Overview and relevance of the publications

The publications are organized in chronological order except for a slight deviation at the end, where a paradigm-related issue, Publication VIII, the paradigm comparison of Code ABC tracks, is presented last, although it was originally written earlier as an extension of Publication V, the Racket track presentation. The scope of the first three publications is more general and they do not focus specifically on the integration of CS into the mathematics syllabus. Instead, they compare the computing curricula of different countries and the pros and cons of having CS as a separate subject on the school curriculum. Nevertheless, mathematics is a basic tenet of the pedagogical considerations right from the start. For example, Publication II itemizes problem-solving skills practiced in mathematics, and algorithms & data structures as the most useful skills for software engineers, and Publication III hypothesizes that a student's competency in mathematics is an expedient predictor of intrinsic motivation for CS.

The remaining five publications further emphasize the prominence of mathematics. Publication IV highlights the two algebraic fundamentals of function and variable, which serve as the common ground between mathematics and computing. This publication also proposes a number of discrete mathematics topics to be added to FNC-2014. Publication V and Publication VI analyze the feedback of the Code ABC MOOC participants, who are mainly mathematics teachers. Publication V concentrates on the quantitative data whereas Publication VI examines the qualitative data, e.g. the participants' reflective essays. Publication VI also synthesizes the results of the data analysis as a learning trajectory for CT, in which the central concepts of CS are positioned in parallel with their mathematical equivalents. Publication VII reviews the effectiveness of studies on mathematics in higher education and emphasizes the usefulness of discrete mathematics while Publication VIII compares the math-suitability of different paradigms. The Python and Racket tracks of the Code ABC MOOC represent here the imperative and functional paradigms respectively.

Together, the publications contribute to a consistent learning progression in integrating mathematics and CS. They clarify the implications for both the mathematics syllabus and teacher training, and assess the appropriateness of this approach.

## 4.1 Publication I: Significance of pedagogy- and context awareness

### Successful design of learning solutions being situation-aware

This publication was initiated while the author was working as a researcher on the SysTech project, which investigated Finnish small- and medium-size enterprises (SMEs) developing or delivering learning solutions for education. The project aimed to assist such enterprises in developing their learning solutions further by collecting and analyzing feedback from both the users, i.e., teachers, and by providing researchers' expert evaluations of the solutions. In order to better understand the motives and orientation of the participating entrepreneurs, the research group carried out face-to-face interviews with twelve of them and transcribed the interviews. These interviews captured the entrepreneurial narratives and provide much valuable qualitative data for analysis. The data was exploited to identify the predictors of successful learning solutions. In accordance with the grounded-theory approach, the analysis categorizes the themes of the interviews as a conceptual framework of situation awareness, through which the study investigates the orientations of each enterprise.

**Relevance of this study:** The publication sketches out a 'situation awareness' framework to explain why some learning solution developers were more successful than the others. The model emphasizes the necessity of taking into account pedagogy, context, technology, and strategy. The entrepreneurial narratives reveal the strategy- and technology-biased nature of the enterprises, and also the fact that some of them fail to focus on pedagogy and context as much as they should. Only a few of the enterprises were sufficiently aware of pedagogy and context, these being the ones with already established close relationships with schools, or an earlier background in teaching. The lack of sites for piloting material, and the lack of any other feedback mechanisms for testing out their material exacerbated the entrepreneurial developers' inadequate understanding of teachers' needs, which prevented them from developing their learning solutions to their full potential.

The principles of DBR necessitate iterative design cycles in order to improve any learning solution, i.e., feedback should be collected, analyzed and then acted upon. For the entrepreneurs, the SysTech project was welcomed because it guaranteed them feedback on their learning solutions from both teachers and researchers. While the teachers were able to report their experiences as users, the researchers were able to examine the solution against the dominant pedagogical theories behind the Finnish curriculum, socio-constructivism being one of the most influential of these. Being aware of these pedagogical aspects helps the entrepreneurs to discuss their solutions with teachers using the same terms, which ultimately assists them in developing learning solutions most appropriate for mainstream educational use.

This publication is based on general learning solutions, and not only those solutions that target better computing skills. As such, it has a wider scope and provides a model for more generally applicable learning solutions. For example, a subsequent article, Publication VIII, confirms that a more context-aware solution is always preferable to one that is not. The crucial point for the developers to learn from this is the necessity of doing one's homework. The solution must be integrated with FNC-2014 requirements and should be well aligned with the principles behind the currently most accepted learning theories, most prominent among which is socio-constructivism. The main selling point of any proposed learning solution is how well it takes into account the restrictions set by the context.

## 4.2 Publication II: A holistic view of CS education

### Educating future coders with a holistic ICT curriculum and new learning solutions

FNC-2014 emphasizes the importance of digital skills including cross-curricular digital competence, familiarizing oneself with robotics in crafts, and computing integrated with mathematics. However, up till now, teaching computing at elementary level has been limited to rote learning, e.g., drilling the basic control structures such as selection (if-then-else) and iteration (for, while). This approach manages to cover only a restricted subset of the whole range of skills required by a skilled SW engineer (SWE). This study surveyed practicing SW engineers in industry. The cohort consisted of seven experienced professionals (four males, three females) with an average of eleven years in service. Although the cohort is relatively small, the answers to the open-ended questions of the survey revealed a rich source of qualitative data. For example, the respondents stressed the importance of user-centered design, modeling an architecture, and managing the requirements and the schedule as vital skills. This indicates that the current focus of CT might be too narrow, and ought to be complemented with specificational thinking.

User-centered design ensures that the actual needs of a customer are met. This echoes Papert (1980), who refers to the principle of cultural resonance, i.e.: *‘The topic (artifact) must make sense in terms of a larger social context.’* If the developed artifact is aimed at improving a user’s personal environment and living conditions, then user-centered design may also be regarded as a tool of empowerment. Empowered citizens who are sufficiently aware of the wider and emerging needs of society become more innovative. Of course, their innovations require the skills needed to implement them. After capturing user requirements, the SWE’s process continues with modeling the software architecture, which demands both technical skills and conceptual competence. These skills are not exploitable in programming only, but are applicable in deliberate knowledge building, as well, in the form of, e.g., concept mapping.

The article concludes with a holistic model for ICT teaching. Instead of teaching students just the basics of computing, higher-level abstraction and project management skills are also proposed as useful competences for future engineers, referred to as specificational thinking later in Publication VII. Although this study does not specifically focus on the concept of CT, it is implicitly present in the areas of conceptual modeling (i.e., abstraction) and in the craftsmanship of coding as a combination of automation and analysis. This study emphasizes that creativity and the ability to innovate and design new SW systems are at the very heart of SWE.

**Relevance of this study:** Innovativeness and creativity are buzzwords often associated with ‘arts and crafts’ curricula. However, new teaching methods and learning solutions mean that creativity may also be fostered in STEM subjects as well. For example, building robots, making animations, and playing games (e.g. Angry Birds Space to assimilate gravity basics) are new engaging ways of learning. The SW engineers regarded hands-on exercises as a vital method of learning, and games and pair-programming as a means of fostering engagement. From the epistemological point of view, collaborative learning with peers is absolutely aligned with both socio-constructivism and the ‘learning by making’ approach proposed by Papert (1980). According to the experienced SWEs surveyed in this study, the essential skills also include algorithms and data structures. These findings are confirmed later by Publication VII, which evaluates the profitability of higher

education mathematics in one's career. This publication contributes to curriculum theory by examining what content is worth learning, and what is not.

### 4.3 Publication III: A problematic switch from visual to textual

#### **All rosy in Scratch lessons: no bugs but guts with visual programming**

This case study addresses motivational issues and the affective side of learning computing. The study was carried out at an international school in Asia that followed the UK National Curriculum of Computing (UKNC). The study examined the development of different motivations and their impact on learning outcomes. The cohort of this study started computing in Year 8 and used Scratch as a computing primer, followed by Khan Academy's JavaScript and Python basics. Surveys, interviews, and an analysis of the Scratch coursework were employed to examine the genesis of computational thinking.

The Self-Determination Theory (Ryan and Deci, 2000) was utilized to explain the differences in students' motivation, which tend to grow during the school years. The innate psychological needs, the structure of the person's psyche and their skills profile affect the development of intrinsic motivation, whereas extrinsic motivation is mainly shaped by external factors. These factors consist of environmental pressures, such as control and expectations, as well as the grading and other anticipated feedback after completing a task. Alternatively, the after-task reward may be more abstract or remote, for example, approval from the wider community or a position in the desired field. The Intentional Learning Theory (Bereiter and Scardamalia, 1989) considers the long-term objective of gradual knowledge building to be more comprehensive than intrinsic motivation, and is often the preference of the 'serious student' who sets subject-specific lifelong learning objectives.

**Relevance of this study:** This study concentrates on the affective rather than the conceptual side of teaching computing, thus it is not at the very core of the thesis. Its main contribution is to illustrate the importance of language selection, the order in which those languages are introduced, the utilized methods, and their influence on motivation and thus ultimately on learning. If the affective side is neglected, the efforts of educators may be stymied by students' disengagement. The results of this study show that Scratch is a useful tool for scaffolding programming basics and for fostering motivation in all student groups. The discontinuity point from visual to textual programming appears to be problematic: textual programming with JavaScript and Python seems to engage mathematically talented students who have developed intrinsic motivation, while other students may become more disengaged because of their self-perceived incompetence. Those students with authentic interest areas, such as design, animation, or social media, engaged adequately after this transition, but a few students totally lost interest, so the motivational aspects of learning are crucial in planning a syllabus.

In the study, a minor thread elicits a viable way for CS teachers to equip themselves by exploiting ready-made material, i.e., previous projects on the Scratch database. Scratch is designed as a learning tool based on the pedagogical principles of constructionism in that it calls for co-operation by encouraging sharing, contacting and remixing other users. In that sense, Scratch is a deliberate attempt to follow Papert's vision and can be regarded as the successor to LOGO language (Resnick, 2012). However, care must be taken not to get too attached to the preparatory 'logos'. According to the theory of

instrumentalization, a tool must not leave too deep a trace on the action scheme of a student. Instead, visual programming is to be exploited as a stepping stone in the process of developing abstraction to the next phase of ‘lambdas’.

#### 4.4 Publication IV: CS curriculum comparison: UK, US, FI

##### **K–12 curriculum research: the chicken and the egg of math-aided ICT teaching**

This publication studies the links between mathematics and ICT to find the mathematical affordances that can be used in teaching ICT. The study is based on a literature review and a comparison of FNC-2014 with the equivalent curricula of the UK and the US. The most prominent fundamentals are function and variable, both of which a novice will encounter in even the simplest ‘Hello World’ example. On the other hand, for mathematics teachers, algebra that introduces functions is regarded as one of the most challenging areas in the syllabus. To scaffold learning, functions should be introduced slowly and incrementally so that new features can be linked with students’ prior knowledge by exploiting multiple representations. Computational functions may be interpreted as yet another external representation of functions in mathematics.

A variable in mathematics is a straightforward concept compared with its counterpart in computing, whose dual nature (value/address) and the range of possible types are complications. In mathematics, with a real input value, a valid function should return one, and only one, output value, which is also a real number. In computing, functions are more flexible. First, return type may be non-numeric, such as a string. Secondly, functions may return a different value with the same input based on their internal state, and thirdly, they may return no value at all. Despite of the differences, computing has its roots in mathematics, and similarly, it can be perceived as problem-solving, requiring the decomposition of a problem into smaller solvable sub-problems. Throughout the solving process, algorithmic thinking and logic are applicable. These are also the requirements present in FNC-2014, even if CT is not explicitly mentioned there.

**Relevance of this study:** The study describes bi-directional transfer implying that computing integrated with mathematics, notably algebra, would benefit learning mathematics, and vice versa. In fact, algebra learning outcomes could provide an objective means of comparing programming languages and their effectiveness in learning. The fundamentals of algebra, function and variable, and their significance as the synthesizers of mathematics and computing are highlighted. In addition, discrete mathematics modules such as logic, basic linear algebra and set theory would be beneficial and preparatory for further studies of computing, see also Publication VII.

To conclude, this study summarizes the results of comparison between the UKNC approach of having computing as a separate subject and the Finnish approach of integrating it with mathematics. Although there are certain benefits and synergy in integration, a more versatile and dedicated computing syllabus might serve better and give more freedom both in topic coverage and in scheduling. UKNC comprises a multitude of digital skills that are not covered in the FNC, because there is not enough time and/or resources. For example, security issues, the basics of computer and network architecture, and overall fluency with technology are nowadays more comparable with core civic skills. Furthermore, computing as a separate subject would mandate a definition for the required teacher qualifications, which would lead to improved quality of teaching.



## 4.5 Publication V: Transfer of mathematics teachers' prior knowledge

### **Educating computer science educators online: A Racket MOOC for elementary math teachers of Finland**

FNC-2014 includes CS in the revised mathematics syllabus. Consequently, Finland needs to train mathematics teachers to teach programming at the elementary level. This paper describes how the training was accomplished by using a MOOC, the programming language being Racket. The MOOC emphasized the link between mathematics and programming, and exercises that included creative parts were exploited to engage the participants. It aimed to increase both content knowledge and technological pedagogical content knowledge (TPACK). By analyzing the course feedback, questionnaires and exercise data, the study distills the participants' views of the course, and the effects the course has on their professional development (TPD).

**Relevance of this study:** This study contributes to all the main research questions of this thesis, but in particular to the third research question, which concerns the in-service training of teachers. Furthermore, the initial findings of the research reveal mathematics teachers' willingness to learn new skills and their appreciation of the pedagogical considerations in particular: in suitability and usefulness the final pedagogical essay scored the highest. It allowed reflection on the CS enhancement to the syllabus, and its implications for teaching mathematics. The programming exercises were designed to fit an authentic context, i.e., teaching in a classroom setting. As such, at the same time as learning to program, the teachers were encouraged to come up with new ideas for classroom exercises. The first six programming exercises mainly aimed at improving the teachers' knowledge of the CS content, but the final essay and the exercise proposals covered the whole range of TPACK concerns.

The second finding of the study is that the teachers do regard the content of the course as being both suitable and useful for their purposes. The course, especially the geometry-related creativity challenges, generated an appreciable degree of enthusiasm, demonstrating that this type of programming MOOC can provide a motivating and interesting form of professional development for in-service teachers. The third finding indicates that there is a measurable increase in professional development and self-efficacy. However, follow-up research is needed to study the long-term effects of the course, e.g., how many participants actually started to use the learned material and skills in their work. As Kennedy (2016) points out, the final test of such a course's efficacy is the actual enactment of the skills in a real classroom context. Further research needs to examine the suitability of the material from the students' perspective as well, and whether the course gave participants a satisfactory insight into CT. The final essays from the Racket MOOC provide a plethora of data for further analysis, and those results are reported in following Publication VI.

## 4.6 Publication VI: CT/CS integrated in mathematics syllabus

### Computational thinking as an emergent learning trajectory of mathematics

This publication examines the Code ABC MOOC participants' pedagogical views about computing, and how they perceive CT and the fundamentals of CS. The MOOC trains in-service mathematics teachers and equips them with basic programming skills. The study complements the previous Publication V, which analyzed exercises and other feedback instead, such as Likert-scale questions. The data of this study, the reflective essays, were written during the second iteration of the Racket track MOOC in the spring of 2016. The essays are used as a source for enlightening quotations, and the corpus is analyzed quantitatively by scanning the occurrence frequencies of CT/CS topics in the essays, and the mathematics syllabus areas targeted in the teachers' exercise proposals. The teachers' contributions are combined into a hypothetical learning trajectory for CT.

In their reflections on CT, the topics can be categorized into the three 'A's: abstraction, automation, and analysis. In addition to these three 'A's, logic and creativity are also frequently mentioned. Logic includes both the ability to think consistently, and the substance knowledge of logic, such as solving truth values of conditions. Creativity is interpreted both as innovative problem-solving and the option of creating visually appealing geometry art, a classroom practice that a number of the teachers clearly cherish. In consequence, the types of programming exercises most frequently proposed by the teachers were for geometry. Geometry was clearly favored over the more conceptually-adjusted areas of algebra (function, variable) and arithmetic (basic operations, right order, condition primers). In geometry exercises, the visually educational, showy and sometimes serendipitous outcomes were found to be particularly appealing.

Judging by the vague, or non-existent descriptions obtained from their feedback, the teachers had not internalized the fundamentals of CS as well as they had done with CT. This is natural, since the mathematical problem-solving practices that the teachers are familiar with are analogous to CT, but the introduced CS concepts – other than function and variable – are not familiar to the teachers. Controversially, a few teachers considered the integration problematic not so much from the math-teaching perspective, but more from the perspective of teaching computing. Their fear was that the integration could easily taint the teaching of computing, if students started to regard it as intrinsically difficult. Their reasoning was that mathematics already has a reputation of being a hard subject, as exemplified by the following comment: *'Current youth have no interest in mathematics because of too much work (and complexity). Hence, first programming experiences should be as remote to mathematics as possible.'*

**Relevance of this study:** In this synthesized learning trajectory, the CS topics are divided into the layers of CT: abstraction (function, variable, type), automation (selection, iteration), and analysis, (testing and optimization). These layers can be mapped with the mathematical problem-solving practices of first modeling the problem and dividing it into smaller solvables (abstracting), then executing the calculation (automation), and finally evaluating the solution in terms of common sense, and possibly trying to capture a more elegant one (analysis).

FNC-2014 needs to go further in defining what CS concepts and CT skills are required. The Racket MOOC has contributed to this aim by refining its most crucial concepts and skills required to teach CT/CS. By the end of elementary school, the CT skills and

CS concepts that the students have learned should be the same for all students, i.e., standardized. Thus, the required concepts should be agreed and clarified. Ultimately, 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. Accordingly, Publication VII extrapolates the learning trajectory of CT to tertiary education.

## 4.7 Publication VII: Necessary but under-taught discrete mathematics

### Elementary math to close the digital skills gap

Digitalization has shifted employability in the more traditional professions related to natural sciences towards computer science. Many employers complain about the shortage of skilled SW engineers. FNC-2014 aims to integrate computing into the mathematics syllabus in order to make up for this shortage. For this change to be accomplished effectively it is vital that the syllabus in its entirety should be checked and revised to ensure that the content is appropriate for providing our future tertiary-level students with the required digital skills and the fundamentals of CS theory. Prominent CS education organizations such as ACM and IEEE recommend more discrete mathematics. In addition to these recommendations, the feedback from SW professionals and educators is also useful in assessing the value of different syllabus areas. The feedback reveals an imbalance between supply and demand, i.e., what is over-taught versus what is under-taught in terms of the skills which will be required from students in their later working lives.

Critics claim that the Finnish school curriculum currently has too much continuous mathematics, e.g. calculus and differential equations. Indeed, the data from this study reveals that SW engineers need stronger skills in algorithms and data structures. Modeling and fluency with multiple representations are considered valuable for illustrating and structuring data, and these skills also facilitate problem-solving. Logic is required to reinforce the theoretical basis of CS, as are set theory, statistics, and probability, albeit less so. The mathematics and CS syllabi of UKNC provide a useful exemplar for the desired syllabus content in elementary education in Finland, as does that of the United States. The USCC defines modeling in order to abstract and structure data as part of the high-school syllabus, and this topic could even be applied age-appropriately for the elementary level. Modeling is also associated with the use case/requirement specifications of SWE, prompting the coinage of a new term, ‘specificational thinking’. To illustrate the learning progressions from elementary to tertiary education, this publication extrapolates the learning trajectories of Publication VI to sketch out a consistent trajectory for the selected topics throughout the students’ whole academic career.

Curriculum planning is a zero-sum game. If you want more discrete mathematics, then you have to have less of something else. The proposal is to move some of the emphasis in the math syllabus away from continuous mathematics and towards discrete mathematics as early as the elementary level. However, CS as a separate subject allowed more space for new contents. Including all the intended content in the current mathematics syllabus would be extravagant.

**Relevance of this study:** In order to close the digital skills gap, this study examines which areas of math are valued most highly by practicing SW engineers. Specifically, these are algorithms and data structures, logic, and some minor elements of sets, statistics, and

probability. This study proposes that age-appropriate subsets of these areas be already included in elementary education.

Curriculum theory values what is worth knowing. Although from an economic perspective students are mainly regarded in terms of their usefulness for the future workforce, there are other more holistic educational values and ideals which are needed to balance this point of view. Instead of rather short-sighted, ‘targeted’ syllabi which would concentrate on mastering the command set for the currently most popular programming language, the Finnish school curriculum should widen its focus to encompass digital skills from a broader perspective. This means that the capabilities of abstracting, automating, and analyzing are taught in a socially relevant way, and holistically. In short, the curriculum must foster both computational and specificational thinking in tandem.

## 4.8 Publication VIII: Paradigms compared in mathematics-suitability

### Code ABC MOOC for math teachers

Because FNC-2014 integrates computing into mathematics, Finland needs to train its elementary mathematics teachers to teach it. The Code ABC MOOC trains teachers by offering four programming languages that represent different paradigms. At primary school, class teachers teach all subjects including computing, which starts first with unplugged exercises followed by visual coding, Scratch being a de facto standard tool. Textual coding is started at secondary level, and in the MOOC, the Python and Racket tracks are provided for this level. The majority of the Python and Racket track participants were mathematics teachers, as intended. This study compares the math-suitability of these two tracks. This assessment of the courses’ math-suitability is based on the feedback from the participants about the concepts they have learned during the course. This analysis focuses in particular on the implications of the underlying programming paradigms.

Problem-solving is at the core of CT, and is thus closely linked to mathematical thinking. In problem solving, decomposing the problem into smaller subproblems is an essential skill. This skill is also needed in programming, where the subproblems are often implemented as subprograms, e.g., functions. In functional Racket, functions are more substantial than in Python, where variables are conspicuous. To systematize function design, the Racket track introduces Design Recipe, which is aimed at producing well-planned functions (Felleisen et al., 2014). This recipe also promotes test-driven development: unit tests are implemented before a function body. Both the Python and Racket courses emphasize the importance of descriptive naming and comments to improve readability; good coding conventions are understood to be an integral part of CT. In addition to functions and variables, both tracks cover a substantial number of basic programming concepts such as data types, Boolean logic, and conditionals.

The difference in the courses’ math-suitability reflects the differences in the way the courses respond to the target group’s needs, in this case the needs of mathematics teachers. Critics of Python track argue that there is a mismatch between the mathematics teachers’ expectation and what the course delivers, e.g., they think that there is an excess of pixel-wise image operations, which they consider to be irrelevant. In addition, many of the respondents would have preferred more hands-on exercises instead of multiple-choice questions. Racket fared better, and most of the participants regarded the pedagogical essay,

geometric art, Turtle graphics, animation and quizzes as being suitable and interesting, and in scores, suitability and enthusiasm correlate.

**Relevance of this study:** An analysis of the course content reveals a conceptual linkage between the functional paradigm and algebra. Even though Racket is generally regarded as being more challenging than Python because of its syntax (e.g., an abundance of parentheses, and using recursion as a typical mechanism for iteration), the Racket track ranks higher in suitability. This is because the track content and exercises are specifically tailored to fit FNC-2014 and are designed for school mathematics lessons. The material is prepared by a Finnish mathematics teacher who was aware of both pedagogy and context.

The publication covers all the research questions of the thesis: a description of the central CT skills and CS fundamentals, their relevance in the context of mathematics teaching, and the mathematics-suitability of the underlying programming paradigms. Conceptually, a functional paradigm is closer to algebra, as stated by Schanzer (2015). In contrast, the imperative paradigm exemplified by Python contains a number of elements that are far-removed from pure mathematics, and the difference may lead to misconceptions.

# 5 Results and discussion

The chapter combines the main findings of the publications to respond to the thesis-wide research questions. The results imply a CT model and the most crucial CS concepts, which are illustrated in the form of learning trajectories attached to the corresponding concepts in mathematics; see Fig. 6.1. In addition, the results indicate the utility of a MOOC as a tool for in-service training of mathematics teachers and for enhancing the participants' professional development. In training, content that takes into account the pedagogy, the context of mathematics lessons, and is adapted to the new requirements of FNC-2014, scores high in suitability. In addition, the mathematics context can be noticed by selecting such a programming paradigm that aligns best with its practices.

## 5.1 CT as an embedded commodity of mathematics

The CT model of this thesis is constructed based on previous CT models, a math syllabus, and mathematics teachers' essays reflecting on the curriculum; the content being analyzed summatively and divided in corresponding categories of CT. However, in the model review, the myriad of different definitions blurred the view, yet none of them was comprehensive enough to act as a reference as such. In constructing the CT model, the most appropriate parts of the models were taken as a starting point and complemented with the topics emphasized in teachers' feedback. Hence, the CT model combines already existing ingredients with the teachers' views. In their reflections, the mathematics teachers focus in particular on problem solving. CT and mathematical thinking share an analogous problem-solving procedure: decomposition, solving subproblems, and evaluation of the result. In mathematics, however, the iterative nature of solving the problem and optimizing the solution is less prominent than in computing, where testing and debugging are business as usual.

In summary, this thesis defines a matrix-style hierarchy, where CT comprises the consecutive phases of abstraction, automation, and analysis, while the three vertical pillars are algorithmic thinking, logical thinking, and creativity. The CT model is visualized in Fig. 6.1, next chapter. In the model, constructivism is inherited with the adaptation to the mathematics syllabus. Presumably, Papert and Wing could sign the model, if creativity and student-centered teaching methods were emphasized adequately. However, time allocation for the new CS content is not abundant, therefore time-consuming exercises are unlikely. The first two vertical pillars of algorithmic and logical thinking would please the surveyed SW engineers and more widely the interests of industry. Yet in the paradigm selection, these interest groups would favor a popularity-based criterion instead of emphasizing conceptual analogies that implies an imperative paradigm instead of a functional one. The following sections describe the components in more detail.

### 5.1.1 CT as subsequent phases of the three ‘A’s

Problem solving starts with modeling – or abstracting – the problem, after which it is decomposed into smaller solvables. Once the solution is found, the result (e.g., the order of magnitude with reference to initial values) will be analyzed and, dependent on the level of the student’s stamina, a more optimal and elegant method to solve the problem might be sought. Thus, mathematical problem solving resembles the phases of abstraction, automation, and analysis in CT. Abstraction implies modeling a given task, where a function is a means to divide functionality into general-purpose entities structuring a program. A variable is another fundamental computational abstraction, whose importance is highlighted more in an imperative paradigm, where a program’s control flow can be understood as subsequent state transitions that imply an assignment operation. Yet another abstraction is type, which comprises both primitives and more complex data structures. In mathematics, sets of numbers, such as integers and reals, provide an affordance to have recourse to type.

In problem solving, the ability to model and abstract the data is crucial. USCC specifies modeling as a syllabus area of high-school mathematics (Core Standards Organization, 2015, 2017). It combines mathematics, statistics and technology, ‘... 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.’ Modeling requires what is referred to as ‘specificational thinking’, necessary for SW engineers in order to reach a common vision with their customers in defining use cases and requirements.

According to Wing’s definition, CT is the automation of abstractions, i.e., automation comprises an implementation of functions. During the implementation, a developer has to make numerous decisions in order to ensure efficiency and prevent errors. The primary means of controlling the execution flow are selection and iteration. In the functional paradigm, iterations are often realized as recursions or higher-order functions that manipulate lists. In the imperative paradigm, iterations make use of incremental counters. In this fashion, the selected paradigm influences the automation phase and how the data is handled and stored.

Testing, debugging, and optimizing constitute the main means of analysis. In computing, testing and debugging are indispensable; no-one expects an error-free program at the first attempt. In Publications V, VI, and VIII, the Racket track actively promotes the test-driven approach, that is, tests being written before an actual function body.

### 5.1.2 Algorithmic thinking

FNC-2014 does not mention CT in the specification text, but the term is traceable back to the parallel concept of algorithmic thinking, and, for example, Denning (2009) equates these two trains of thought. Publication II regards algorithms and data structures as language-agnostic content that is valuable, irrespective of tools or languages chosen. Publication VII confirms the results: the feedback from SW engineers highlights the prominence of algorithms and data structures in particular, and in addition to logic they are among the rare areas of mathematics in need of more emphasis in higher education.

One has to start small, if algorithmic thinking is introduced already at primary school. In understanding how computers work and how modular programs are to be written, sequencing and subsequent structuring as subprograms are necessary starting points. At

the simplest level, command sequences may be interpreted as primitive forms of algorithms. In fact, learning algorithms does not even require a computer, which makes them more accessible. For example, games provide opportunities for algorithmic thinking, especially if an attempt is made to formalize the game logic (Lamagna, 2015). The acquisition of the very basics is followed by a refining of the solution, e.g., by optimizing the number of steps, resources, and efficiency. As with mathematics, the in-built requirements for an elegant solution run parallel to this optimization process. The customary exercises for algorithms are various searches and sorts, e.g., searching for the maximum or minimum, or sorting numeric values in descending order. Although FNC-2014 lacks formal requirements for specific search or sort algorithms, a few computing syllabi abroad introduce an age-appropriate subset, e.g., UKNC comprises binary search and merge sort (Department for Education, 2015).

### 5.1.3 Logic

Logic is the basis on which the whole discipline of CS is built. According to the feedback from practicing SW engineers, logic and algorithmic thinking are two of the most useful topics that are not taught adequately. In integrating CS with mathematics, Publication VI envisions equations and inequalities, and the truth values of algebra as appropriate primers for logical conditions. Boolean algebra enables the combination of conditions, and the substitution of truth values with bits of zero and one facilitates bit-wise operations and truth tables.

In UKNC, logic is taught comprehensively (Department for Education, 2015; English Department for Education, 2013). In addition to the aforementioned topics, it complements truth values with binary and hexadecimal notations that prompt bit-wise operations of addition and shift, and truth tables. As another representation, Boolean operators are illustrated as logic gates in circuits. If CS were not a standalone subject in the UK, logic gates would fall into the area of physics, more specifically electronics, rather than mathematics.

Moreover, the GCSE aims for students to be capable of translating English language sentences into logical statement. Logic exercises start as word problems that first have to be converted into such statements. Appropriate propositions are formulable only within the student's linguistically mediated grasp of the natural language sentences and their semantics. Hence, as a logic primer, preciseness and apposite language skills are helpful. In working life, such skills stand out in the specification of use cases and requirements. This ability to be specific is referred to as 'specificational thinking'. In native language and digital literacy lessons, logic is exploitable in interpreting the deeper meaning of texts and in writing essays with a strong rationale and complete argument. Techniques such as argument-mapping attempt to systematize one's reasoning and explain the inferential structure of arguments (Davies, 2011). Ultimately, logic is a crucial component of critical thinking.

### 5.1.4 Creativity

In pedagogical discourse, a number of psychological and affective viewpoints are attached to CT, such as creativity and innovation, in compliance with Papert's influential legacy (Resnick, 2017). As a LOGO successor, Scratch is designed specifically as a creation tool. Publication III examines Scratch and the effects of visual programming on motivation. The results show that the tinkering approach with creativity options is enthusiastically



received among students, and consequently, the switch from visual to more disciplined and cognitively demanding textual programming causes problems. However, to boost motivation, it is important to engage students by providing opportunities for creativity and authentic self-expression. Similarly, creativity is capable of engaging in-service trained mathematics teachers while they are solving geometry exercises, as shown in Publications V and VI. The exercises comprise turtle moves, computer graphics and animations. In the learning trajectory of CT, creativity is interpreted as an extrapolation of geometry, although this is not the only option for applying creativity in the mathematics syllabus.

### 5.1.5 Discussion of related work

The CT model developed in this thesis complies with the seminal work of both Papert and Wing. Papert acknowledges the mathematical foundation of computing. Beginning with toddlers necessitates awareness of age-appropriate teaching strategies. In Papert's view, the fear of mathematics is the main hindrance to learning, made worse by separating mathematics out as a theoretical subject. Instead, he would rather embed mathematics in everyday life, embracing all activities, such as playing, gaming, and guiding turtles with LOGO.

As a constructionist, Papert is well aware that according to Piaget's genetic epistemology, one has to start with the concrete constructions of 'legos', and proceed through visual 'logo' block snapping to textual programming, i.e., 'lambda' (viz. the functional paradigm). The approach of this thesis is not as panoptic and inclusive as Papert describes it, and the target age group is older, i.e., secondary school students. However, Papert's mathematics-rooted, consistent, and threshold-lowering approach accords with it, and the model resembles the 'problem solving' procedure of mathematics. In addition, computing exercises may be exploited as rich algebra tasks. The analysis phase of computing, in particular, enables more accepting and forgiving practices for evaluating the result than mathematics and its 'technology of grading'.

Both Papert and Wing attempt to spread CS for all. Wing has been an active promoter of CS as a school subject. However, compared with Papert who is a psychologist and mathematician, Wing is an engineer and a computer scientist, who has a more pragmatic approach highlighting the mutual benefits and multi-disciplinary applications of CS. She sees CS as requiring thinking at multiple levels of abstraction (Wing, 2006): abstracting data, code, and the users' needs. In dividing CT into parts, she emphasizes abstractions and algorithms, and describes CT as 'automation of abstractions'. In the CT model of this thesis, abstraction chiefly means modeling a task for implementation. In computing, the design of functions is the main means of abstraction. Type comprises data abstraction.

In descending order of inclusiveness, the ultimate end is the functional paradigm camp after Papert and his world-embracing computational orientation and Wing's profitability-oriented approach. The functional camp highlights computing as a sub-area of mathematics, algebra being the highway to the core (Schanzer, 2015). A function is the most detectable abstraction of a functional paradigm, an enabler of composability, and its design is systematized with the Design Recipe of Function (Felleisen et al., 2014). This camp promotes the educational use of Racket, which is a Scheme dialect, with the added motivation factors of game development and 'algebra of images', i.e., images that can be used as first-class values (Levy, 2013a). In introducing the language, emphasis is placed on its purity; side-effects are prevented by hiding the assignment operations. With this approach, the 'cruelty of really teaching CS' is tangible, maybe as the last nail in the coffin for a math-o-phobic. To counteract this, algebra of images and the Bootstrap

project with game design are provided as tools for engagement (Levy, 2013b; Schanzer et al., 2018).

## Contribution

To sum up, the contribution clarifies the components of CT and explicates their contents in more detail. The CT model divides into horizontal layers of abstraction, automation, and analysis, similarly to SW development having the phases of design, implementation, and testing. Abstraction infers conceptual modeling, such as UML diagrams in SW design. At elementary level, the technique is practicable as a general knowledge building tool of concept/mind mapping, which should be imported in mathematics praxis as well to provide a holistic overview. In the matrix view, the vertical axes are algorithmic and logical thinking, and creativity. Furthermore, the model hypothesizes a preference for the functional paradigm. Paradigms influence the semantics of a programming language and which kinds of abstractions, design patterns, and coding conventions suit best. For example, in imperative languages, counter-based iterations are the basic building blocks, whereas functional languages exploit recursions extensively. Publication VIII points out the differences between paradigms and their implications in mathematics teaching. The CT model, however, is paradigm-agnostic, even if the close conceptual linkage between the functional paradigm and mathematics is acknowledged and implicitly supported by FNC-2014, which integrates CS into mathematics.

Burke (2016) claims that the metaphor connected to computing is influential in positioning CS in a curriculum; he enumerates the metaphors of maker mindset, digital literacy, and grounded mathematics. The referenced literature mainly illustrates two of these orientations: Papert and Wing belong to the maker mindset as CS laymanizers, whereas the grounded mathematics folks of Racket highlight conceptual purity.

## 5.2 Mathematics and CS concept overlap

This thesis presents a hypothetical learning trajectory for CT, and the necessary CS concepts are specified in accordance. To introduce the concepts, the already-established mathematics syllabus provides a suitable schedule and attachment points. Conceptually, algebra is at the center of gravity, but geometry offers a wild card to motivate and engage students, as demonstrated by Publications V and VI. In engagement, creativity seems to be the key and low-threshold/high-ceiling types of tools, such as Scratch, succeed in fostering it while teaching the basics; see Publication III. Publication VII sketches learning trajectories further to university-level mathematics. The feedback from SW engineers reveals useful and CS-supportive mathematics content. Instead of continuous mathematics, they would benefit from more discrete mathematics, especially stronger skills in algorithms and data structures, and logic. Additionally, such areas as sets, statistics and probability are considered useful (Publications IV and VII).

### 5.2.1 Algebraic fundamentals in the core

A few publications address the linkage between algebraic and CS fundamentals, in essence, variable and function. Dependent on the selected paradigm, the concepts deviate more or less from their mathematical counterparts, which can be a source of misconceptions. Publication IV determines the features of the (imperative) variable, such as its identity as a memory store, global versus local scope, and its type often being a non-numeric,

complex data structure instead of simple primitives (*int*, *float*) as in mathematics. The variety of types concerns also the parameters and return types of functions.

Ultimately, assignment is the main complication. Publication VIII raises the peculiarity of the assignment operation of  $x = x + 1$  from the mathematical viewpoint, which reveals the variable's true nature as a memory store and the pivotal role of assignment in distinguishing between the imperative and functional paradigms: in pure functional, neither assignment nor mutable variable exist, in other words, no side-effects are caused. Publication VIII illustrates the control flow of imperative code as consecutive state changes, in contrast to the functional paradigm and its value-passing chain, where a function equals a value. In mathematics, a function must return only one output for an input. In computing, a function may return multiple values for one input, which implies side-effects and mutable data, for example, assignment of a global variable. However, pure functional language does not provide an assignment operation.

In addition to variable and function, the most crucial CS concepts include also type, and the control structures of selection and iteration. Assignment has its implications for the control structures as well: the imperative paradigm employs iterations, whereas the functional paradigm utilizes recursions in the absence of incremental counters.

### 5.2.2 Basics with visual programming

With visual programming, students familiarize themselves with sequencing, grouping commands as bigger blocks, and the control structures of selection and iteration. However, a special emphasis is required to avoid reinforcing the most common misconceptions in Scratch, such as '*loops are forever*' (Armoni et al., 2015). Currently, testing and debugging are poorly supported in Scratch. To self-assess Scratch projects, Publication III recommends making routine checks by exploiting automatic evaluation tools, such as Dr. Scratch. Passing the evaluation with the level of 'developing' or 'mastery' could be used as a prerequisite for a certain grade.

On the other hand, the benefits of Scratch are apparent. Scratch fosters creativity and enables self-expression by implementing projects with authentic goals. Most importantly, visual block snapping prevents students from writing syntactic errors. Blocks function like legos, they fend for themselves: they either fit or do not fit together, which reduces unnecessary cognitive load and the need for debugging. In the beginning, error-free programming is good for increasing students' self-efficacy. However, subroutines, which are called blocks in Scratch, do not function optimally: parameters cannot be conveyed and blocks do not return any value. However, the basis of subroutines is the sequencing and grouping together of associated commands, which, in all its simplicity, constitutes the dawn of algorithmic thinking.

### 5.2.3 Algorithms and discrete mathematics contents

According to SW professionals' evaluation of useful mathematics topics, algorithms and data structures are the most profitable. Publications IV and VII recommend setting clearer learning targets for algorithms, such as a binary search, and merge sort, as in the UKNC. In computing, the most common data structures for storing numbers or other data include arrays, lists, and vectors. Often, the structures provide various in-built convenience functions for set operations that may come in handy, e.g., in iterations. Publication VII highlights discrete mathematics as a necessary support for CS theory studies. In the case of algorithms, for example, logic and sets were advantageous. For

example, naïve set theory provides tools for describing and visualizing set operations, such as unions, intersections and cuts.

#### 5.2.4 Discussion of related work

FNC-2014 is the primary source for sketching the learning trajectory of mathematics. The learning trajectory embodies the learning theories of cognitive constructivism, which state that knowledge is built on top of already acquired knowledge. The trajectory defines consistent progression from simple to more complex tasks. As Clements and Sarama (2004) put it, the learning trajectory is ‘*a conjectured route through a set of instructional tasks designed to engender those mental processes or actions hypothesized to move children through a developmental progression of levels of thinking*’. This study complements the trajectory with a selected subset of CS concepts, i.e., function, variable, type, selection and iteration. Algebra is essential, and its building block, early algebra would facilitate the introduction of principal CS fundamentals already at primary level. However, ‘*early algebra does not mean algebra early*’, instead, the learning material should be aligned with the children’s level of cognitive development (Carraher et al., 2008). In learning, early prompting of topics provides ‘met-befores’ that enhance the readiness for easier comprehension in later iterations (Bruner, 2009; McGowen and Tall, 2010). In regard to the very basics of CS, algebra is at the center of gravity; the synergy is mutually exploitable (Schanzer, 2015).

The mathematics syllabus should be reviewed in its entirety to ensure an appropriate theoretical basis for both natural sciences and computer science in accordance with the changes. For example, the evaluation of the most useful mathematics for SW engineers demonstrates the imbalance between continuous and discrete mathematics. The engineers talk of an excess of continuous mathematics (calculus, differential equations) at the expense of discrete mathematics, which they desire to be emphasized more. Algorithms and data structures are the most needed, followed by logic, sets, statistics, and probability. These concepts are backed up by ACM/IEEE recommendations for an applicable CS syllabus for higher education (ACM&IEEE, 2013). FNC-2014 mostly omits these contents, but UKNC and USCC mathematics and CS provide points of reference.

A proper implementation of mathematics-CS integration would require sufficient resources and time, which currently are inadequate. Preferably, new knowledge would be gained through hands-on experiences and social interactions. Papert promotes learning by making, which complies with the active learner approach of FNC-2014. However, this is time-consuming. Fortunately, the crafts syllabus is capable of complementing it by introducing robotics and automation. There should be an active dialog and synchronization of learning goals between the subjects. Some prominent topics, however, are absent from both the mathematics and crafts syllabi, e.g., safe use of the Internet, the protection of one’s on-line identity, and responsible and respectful on-line behavior, which can be termed as ‘new civics’ and a few of the topics are handled in environmental studies. Having CS as a separate subject enables the UK to introduce a robust theoretical basis for CS, the above-mentioned skills, and in addition, the infrastructure of networks and devices, and a means to improve students’ technological fluency overall.

#### Contribution

The main contribution is the extraction and justification of the most prominent CS fundamentals. Moreover, their position and intended schedule are hypothesized in

accordance with their integration in the learning trajectories of mathematics. In addition, the explication of the most useful, but under-taught mathematics contents aims to give a proper theoretical basis to support CS teaching. Even if the territory of CT models is well occupied, this thesis establishes the niche of transferring CT and the CS fundamentals to elementary education and mathematics teaching, which is globally unique.

### 5.3 Mathematics teachers' professional development

FNC-2014 enhances the job descriptions of practicing mathematics teachers retrospectively by requiring them to teach CS basics. However, in-service training is not provided to the true extent necessary, which obligates teachers to gain the required skills by themselves. MOOCs and the ever-increasing educational resources in the Internet provide immediate self-help. However, the education authorities cannot abrogate their responsibility of training teachers to do their job properly.

#### 5.3.1 Influential in-service training via MOOC

The lack of formal training options triggered a group of volunteers to kick-start the Code ABC MOOC studied in Publications V, VI, and VIII. The MOOC comprises the tracks of Scratch, Python and Racket, where Scratch targets the primary level, and Python and Racket target the secondary. The MOOC feedback comprises both quantitative and qualitative data. After each topic, the participants rate its level of challenge, inspiration, and suitability. Publication V reports mainly the quantitative results of the MOOC participant surveys; Publication VI, in turn, concentrates on the qualitative data, the essays written as the last reflective exercise, and merges the teachers' views into the learning trajectories of mathematics enhanced with CS fundamentals.

Publication VIII compares the Python and Racket tracks of the MOOC by focusing on the underlying programming paradigms. The trends in the scores are contradictory for the Python and Racket tracks. Python is taught imperatively and its scores decrease topic-by-topic, whereas in Racket, the scores increase, apart from for the low-scoring recursion. Due to its complexity, recursion is not considered suitable for elementary level. However, the results do not directly indicate the superiority of Racket over Python, but they show that other factors, such as teaching methods, are also influential. The multiple-choice questions of Python are considered a pedagogically weaker alternative than Racket's hands-on exercises. The teachers are highly motivated to learn, and filling multi-choice questions simply does not suffice as their anticipated programming exercise. The low scores reflect the teachers' frustration.

The comparison of the Python and Racket tracks demonstrates that both the pedagogical and contextual aspects should be taken into account: the course should improve content knowledge (CK) as well as technological and pedagogical self-efficacy (TPACK) (Voogt et al., 2013). This finding confirms that situation awareness, especially being aware of pedagogy and context, is a useful predictor of the successful development of a learning solution, as suggested by Publication I. Originally, the Python material was translated from the American CS4All projects (Ericson et al., 2015, 2016), whereas an experienced mathematics teacher having a strong background in SW prepared the Racket track material, which was tailored to fit the FNC-2014 mathematics syllabus in particular. This difference is clearly manifested in the suitability scores.

### 5.3.2 Exploitation of prior knowledge

Bridging prior mathematical knowledge for the benefit of computing is achieved by exploiting the mathematics teachers' especially suitable background. The more analogous the concept, the smoother is the transfer. Transfer can be near or far, where near transfer implies closer conceptual connections within one domain and far transfer means applying skills between domains (Barnett and Ceci, 2002). The grounded mathematics approach considers CS as a sub-area of mathematics, which highlights its mathematical nature and implies conceptual analogies, as with algebra and the functional programming paradigm. Accordingly, Publications V, VI, and VIII refer to the pedagogical theory of transfer, and Publication VIII ends up recommending the functional paradigm, after sketching the smoothest path from mathematics to computing without disconcerting misconceptions.

If students are often math-o-phobic, their teachers, in turn, are tech-o-phobic, which becomes apparent by reading their essays. Publication VI itemizes the teachers' fear of technology and of 'not learning quickly enough' – or at all – causing feelings of incompetence, frustration, and shame, which reflects the disorientation dilemma of adult learners (Mezirow, 1997). One of the MOOC track leaders speculated that an aptitude for technology may even have set the direction of one's studies: if strong, a career as a SW engineer is chosen, otherwise, that of a natural scientist or a teacher. If this hypothesis is accurate, the current situation of integrating CS into mathematics lessons is especially worrying, 'what goes around, comes around'.

The change in teachers' job descriptions, languages, paradigms and other pedagogical viewpoints are vividly discussed in social media, and following these discussions improves a participant's situation-awareness. For instance, language/paradigm selections, and underlying metaphors associated with CS, cause controversies. The borderlines can be sketched between imperative and functional paradigms, or alternatively, between the metaphors of maker mindset and grounded mathematics. In addition to implying what are the best-fit languages and tools, these perceptions infer the appropriate target group, i.e., whether CS education should guarantee good computing skills for all, or exclusively for future SW engineers.

### 5.3.3 Discussion of related work

Papert (1980) hypothesizes that engineer-led development leads to the alienation of non-engineers: *'the people who make decisions about what languages their computers will speak, generally engineers, find (typical programming languages) easy to learn. Thus, a particular subculture, one dominated by computer engineers, is influencing the world of education to favor those school students who are most like that subculture. The process is tacit, unintentional and it has never been publicly articulated, let alone evaluated.'* To counteract this, Papert and his group offer easily approachable computing tools for everyone, first LOGO, then Scratch as its successor. Furthermore, Scratch has been designed with the principles of constructionism in mind (Maloney et al., 2010; Resnick, 2012), thus it is providing all the facilities for connecting, commenting, and co-creating in order to become a fully-fledged professional learning network.

FNC-2014 states that primary schools should utilize visual programming and secondary schools should move forward to textual programming. To the relief of all, Scratch is currently a de facto standard in visual programming. However, there is no such consensus of opinion for textual programming. The transition revives the question of a fit language and programming paradigm. Racket is an exemplar of the functional programming

paradigm. It is the functional purists who have set the guidelines for Racket's educational use, e.g. a removal of an assignment and other unorthodox features of the language. For instance, in their book 'How to Design Programs', Felleisen et al. (2014) introduce a purely functional procedure to be followed. However, this approach lacks accessibility. Even mathematics teachers struggle with the language and with the concepts, which decreases their feelings of self-efficacy. In their essays, they talk about shame and even self-loathing because they are not performing up to their own expectations. Thus, it appears that digitalization, the new curriculum requirements and the pressure to perform may be causing a disorienting dilemma (Mezirow, 1997) or at best a mildly reorienting one.

Despite the enthusiasm with which the functional paradigm purists promote the grounded mathematics approach, the maker-mindset and digital-literacy brigades are equally convinced of the benefits of their approaches. In his article, Burke (2016) has examined the rhetoric of introducing CS in K–12 education, whether it is associated with mathematics, technical skills, or literacy. Regarding computing as a technical skill underlines its applicability, e.g., in the construction of artifacts. Educators of digital literacy emphasize critical thinking and discerning the role of technology in society overall. For example, modern digital technology has been used to manipulate people's opinions, through algorithm-aided targeted news and the communication bubbles that arise between like-minded people in social media. Digital literacy comprises computer literacy (Hoar, 2014), digital literacy (Scalise, 2018; Watson et al., 2014), code literacy (Dufva, 2013; Vee, 2013), and/or software literacy (Khoo et al., 2017) as the emerging new literacies of the 21st century.

This thesis considers the dilemma of positioning CS in the school curriculum. It is somewhat analogous to mathematics: everyone should learn the basics, yet not everyone is going to become a mathematician. Whatever is the targeted level or ultimate content, consistent progression and a clear conceptual base benefit all. A wider range of students can be reached with more accessible options, thus those approaches that appeal exclusively to mathematically-oriented students should be complemented with other solutions. The all-inclusive strategy is better realizable as part of a crafts syllabus (robotics, one's own artifacts) that complements computing integrated in mathematics, or by introducing CS as a separate subject.

## Contribution

This thesis aims at improving mathematics teachers' content knowledge and TPACK, by explicating CS fundamentals and situating them in the learning trajectories of mathematics. Moreover, the underlying paradigms and their implications for mathematics teaching are explained. To comply with the DBR method, the publications of this thesis provide feedback from the research side to be disseminated in the next increments of curriculum development and in-service training of mathematics teachers. Current resources for teacher training are inadequate. In the transition phase, the scarcity of training resources is excusable, however, the current state of affairs should not become the new norm. As complementary alternatives, MOOCs and other social networking sites can supply the training that mathematics teachers need. In addition to clarifying learning targets, the Ministry of Education ought to specify the required qualifications for teachers to ensure the quality of teaching. This thesis has contributed to the in-service training of mathematics teachers by defining and incrementally developing the Code ABC MOOC and FNC-2014-compliant content.

## 6 Conclusions

The results of this study provide clarifications of FNC-2014, especially in regard to mathematics, CS, and their integration. The most crucial CS concepts are extracted and linked to form consistent continua for the learning trajectories in mathematics. The main themes of CS in FNC-2014 are deliberately sketched out as a trajectory from legos and logos to lambda. For example, first computing experiences are often gained with a tangible construction series and unplugged exercises. Taken a step further, visual programming enables the construction of simple animations, stories and games by simply dragging virtual blocks to construct the code. Visual programming is currently almost synonymous with Scratch, the online learning environment developed in the spirit of constructionism and inspired by Papert's LOGO. After moving to secondary education, the accumulated concrete operations of primary school will be explained and explicitly abstracted, which raises the procedural learning to a more conceptual level.

In addition to students, mathematics teachers as novices in computing go through similar phases of the genetic epistemology of computational thinking. However, they have their prior mathematical knowledge to exploit, which paves the way in particular to lambda calculus and its computational derivatives. The computing addition to the syllabus triggers the need to train in-service mathematics teachers. This study presents a viable approach to in-service training based on exploiting mathematics teachers' prior knowledge and bridging it with the programming paradigm most analogous to mathematics, i.e., the functional paradigm. Further studies are needed to identify which strategy works best, that is, which language, paradigm and subject combination produces the most appropriate know-how.

### 6.1 Implications for FNC-2014

**RQ1: How to integrate computational thinking into the mathematics syllabus?** The CT model of this study is divided into the layers of abstraction, automation, and analysis, while algorithms, logical thinking and creativity embrace the whole process. Abstraction interlaced with problem solving in mathematics necessitates modeling and decomposition of the problem. Automation implies the implementation of abstractions via the utilization of functions, variables, and the structures controlling the execution flow. The analysis phase in computing comprises testing, debugging, and optimization. In a test-driven SW process, the analysis phase comes to the fore since the tests are written even before the actual function body. This definition of CT resembles the design cycle of the DBR method, where abstraction equates with design, automation with development and the enactment and analysis phase is the same in both. In addition, it has similarities with the software development process, including the design, implementation, and testing phases. In mathematics, the analysis phase is, however, less prominent, and consists of



evaluating the sensibleness of the result and the elegance of the intermediate phases and the solution.

Practicing algorithmic and logical thinking should be started as early as at primary level, even without digital devices. Children can imbibe CS basics by playing games or doing other unplugged exercises which may include multidisciplinary approaches such as using natural language sentences as a playground for logic. Creativity and innovation are the prominent engagement factors in motivating both students and in-service teachers to learn computing. CT can provide authentic opportunities for self-expression, e.g., when implementing artifacts. Creativity and innovativeness are the buzzwords for 21st century skills.

**RQ2: Which are the computer science fundamentals that suit mathematics education best?** The most essential CS fundamentals are function and variable. Variable implies a concept of type that can be either a simple primitive or a more complex data structure. In addition, the most expressive control structures, i.e., selection and iteration, should be introduced early on. Of all the mathematics syllabus areas, algebra is the one closest to the conceptual basis of computing; it shares the same fundamentals and also facilitates algorithm development. Algebraic fundamentals are most unambiguously transferable to functional programming, even if the differences in syntax, e.g., the Scheme-like prefix notation and an excess of parentheses, may blur the picture. In addition, the theoretical basis of CS should be supported with more discrete mathematics, i.e., the basics of algorithms & data structures and logic, and, optionally, sets, statistics, and probability. Multiple representations are helpful in modeling a problem, for example, visualizing the data as Venn diagrams or tree-form graphs. In an attempt to close the digital skills gap, discrete mathematics should be emphasized more at the expense of continuous mathematics.

**RQ3: How to train in-service mathematics teachers to be computing teachers?** In educating teachers, bridging their prior mathematical knowledge with computing basics fosters transfer and complies with Papert's continuity principle, which states that '*mathematics must be continuous with well-established personal knowledge from which it can inherit a sense of warmth and value as well as cognitive competence*' (Papert, 1980). In addition to conceptual learning, the affective domain of learning is influential. The in-service mathematics teachers who took the MOOC and provided the data for this thesis were inspired by the opportunities for creativity in implementing their own designs, as were their students. Thus, when evaluating the most suitable mathematics areas for integration, geometry ranked higher than its conceptual weight in the teachers' exercise proposals; explained by the fact that the teachers recognized geometry's value as a motivator. When formal in-service training resources prove inadequate, they can be complemented with more informal learning provided by professional learning sites, such as MOOCs.

Is mathematics a good choice for computing integration, then? – Yes and no. As a discipline, CS has its roots in mathematics, and its further development requires a robust theoretical background in mathematics. For example, the links with algebra facilitate the knowledge transfer (Schanzer, 2015). For its part, computing offers a new angle of approach to teaching mathematics by providing attractive visualizations of data and the power to handle elaborate calculations. However, integrating CS with mathematics has its down-sides. Even if mathematically- or technically-oriented students might become engaged more easily, those students who are already struggling with mathematics (its reputation for being 'hard') might fail to engage with CS precisely because of their

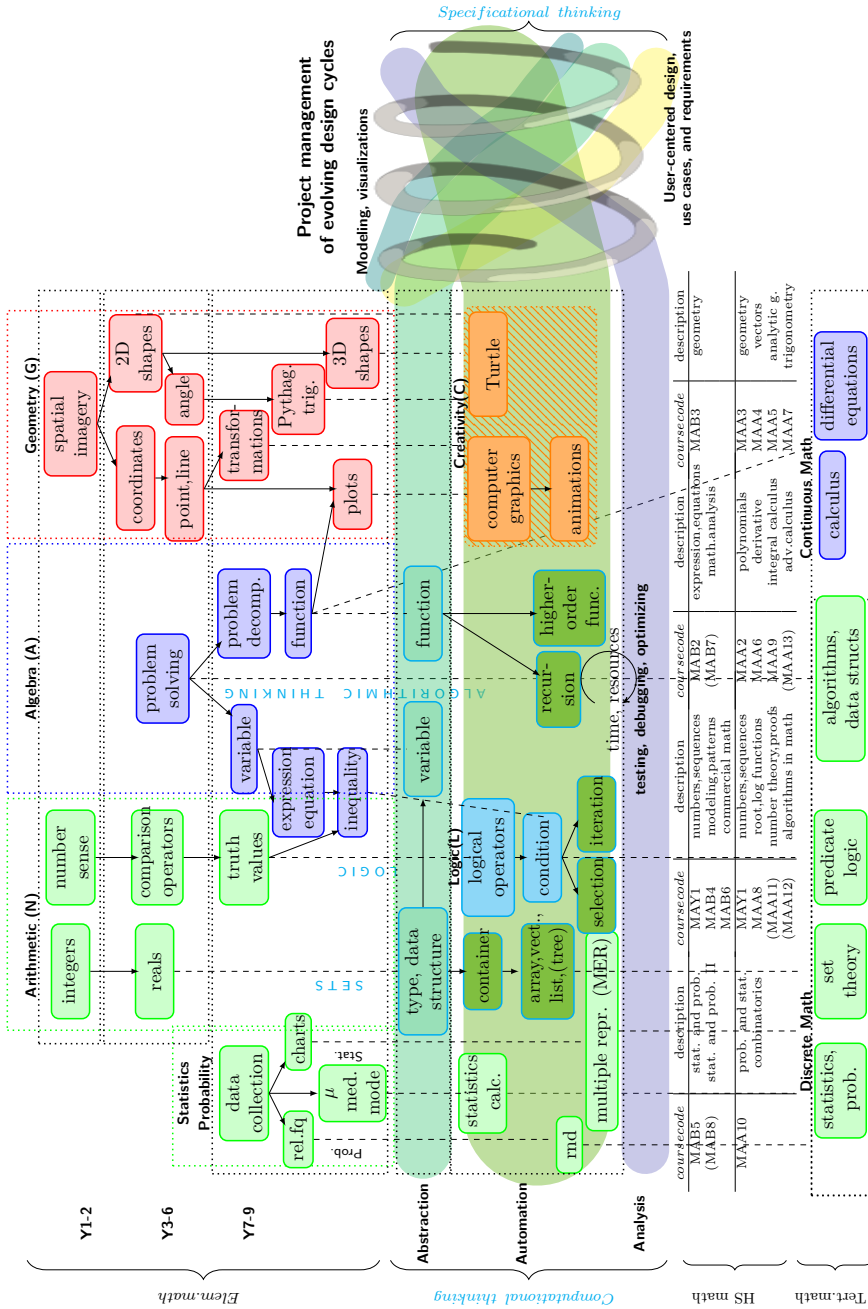
lowered mathematical self-esteem and -efficacy and regarding CS in the same way by association. Another downside of the mathematics-integration path is the rather one-sided and narrow range of topics that can be exploited compared to the total range of topics that CS embraces. If CS were to be taught as a separate subject, the syllabus would cover a much wider and more eclectic range of topics than it does when it is integrated with other subjects. The ‘separate-subject’ strategy would also require that the teachers be formally qualified as CS teachers, which would improve the general quality of the teaching. The UK has followed such a policy and their CS syllabus targets such topics as overall technological fluency. It also introduces the domains of ICT infrastructure, devices and networks, and in addition it covers socially responsible online behaviors, such as security and ethics. Besides providing opportunities for self-expression, data searching, and informal learning, computer skills should also include knowing how to use the Internet responsibly and safely. In some senses, computer skills may be seen as a part of civics: they are the basic skills needed to survive and flourish in the modern information society.

## 6.2 Conclusive CT model

Fig. 6.1 divides CT into four horizontal layers: first, the primary education comprises Years 1–2 and 3–6, and secondly, the secondary education Years 7–9, which together constitute the elementary education. The third layer is high school, Years 10–12, referred to as upper secondary education as well, and the fourth one is tertiary education; these two layers are elective. In figure, the CT layer is positioned between secondary and high school mathematics, and its content is tuned to correspond with the secondary mathematics syllabus. The vertical dashed lines represent the learning trajectories of mathematics that extend into the domain of CT.

From the mathematics syllabus, algebra is a useful scaffold for the internalization of the conceptual basis of computing. The fundamentals of function and variable run throughout algebra. A variable, in turn, leads to types. Types can be divided into primitives, such as integers and decimals, and also into more complex container-type data structures, such as arrays, lists, and vectors. In abstracting and modeling data types and their operations, set theory, or simply ‘Sets’, provide a number of highly appropriate mathematical analogies. For example, in Fig. 6.1 the learning trajectory of Sets starts from number sets in mathematics and links them with types in computing, starting from simple numeric types followed by containers. In the UKNC syllabus, Venn diagrams are used to illustrate the basic set operations of union, intersection, and cut. Correspondingly, the same operations can be covered by computing exercises.

The learning trajectories of algorithms and data structures are vital for CT. Currently, the elementary mathematics curriculum does not define specific learning targets for algorithms, merely stating that there is a need for algorithmic thinking. However, for the sake of constructive alignment and to enable a more detailed instructional design, the expected learning outcomes must be explained thoroughly (Biggs, 1996). Whatever the approach, problem-solving and decomposition are of primary importance. The simplest definition of an algorithm defines it as a sequence of commands. Decomposition in computing implies dividing the overall code into subroutines. Optimization of the number of steps, time, and resources signals the dawn of algorithmic thinking. Ultimately, the simplest algorithms, e.g. of the sort and search type, were natural learning targets.



**Figure 6.1.** Learning trajectories bridged from the FNC-2014 elementary to higher-education mathematics. High school course codes, MAA\* and MAB\*, refer to the course contents specified in 2016 Finnish National Core Curriculum for general upper secondary education (Finnish National Board of Education, 2015).

According to the feedback from the SW engineers, logic is regarded as the next most useful topic after algorithms, and thus needs to be emphasized in any CS syllabus. Algorithms and logic are clearly the two most important skills to be taught for CS, but there are several other significant topics. For example, sets and probability are also important skills. Despite of their partial support, Fig. 6.1 presents consistent trajectories for these topics as well. Although FNC-2014 specifies the goal of logical thinking, there is precious little advice for the mathematics teacher as to how this should be taught. An existing purpose-built logic subset could be copied from the UK, for example, as the UKNC introduces logic as a subset including the following topics: binary and hexadecimal notations, binary addition and shift, Boolean values and operators, and truth tables. It is not only mathematics that provides topics in which to embed aspects of logic. Studying one's native language also provides opportunities to teach logic. For instance, parsing a sentence involves considering its truth value and its ambiguities. In longer essays, a logical chain of argument should lead naturally to the conclusions that can be drawn. These kinds of exercises also promote logical thinking.

Leaving the CT layer until last, there are the two remaining elective mathematics layers, high-school and tertiary mathematics, to deal with. High-school mathematics is divided into A and B mathematics: A is mathematics as a major subject, and B is mathematics as a minor one (Finnish National Board of Education, 2015). It is perhaps regrettable, but the Finnish high-school curriculum is rigidly targeted at the matriculation exam. This exam's importance for students is immense, as it is the main selection criterion for tertiary education. If this mathematics syllabus is to be expanded with computing topics, it raises a lot of ancillary issues. Algorithms only appear in the descriptions of the elective courses of number theory and proofs (the course code: MAA11), and in mathematics (MAA12). MAA11 also introduces conjunctives and truth values, which are approaching logic. With regard to the remaining trajectories, sets are completely missing from the current FNC, both at the elementary and high school levels, but statistics and probability fare better as they are already introduced at the elementary level. Tertiary mathematics presents the required mathematics skills for modern SW engineers by presenting the two topics they regarded the most prominent, algorithms and logic. The main dilemma is finding an appropriate balance between discrete and continuous mathematics.

In the CT layer, the CS fundamentals correspond to the mathematics fundamentals at the secondary level, except for random values, selection and iteration, which are missing. This implies that the already established mathematics schedule in Years 7–9 would be an appropriate order of introduction for the corresponding CS fundamentals. The CS concepts are layered in the corresponding phases of CT, i.e., abstraction, automation, and analysis. Providing cyclic and self-reflective iterations of these phases, the CT design cycle starts to resemble an incremental and iterative SW development process that gradually refines a product and process. In order to ensure a fluent delivery of the product, project management takes care of, e.g., resourcing, scheduling, and negotiating with a client. For instance, in agile project management, the end criterion is to fulfill the 'definition of done'. In addition to the rigorous implementation, tests and the validation of desired functionality in the specification (user story) ensure the desired quality (James, 2010).

### Specification thinking

The 'specificational thinking' on the right in Fig. 6.1 represents the equivalent of computational thinking on the SW development side. Similarly, programming is the SW counterpart of computing. In addition to abstraction, automation, and analysis, the

threads of modeling and user-centric design are also in the mix. Modeling implies both data modeling and visualization in order to conceptualize a system. User-centered design means that a product should respond to the user's expectations, which begins by specifying the needs. In this process, first, the use cases and requirements are defined together with a customer. This requires more than just negotiation skills. In order to capture all the essentials in a specification, the SW engineer needs knowledge of the domain and cautious observation as primary practice in user-centered design. Translating all these influences and information into clearly worded specifications, while minimizing the chances for misunderstandings, requires precision and a mastery of the nuances of spoken language, i.e., the capacity to recognize sentences as implicit logical propositions.

For the SW engineer, the next stage in user-centered design is to read the specification, extract the most relevant concepts and define their mutual relations. Then, a system-wide architecture has to be sketched out as a UML diagram, for instance, class diagrams in object-oriented programming. Class diagrams, as a means of abstraction, can be equated to concept maps, which are already utilized in the Finnish education system in a few academic subjects, but not in mathematics. Other means of abstraction covered by the research include flow charts and Design Recipe.

Computational thinking is mainly focused on the thinking skills required in CS, whereas professional SW development demands specificational thinking skills. On a scale of 1 to 4, (1, not useful; 4, very useful) (Surakka, 2007), SW engineers ranked the skills of design (3.7) and the management of user requirements (3.6) as the most useful skills after algorithms and data structures (3.8). In another study the top ten topics include general SW architecture and design, requirements gathering, and project management (Lethbridge, 1998). Puhakka and Ala-Mutka (2009) conclude that the topics that are inadequately covered during higher education include databases, and a number of SW development skills such as planning, requirements engineering, and general topics needed in professional life, such as presentation, negotiation and writing skills, all of which fall into the domain of specificational thinking.

### 6.3 Further research

In order to close the digital skills gap, the position of CS in the curriculum should be established and strengthened. There is still some dispute about what is the best approach to introducing CS into the school curriculum. Should it be integrated into mathematics, or should it be taught as a separate subject? The latter approach would require a system for CS teacher certification, a mammoth task but one which would improve the quality of teaching. In the meantime, mathematics teachers will need a lot of support, in-service training, and good training material that clarifies the conceptual basis of CS. As an elective subject in high school, CS should be an option in the matriculation exams. These exams would be appropriate for students applying for tertiary education, in particular any studies including technological aspects. The shift in teaching mathematics away from its classical origins and towards computing and discrete mathematics must be carried out in an evidence-based manner, i.e., the learning outcomes must be carefully evaluated in co-operation with pedagogical experts. Care should be taken that any changes in the mathematics syllabus should not threaten Finland's enviable position in international comparisons and assessments of students' performance in mathematics, such as PISA.

# Bibliography

- ACM&IEEE, “Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, December 20, 2013,” Tech. Rep., 2013. [Online]. Available: <http://www.acm.org/education/CS2013-final-report.pdf>
- Anderson, J. R., *Cognitive psychology and its implications*. WH Freeman Times Books Henry Holt & Co, 1990.
- Anderson, T. and Shattuck, J., “Design-based research: A decade of progress in education research?” *Educational researcher*, vol. 41, no. 1, pp. 16–25, 2012.
- Archer, L., “Systematic Methodology for Designs,” 1965.
- Armoni, M., Meerbaum-Salant, O., and Ben-Ari, M., “From Scratch to "real programming",” *ACM Transactions on Computing Education (TOCE)*, vol. 14, no. 4, p. 25, 2015.
- Artigue, M., “Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work,” *International Journal of Computers for Mathematical Learning*, vol. 7, no. 3, pp. 245–274, 2002.
- Avalos, B., “Teacher professional development in teaching and teacher education over ten years,” *Teaching and teacher education*, vol. 27, no. 1, pp. 10–20, 2011.
- Bal, H. E. and Grune, D., *Programming language essentials*. Addison-Wesley, 1994.
- Balanskat, A. and Engelhart, K., “Computing our future: Computer programming and coding – Priorities, school curricula and initiatives across Europe,” 2014.
- Bandura, A., “Guide for constructing self-efficacy scales,” *Self-efficacy beliefs of adolescents*, vol. 5, no. 307-337, 2006.
- Barefoot, C., “Computational thinking,” Available: [http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking/\[Stand: 15.01. 2017\]](http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/computational-thinking/[Stand: 15.01. 2017]), 2014.
- Barnett, S. M. and Ceci, S. J., “When and where do we apply what we learn?: A taxonomy for far transfer.” *Psychological bulletin*, vol. 128, no. 4, p. 612, 2002.
- Baroody, A. J., “The developmental bases for early childhood number and operations standards,” *Engaging young children in mathematics: Standards for early childhood mathematics education*, pp. 173–219, 2004.

- Barr, V. and Stephenson, C., “Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community?” *ACM Inroads*, vol. 2, no. 1, pp. 48–54, 2011.
- Bayman, P. and Mayer, R. E., “A diagnosis of beginning programmers’ misconceptions of basic programming statements,” *Communications of the ACM*, vol. 26, no. 9, pp. 677–679, 1983.
- Bechmann, A. and Lomborg, S., “Mapping actor roles in social media: Different perspectives on value creation in theories of user participation,” *New media & society*, vol. 15, no. 5, pp. 765–781, 2013.
- Bereiter, C. and Scardamalia, M., “Intentional learning as a goal of instruction,” *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, pp. 361–392, 1989.
- Bernack-Schüler, C., Erens, R., Leuders, T., and Eichler, A., *Views and Beliefs in Mathematics Education: Results of the 19th MAVI Conference*, ser. Freiburger Empirische Forschung in der Mathematikdidaktik. Springer Fachmedien Wiesbaden, 2015.
- Biggs, J., “Enhancing teaching through constructive alignment,” *Higher education*, vol. 32, no. 3, pp. 347–364, 1996.
- Billett, S., “Knowing in practice: Re-conceptualising vocational expertise,” *Learning and Instruction*, vol. 11, no. 6, pp. 431–452, 2001.
- Blackwood, N., “Digital skills crisis: second report of session 2016–17,” 2016.
- Bransford, J. D., Brown, A. L., and Cocking, R. R., “How people learn,” 2000.
- Brizuela, B. M., Blanton, M., Sawrey, K., Newman-Owens, A., and Gardiner, A. M., “Children’s use of variables and variable notation to represent their algebraic ideas,” *Mathematical Thinking and Learning*, vol. 17, no. 1, pp. 34–63, 2015.
- Brownell, W. A. and Chazal, C. B., “The effects of premature drill in third-grade arithmetic,” *The Journal of Educational Research*, vol. 29, no. 1, pp. 17–28, 1935.
- Bruner, J. S., *The process of education*. Harvard University Press, 2009.
- Burke, Q., “Mind the metaphor: charting the rhetoric about introductory programming in K-12 schools,” *On the Horizon*, vol. 24, no. 3, pp. 210–220, 2016.
- Butterfield, E. C. and Nelson, G. D., “Promoting positive transfer of different types,” *Cognition and Instruction*, vol. 8, no. 1, pp. 69–102, 1991.
- Cai, J., Moyer, J. C., Wang, N., and Nie, B., “Examining students’ algebraic thinking in a curricular context: A longitudinal study,” *Early algebraization*, pp. 161–185, 2011.
- Carraher, D. W., Schliemann, A. D., and Schwartz, J., “Early algebra is not the same as algebra early,” 2008.
- Chen, J.-Q. and McCray, J., “A conceptual framework for teacher professional development: The whole teacher approach,” *NHSA dialog*, vol. 15, no. 1, pp. 8–23, 2012.
- Choi, J., An, S., and Lee, Y., “Computing education in Korea — current issues and endeavors,” *ACM Transactions on Computing Education (TOCE)*, vol. 15, no. 2, p. 8, 2015.

- Clements, D. H., “Linking research and curriculum development,” *International research in mathematics education*, p. 599, 2002.
- Clements, D. H. and Sarama, J., “Learning trajectories in mathematics education,” *Mathematical thinking and learning*, vol. 6, no. 2, pp. 81–89, 2004.
- — —, “Effects of a preschool mathematics curriculum: Summative research on the building blocks project,” *Journal for Research in Mathematics Education*, pp. 136–163, 2007.
- Collins, A., “Toward a design science of education,” in *New directions in educational technology*. Springer, 1992, pp. 15–22.
- Core Standards Organization, “Mathematics Standards | Common Core State Standards Initiative,” 2015. [Online]. Available: [http://www.corestandards.org/wp-content/uploads/Math\\_Standards1.pdf](http://www.corestandards.org/wp-content/uploads/Math_Standards1.pdf)
- — —, “High School: Modeling,” <http://www.corestandards.org/Math/Content/HSM/>, 2017.
- Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., and Woollard, J., “Computational thinking: A guide for teachers,” 2015.
- CSTA, “Computer science standards,” [https://www.csteachers.org/resource/resmgr/Documents/Standards/2016StandardsRevision/INTERIM\\_StandardsFINAL\\_07222.pdf](https://www.csteachers.org/resource/resmgr/Documents/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf), 2016.
- Davies, M., “Concept mapping, mind mapping and argument mapping: what are the differences and do they matter?” *Higher education*, vol. 62, no. 3, pp. 279–301, 2011.
- Davis, E., “The Work Ahead: Europe’s Digital Imperative,” Tech. Rep., 2017. [Online]. Available: <http://www.futureofwork.com/images/article/documents/the-work-ahead-europes-digital-imperative.pdf>
- Denning, P. J., “Is computer science science?” *Communications of the ACM*, vol. 48, no. 4, pp. 27–31, 2005.
- — —, “The profession of it beyond computational thinking,” *Communications of the ACM*, vol. 52, no. 6, pp. 28–30, 2009.
- Denscombe, M., “Communities of practice: A research paradigm for the mixed methods approach,” *Journal of mixed methods research*, vol. 2, no. 3, pp. 270–283, 2008.
- Department for Education, “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](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf), p. 6, 2015.
- Dewey, J., *The child and the curriculum*. University of Chicago Press, 1902, no. 5.
- Dijkstra, E. W., “Programming as a discipline of mathematical nature,” *The American Mathematical Monthly*, vol. 81, no. 6, pp. 608–612, 1974.
- — —, “How do we tell truths that might hurt?” in *Selected Writings on Computing: A Personal Perspective*. Springer, 1982, pp. 129–131.
- Doherty, M., *Theory of mind: How children understand others’ thoughts and feelings*. Psychology Press, 2008.



- Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., and Basnet, R. B., “Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance,” *Journal of Computers in Education*, vol. 4, no. 4, pp. 355–369, 2017.
- Donaldson, M., “Children’s minds,” 1978.
- Dreyfus, T. and Eisenberg, T., “On different facets of mathematical thinking,” *The nature of mathematical thinking*, pp. 253–284, 1996.
- Drijvers, P. and Trouche, L., “From artifacts to instruments,” *Research on technology and the teaching and learning of mathematics*, vol. 2, pp. 363–391, 2008.
- Dufva, T., “Code Literacy. Understanding the programmed world,” G2 Pro gradu, diplomityö, 2013. [Online]. Available: <http://urn.fi/URN:NBN:fi:aalto-201308107486>
- Elliott, B., Oty, K., McArthur, J., and Clark, B., “The effect of an interdisciplinary algebra/science course on students’ problem solving skills, critical thinking skills and attitudes towards mathematics,” *International Journal of Mathematical Education in Science and Technology*, vol. 32, no. 6, pp. 811–816, 2001.
- Endsley, M. R., “Toward a theory of situation awareness in dynamic systems,” *Human factors*, vol. 37, no. 1, 1995.
- English Department for Education, “National Curriculum in England: Computing programmes of study,” 2013.
- Ericson, B., Guzdial, M., Morrison, B., Parker, M., Moldavan, M., and Surasani, L., “An eBook for teachers learning CS principles,” *ACM Inroads*, vol. 6, no. 4, pp. 84–86, 2015.
- Ericson, B., Adrion, W. R., Fall, R., and Guzdial, M., “State-Based Progress Towards Computer Science for All,” *ACM Inroads*, vol. 7, no. 4, pp. 57–60, 2016.
- Ericsson, K. A. *et al.*, “The influence of experience and deliberate practice on the development of superior expert performance,” *The Cambridge handbook of expertise and expert performance*, vol. 38, pp. 685–705, 2006.
- Felleisen, M., Findler, R., Flatt, M., and Krishnamurthi, S., *How to Design Programs, Second Edition*. MIT-Press, 2014. [Online]. Available: <http://www.ccs.neu.edu/home/matthias/HtDP2e/>
- Finnish National Board of Education, “Peruskoulun opetussuunnitelman perusteet 1994,” 1994. [Online]. Available: <https://www.finna.fi/Record/jykdok.829368>
- , “Finnish National Curriculum 2004,” [http://www.oph.fi/english/curricula\\_and\\_qualifications/basic\\_education/curricula\\_2004](http://www.oph.fi/english/curricula_and_qualifications/basic_education/curricula_2004), 2004.
- , “Finnish National Curriculum 2014,” 2014. [Online]. Available: [https://www.oph.fi/download/163777\\_perusopetuksen\\_opetussuunnitelman\\_perusteet\\_2014.pdf](https://www.oph.fi/download/163777_perusopetuksen_opetussuunnitelman_perusteet_2014.pdf)
- , “National core curriculum for general upper secondary education,” [http://www.oph.fi/download/172124\\_lukion\\_opetussuunnitelman\\_perusteet\\_2015.pdf](http://www.oph.fi/download/172124_lukion_opetussuunnitelman_perusteet_2015.pdf), 2015.
- Fleury, A. E., “Programming in Java: student-constructed rules,” in *ACM SIGCSE Bulletin*, vol. 32, no. 1. ACM, 2000, pp. 197–201.

- Fong, A. B., Jaquet, K., and Finkelstein, N., “Who Repeats Algebra I, and How Does Initial Performance Relate to Improvement When the Course Is Repeated?” *Regional Educational Laboratory West*, 2014.
- Frey, C. B. and Osborne, M. A., “The future of employment: how susceptible are jobs to computerisation?” *Technological Forecasting and Social Change*, vol. 114, pp. 254–280, 2017.
- Fuller, R. B., “A comprehensive anticipatory design science,” *No More Secondhand God and Other Writings*, pp. 84–117, 1957.
- Gagné, M. and Deci, E. L., “Self-determination theory and work motivation,” *Journal of Organizational Behavior*, vol. 26, no. 4, pp. 331–362, 2005.
- Gagné, R. M., *The Conditions of Learning*. New York: Holt, Rinehart and Winston, 1965.
- García-Peñalvo, Reimann, Tuul, Rees, and Jormanainen, “An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers,” 2016.
- 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](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf), p. 6, 2015.
- Gelman, R. and Gallistel, C., “Young children’s understanding of numbers,” 1978.
- Goldman, A. I. *et al.*, “Theory of mind,” 2012.
- Gray, E. M. and Tall, D. O., “Duality, ambiguity, and flexibility: A proceptual view of simple arithmetic,” *Journal for research in Mathematics Education*, pp. 116–140, 1994.
- Grover, S. and Pea, R., “Computational Thinking in K–12 A Review of the State of the Field,” *Educational Researcher*, vol. 42, no. 1, pp. 38–43, 2013.
- Gueudet, G. and Trouche, L., “Towards new documentation systems for mathematics teachers?” *Educational Studies in Mathematics*, vol. 71, no. 3, pp. 199–218, 2009.
- Gupta, M., “Liquid workforce: The workforce of the future,” *Radical Reorganization of Existing Work Structures through Digitalization*, p. 1, 2017.
- Gülbahar, Y. and Kalelioglu, F., “The effects of teaching programming via Scratch on problem solving skills: A discussion from learners’ perspective,” *Informatics in Education-An International Journal*, vol. 13.1, no. Vol13.1, pp. 33–50, 2014.
- Halmos, P. R., “Mathematics as a creative art,” *American Scientist*, vol. 56, no. 4, pp. 375–389, 1968.
- Haskell, R. E., *Transfer of learning: Cognition and instruction*. Elsevier, 2000.
- Heintz, F. and Mannila, L., “Computational Thinking for All – An Experience Report on Scaling up Teaching Computational Thinking to All Students in a Major City in Sweden,” in *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2018.

- Heintz, F., Mannila, L., and Färnqvist, T., “A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education,” *Frontiers in Education*, vol. October, 2016.
- Hemmendinger, D., “A plea for modesty,” *ACM Inroads*, vol. 1, no. 2, pp. 4–7, 2010.
- Hibert, J. and Lefevre, P., “Conceptual and procedural knowledge in mathematics: An introductory analysis,” *Conceptual and procedural knowledge; The case of mathematics*, pp. 1–23, 1986.
- Hiebert, J., *Conceptual and Procedural Knowledge: The Case of Mathematics*. Taylor & Francis, 2013.
- Hoar, R., “Generally educated in the 21st century: The importance of computer literacy in an undergraduate curriculum,” in *Proceedings of the Western Canadian Conference on Computing Education*. ACM, 2014, p. 6.
- House of Commons, “Oral evidence: Digital skills gap,” 2016.
- Hsieh, H.-F. and Shannon, S. E., “Three approaches to qualitative content analysis,” *Qualitative health research*, vol. 15, no. 9, pp. 1277–1288, 2005.
- ISTE, “CT Leadership toolkit,” 2015. [Online]. Available: <http://www.iste.org/docs/ct-documents/ct-leadership-toolkit.pdf?sfvrsn=4>
- Izsák, A., “Representational competence and algebraic modeling,” *Early algebraization*, pp. 239–258, 2011.
- James, M., “Scrum reference card,” *CollabNet Inc*, 2010.
- Jarvis, S. and Pavlenko, A., *Crosslinguistic influence in language and cognition*. Routledge, 2008.
- Jordan, H., Botterweck, G., Noll, J., Butterfield, A., and Collier, R., “A feature model of actor, agent, functional, object, and procedural programming languages,” *Science of Computer Programming*, vol. 98, pp. 120–139, 2015.
- Joutsenlahti, J., “Kielentäminen matematiikan opiskelussa,” in *Teoksessa A. Virta & O. Marttila (toim.) Opettaja, asiantuntijuus ja yhteiskunta. Ainedidaktinen symposium*, vol. 7, 2003, pp. 188–196.
- Jurdak, M. E. and Mouhayar, R. R. E., “Trends in the development of student level of reasoning in pattern generalization tasks across grade level,” *Educational Studies in Mathematics*, vol. 85, no. 1, pp. 75–92, 2014.
- Kaczmarczyk, L. C., Petrick, E. R., East, J. P., and Herman, G. L., “Identifying student misconceptions of programming,” in *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010, pp. 107–111.
- Kaput, J. J., “What is algebra? What is algebraic reasoning,” *Algebra in the early grades*, pp. 5–17, 2008.
- Kennedy, M., “How does professional development improve teaching?” *Review of Educational Research*, 2016.

- Khoo, E., Hight, C., Torrens, R., and Cowie, B., "Software literacy: Education and beyond," in *Software Literacy*. Springer, 2017, pp. 91–99.
- Kieran, C., "Overall commentary on early algebraization: Perspectives for research and teaching," *Early algebraization*, pp. 579–593, 2011.
- — —, "Algebraic thinking in the early grades: What is it," *The Mathematics Educator*, vol. 8, no. 1, pp. 139–151, 2004.
- Kimonen, E. and Nevalainen, R., "Active learning in the process of educational change," *Teaching and Teacher Education*, vol. 21, no. 6, pp. 623–635, 2005.
- Koellner, K., Jacobs, J., Borko, H., Roberts, S., and Schneider, C., "Professional development to support students' algebraic reasoning: An example from the problem-solving cycle model," *Early Algebraization*, pp. 429–452, 2011.
- Küchemann, D., "Children's understanding of numerical variables," *Mathematics in school*, vol. 7, no. 4, pp. 23–26, 1978.
- Kurki-Suonio, K. and Kurki-Suonio, R., "Ajatuksia didaktisesta fysiikasta," *Teoksessa J. Lavonen & M. Erätuuli (toim.) Tuulta purjeisiin. Matemaattisten aineiden opetus*, pp. 62–82, 2000.
- Kurvinen, E., Lindén, R., Rajala, T., Kaila, E., Laakso, M.-J., and Salakoski, T., "Automatic assessment and immediate feedback in first grade mathematics," in *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*. ACM, 2014, pp. 15–23.
- Kuusisaari, H. *et al.*, "Kehittävä kollaboraatio: Uuden tiedon tuottaminen opettajien lähikehityksen vyöhykkeellä," 2016.
- Köhler, W., *Gestalt psychology: An introduction to new concepts in modern psychology*. WW Norton & Company, 1970.
- Lamagna, E. A., "Algorithmic thinking unplugged," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 45–52, 2015.
- Lavonen, J., "Learning and the use of ICT in science education," *Effective use of ICT in Science Education*, p. 6, 2008.
- Lavonen, J., Krzwacki, H., Koistinen, L., Welzel-Breuer, M., and Erb, R., "In-service teacher education course module design focusing on usability of ICT applications in science education," *Nordic Studies in Science Education*, vol. 8, no. 2, pp. 138–149, 2012.
- Lehtinen, E., Hakkarainen, K., and Palonen, T., "Understanding learning for the professions: How theories of learning explain coping with rapid change," in *International handbook of research in professional and practice-based learning*. Springer, 2014, pp. 199–224.
- Lent, R. W., Lopez, F. G., and Bieschke, K. J., "Mathematics self-efficacy: Sources and relation to science-based career choice." *Journal of counseling psychology*, vol. 38, no. 4, p. 424, 1991.

- Lethbridge, T. C., “The relevance of software education: A survey and some recommendations,” *Annals of Software Engineering*, vol. 6, no. 1-4, pp. 91–110, 1998.
- Levy, D., “Computer science education as part of an undergraduate program in community information systems,” in *Frontiers in Education Conference, 2013 IEEE*. IEEE, 2013, pp. 417–422.
- — —, “Racket fun-fictional programming to elementary mathematic teachers,” 2013.
- Liu, J., “DragonBox: Algebra beats Angry Birds,” *Wired*, June, 2012.
- LLC, M. (2017) kidsakoder.no. [Online]. Available: <https://kidsakoder.no/>
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E., “The Scratch programming language and environment,” *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.
- Marghitsu, D., Hur, J. W., Rawajfih, Y., Hall, J., and Stephens, C., “Promoting Computer Science among Girls: An Auburn University Pilot Program,” in *Society for Information Technology & Teacher Education International Conference*. Association for the Advancement of Computing in Education (AACE), 2014, pp. 112–119.
- Marttala, L., “Tietotekniikan valtakunnallisten oppisisältöjen toteutuminen Keski-Suomen peruskoulujen opetuskäytänteissä,” 2017.
- McGowen, M., DeMarois, P., and Tall, D., “Using the function machine as a cognitive root,” 2000.
- McGowen, M. A. and Tall, D. O., “Metaphor or Met-Before? The effects of previous experience on practice and theory of learning mathematics,” *The Journal of Mathematical Behavior*, vol. 29, no. 3, pp. 169–179, 2010.
- Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M., “Habits of programming in scratch,” in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 2011, pp. 168–172.
- Mezirow, J., “Transformative learning: Theory to practice,” *New directions for adult and continuing education*, vol. 1997, no. 74, pp. 5–12, 1997.
- Nicolaides, A. and Marsick, V. J., “Understanding adult learning in the midst of complex social “liquid modernity”,” *New Directions for Adult and Continuing Education*, vol. 2016, no. 149, pp. 9–20, 2016.
- Niemelä, P., “All rosy in Scratch lessons: no bugs but guts with visual programming,” in *Frontiers In Education Conference (FIE)*, 2017.
- Niemelä, P. and Helevirta, M., “K-12 Curriculum Research: The Chicken and the Egg of Math-aided ICT Teaching,” *International Journal of Modern Education and Computer Science*, vol. 9, no. 1, p. 1, 2017.
- Niemelä, P. and Valmari, A., “Elementary math to close the digital skills gap,” in *CSEDU 2018 Conference*, vol. 10, 2018.
- Niemelä, P., Di Flora, C., Helevirta, M., and Isomöttönen, V., “Educating future coders with a holistic ICT curriculum and new learning solutions,” 2016.

- Niemelä, P., Isomöttönen, V., and Lipponen, L., “Successful design of learning solutions being situation aware,” *Education and Information Technologies*, vol. 21, no. 1, pp. 105–122, 2016.
- Niemelä, P., Partanen, T., Harsu, M., Leppänen, L., and Ihantola, P., “Computational Thinking as an Emergent Learning Trajectory of Mathematics,” in *Koli Calling International Conference on Computing Education Research*, vol. 17, no. 1, 2017.
- Niemelä, P., Partanen, T., Mannila, L., Poranen, T., and Järvinen, H.-M., “Code abc mooc for math teachers,” in *International Conference on Computer Supported Education*. Springer, 2017, pp. 66–96.
- Norwegian Ministry of Education and Research, “Science for the Future,” Tech. Rep., 2010. [Online]. Available: [https://www.regjeringen.no/globalassets/upload/kd/vedlegg/uh/rapporter\\_og\\_planer/science\\_for\\_the\\_future.pdf](https://www.regjeringen.no/globalassets/upload/kd/vedlegg/uh/rapporter_og_planer/science_for_the_future.pdf)
- OECD, “Students, computers and learning,” 2015. [Online]. Available: <http://dx.doi.org/10.1787/9789264239555-en>
- Ørngreen, R., “Reflections on design-based research,” in *Human Work Interaction Design. Work Analysis and Interaction Design Methods for Pervasive and Smart Workplaces*. Springer, 2015, pp. 20–38.
- Papert, S., *Mindstorms: Children, computers, and powerful ideas*, 1980.
- — —, “An exploration in the space of mathematics educations,” *International Journal of Computers for Mathematical Learning*, vol. 1, no. 1, pp. 95–123, 1996.
- Pappano, L., “Learning to think like a computer,” *Education Life*, vol. April, 2017. [Online]. Available: <https://nyti.ms/2nS72IU>
- Partanen, T., Niemelä, P., Mannila, L., and Timo, P., “Educating Computer Science Educators Online: A Racket MOOC for Elementary Math Teachers of Finland,” in *CSEDU 2017 Conference*, vol. 9, 2017.
- Partovi, H., “Transforming US education with computer science,” in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. ACM, 2014, pp. 5–6.
- Pehkonen, E., “A hidden regulating factor in mathematics classrooms: mathematics-related beliefs,” *Document Resume*, p. 15, 2001.
- Perkins, D. N. and Salomon, G., “Teaching for transfer.” *Educational leadership*, vol. 46, no. 1, pp. 22–32, 1988.
- Piaget, J., “Intellectual evolution from adolescence to adulthood,” *Human development*, vol. 15, no. 1, pp. 1–12, 1972.
- Piaget, J. and Duckworth, E., “Genetic epistemology,” *American Behavioral Scientist*, vol. 13, no. 3, pp. 459–480, 1970.
- Pinar, W. F., *What is curriculum theory?* Routledge, 2012.
- Pólya, G., “How to solve it,” 1945.

- Puhakka, A. and Ala-Mutka, K., “Survey on the knowledge and education needs of Finnish software professionals,” *Tampere University of Technology, Department of Software Systems*, 2009.
- Resnick, M., “Reviving Papert’s dream,” *Educational technology*, vol. 52, no. 4, 2012.
- — —, “Fulfilling Papert’s Dream: Computational Fluency for All,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 5–5.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., and Silverman, B., “Scratch: programming for all,” *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.
- Rich, P. J., Leatham, K. R., and Wright, G. A., “Convergent cognition,” *Instructional Science*, vol. 41, no. 2, pp. 431–453, 2013.
- Rocker, I. M., “When code matters,” *Architectural Design*, vol. 76, no. 4, pp. 16–25, 2006.
- Rodriguez, B., Kennicutt, S., Rader, C., and Camp, T., “Assessing computational thinking in cs unplugged activities,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 501–506.
- Rogers, E. M., “Elements of diffusion,” *Diffusion of innovations*, vol. 5, pp. 1–38, 2003.
- Ryan, R. and Deci, E., “Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being,” *American psychologist*, vol. 55, no. 1, 2000.
- Sarama, J. and Clements, D., *Early Childhood Mathematics Education Research: Learning Trajectories for Young Children*, ser. Studies in Mathematical Thinking and Learning Series. Taylor & Francis, 2009.
- Scalise, K., “Next wave for integration of educational technology into the classroom: Collaborative technology integration planning practices,” in *Assessment and Teaching of 21st Century Skills*. Springer, 2018, pp. 239–255.
- Schanzer, E., Krishnamurthi, S., and Fisler, K., “Creativity, Customization, and Ownership: Game Design in Bootstrap: Algebra,” 2018.
- Schanzer, E. T., *Algebraic Functions, Computer Programming, and the Challenge of Transfer*, 2015.
- Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O’Grady-Cunniff, D., Owens, B. B., Stephenson, C., and Verno, A., “CSTA K–12 Computer Science Standards: Revised 2011,” 2011.
- Sinclair, B. B., Naizer, G., and Ledbetter, C., “Observed implementation of a science professional development program for K–8 classrooms,” *Journal of Science Teacher Education*, vol. 22, no. 7, pp. 579–594, 2011.
- Skolverket, “Läroplan för grundskolan, förskoleklassen och fritidshemmet 2011 (reviderad 2017),” <https://www.skolverket.se/publikationer?id=3813>, 2017.
- Skolverket, “Få syn på digitaliseringen på grundskolenivå,” <https://www.skolverket.se/publikationer?id=3783>, 2017.

- Stephan, M., *Learner-centered teaching in mathematics education*, ser. Encyclopedia of Mathematics Education. Springer, 2014, pp. 338–343.
- Suhonen, J., de Villiers, M. R., and Sutinen, E., “Fodem: a multi-threaded research and development method for educational technology,” *Educational Technology Research and Development*, vol. 60, no. 2, pp. 287–305, 2012.
- Surakka, S., “What subjects and skills are important for software developers?” *Communications of the ACM*, vol. 50, no. 1, pp. 73–78, 2007.
- Susac, A., Bubic, A., Vrbanc, A., and Planinic, M., “Development of abstract mathematical reasoning: the case of algebra,” *Frontiers in human neuroscience*, vol. 8, pp. 679–679, 2014.
- Tall, D., Gray, E., Ali, M. B., Crowley, L., DeMarois, P., McGowen, M., Pitta, D., Pinto, M., Thomas, M., and Yusof, Y., “Symbols and the bifurcation between procedural and conceptual thinking,” *Canadian Journal of Math, Science & Technology Education*, vol. 1, no. 1, pp. 81–104, 2001.
- Tedre, M., *The Science of Computing: Shaping a Discipline*. Taylor & Francis, 2014.
- Tedre, M. and Denning, P. J., “The long quest for computational thinking,” in *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 2016, pp. 120–129.
- The Design-Based Research Collective, “Design-based research: An emerging paradigm for educational inquiry,” *Educational Researcher*, pp. 5–8, 2003.
- Tikkanen, P., “"Helpompaa ja haus Kempaa kuin luulin": matematiikka suomalaisten ja unkarilaisten perusopetuksen neljäluokkalaisten kokemana,” 2008.
- Trouche, L. and Drijvers, P., “Handheld technology for mathematics education: flashback into the future,” *ZDM*, vol. 42, no. 7, pp. 667–681, 2010.
- Trust, T., Krutka, D. G., and Carpenter, J. P., “"Together we are better": Professional learning networks for teachers,” *Computers & Education*, 2016.
- Tulivuori, J., “Digi will not replace teachers (blog writing in Finnish,” 2018.
- Van Roy, P., “Programming paradigms for dummies: What every programmer should know,” *New computational paradigms for computer music*, vol. 104, 2009.
- Van-Roy, P. and Haridi, S., *Concepts, techniques, and models of computer programming*. MIT press, 2004.
- Varga, T., “Mathematics education in Hungary today,” *Educational Studies in Mathematics*, vol. 19, no. 3, pp. 291–298, 1988.
- Vee, A., “Understanding computer programming as a literacy,” *Literacy in Composition Studies*, vol. 1, no. 2, pp. 42–64, 2013.
- Vilkka, H., “Tutki ja kehittä. 1.-2. painos,” *Helsinki: Tammi*, 2005.
- Vogel, S., Santo, R., and Ching, D., “Visions of Computer Science Education: Unpacking Arguments for and Projected Impacts of CS4All Initiatives,” pp. 609–614, 2017.



- Von Neuman, J., "First Draft of a Report on the EDVAC," *Pennsylvania: University of Pennsylvania*, 1945.
- Voogt, J., Fisser, P., Roblin, N. P., Tondeur, J., and van Braak, J., "Technological pedagogical content knowledge—a review of the literature," *Journal of Computer Assisted Learning*, vol. 29, no. 2, pp. 109–121, 2013.
- Vygotsky, L. S., *Mind in society: The development of higher psychological processes*. Harvard University Press, 1980.
- Wang, F. and Hannafin, M. J., "Design-based research and technology-enhanced learning environments," *Educational Technology Research and Development*, vol. 53, no. 4, pp. 5–23, 2005.
- Watson, J., Pape, L., Murin, A., Gemin, B., and Vashaw, L., "Keeping pace with K-12 digital learning: An annual review of policy and practice," *Evergreen Education Group*, 2014.
- Wegner, P., "Guest editor's introduction to special issue of computing surveys," *ACM Comput. Surv.*, vol. 21, pp. 253–258, 1989.
- Wilkie, K., "Students' use of variables and multiple representations in generalizing functional relationships prior to secondary school," *Educational Studies in Mathematics*, pp. 1–29, 2016.
- — —, "Learning to teach upper primary school algebra: changes to teachers' mathematical knowledge for teaching functional thinking," *Mathematics Education Research Journal*, vol. 28, no. 2, pp. 245–275, 2016.
- Wilkie, K. J. and Clarke, D. M., "Developing students' functional thinking in algebra through different visualisations of a growing pattern's structure," *Mathematics Education Research Journal*, vol. 28, no. 2, pp. 223–243, 2016.
- Wilson, C., "Hour of code—a record year for computer science," *ACM Inroads*, vol. 6, no. 1, pp. 22–22, 2015.
- Wing, J. M., "Computational thinking and thinking about computing," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 366, no. 1881, pp. 3717–3725, Oct 28 2008.
- — —, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.
- — —, "Computational Thinking: What and Why? Link Magazine," 2010.
- Yackel, E. and Cobb, P., "Sociomathematical norms, argumentation, and autonomy in mathematics," *Journal for research in mathematics education*, pp. 458–477, 1996.
- Yadav, A., Hong, H., and Stephenson, C., "Computational thinking for all: pedagogical approaches to embedding 21st century problem solving in K-12 classrooms," *TechTrends*, vol. 60, no. 6, pp. 565–568, 2016.
- Zeldin, A. L. and Pajares, F., "Against the odds: Self-efficacy beliefs of women in mathematical, scientific, and technological careers," *American Educational Research Journal*, vol. 37, no. 1, pp. 215–246, 2000.

# Publications



# Publication I

Niemelä P., Isomöttönen V. and Lipponen L., “Successful design of learning solutions being situation aware”, *Education and Information Technology*, 2016

DOI: <https://doi.org/10.1007/s10639-014-9311-2>

Niemelä et al. (2016)

# Successful design of learning solutions being situation aware

Pia Niemelä · Ville Isomöttönen · Lasse Lipponen

© Springer Science+Business Media New York 2014

**Abstract** Education is increasingly enhanced by technology, and at the same time, the rapid pace of technology innovation and growing demand of consumers introduces challenges for providers of technological learning solutions. This paper investigates Finnish small and medium size companies who either develop or deliver technological solutions for education. Twelve companies were interviewed in order to capture the entrepreneurial narratives of successful design of learning solutions. Data was analyzed based on a conceptual framework. The framework draws on the situation awareness concept, meaning that we seek to answer the question how the participant enterprises examine relevant elements in their environment with regard to their development process. The results show that all the mature companies included in the study have well balanced situation awareness, but amongst the incubating and accelerating enterprises, balanced profiles are rare.

**Keywords** Situation awareness · Entrepreneurial narrative · Learning solutions

## 1 Introduction

Finland has been a top-ranked country regarding education, ICT readiness, and innovation (McKinsey 2009; OECD 2011; WEF 2010). The Fund for Peace rated Finland in its 2012 report as the least failed state and the structure of the society to be very sustainable. Sustainability is supported by a layer of public services that, for example, in regard to education take care of the quality of teachers' pedagogical training and

---

P. Niemelä (✉)

CICERO Learning, University of Helsinki, Siltavuorenpenger 5 A, 00170 Helsinki, Finland  
e-mail: pia.s.niemela@gmail.com

V. Isomöttönen

Department of Mathematical Information Technology, University of Jyväskylä, Agora Center,  
Mattiilanniemi 2, 40100 Jyväskylä, Finland

L. Lipponen

Department of Teacher Education, University of Helsinki, Siltavuorenpenger 5 A, 00170 Helsinki,  
Finland

ensure equal and free education for every citizen. Education is seen as a key factor for future success and one of the focus areas that is invested in to boost innovation as well as good learning outcomes.

For the sustainability of education, it is necessary to study what constitute successful aspects of education that is now increasingly based on technology. Failures of technology-based education have been witnessed in projects that have emphasized technology at the cost of pedagogical and contextual considerations. Furthermore, projects facing problems are terminated, rather than being used as opportunities for improvement and revitalizing efforts (Romiszowski 2004). In this light, it is the operations of learning solutions providers that must be put under empirical scrutiny to improve the provision and sustainability of technological innovation in education.

This paper investigates Finnish small and medium size companies who either develop or deliver technological solutions for education. Twelve companies were interviewed in order to capture the entrepreneurial narratives of the successful design of learning solutions. In what follows, we first review the related work and the conceptual framework of our study. We then present the design of the study, the analysis, and results. The paper ends with a discussion of the results.

## 2 Related work

### 2.1 On technology integration

Many authors hold a view that the learning gain due to technology integration relates to such elements as increased participation, action and feedback, and access to meaningful activities. Hence ICT contributes indirectly through pedagogical means and the exploitation of new resources and modes of learning (Dalgarno and Lee 2010; Mikropoulos and Natsis 2011). It is also assumed that several psychological aspects influence the learning experience (Lee et al. 2010). For instance, the study by Krentler and Willis-Flurry (2005) indicates that the use of technology increases students' interest and engagement by having an equalizing effect among the students, which finally leads to improved performance. The study by Linder et al. (2006) provides evidence how computers can contribute to teachers' metacognition about teaching and thereby improve their teaching.

While plenty of evidence on at least indirect learning gains exists, cautions against an uncritical stance towards technology integration have been noted. Bennett et al. (2008), for instance, suggest that a discourse centered at the notion of digital natives, whose learning is claimed to be dramatically different from preceding generations, is being negligently constructed in the research literature. In addition, there are groups for whom extensive use of ICT has been reported to hinder, rather than enhance, learning. One important aspect is the user's attitude towards technology: technophobes are not achieving set learning goals as easily as their more confident counterparts (Stutsman 2013; Watson 2001). Another issue with many learning technologies is poor self-regulation skills; multi-tasking prevents students from focusing on the subject properly (Daniel and Woody 2013). Gender-sensitive issues may influence the attitude as well. The masculine stereotype of gaming culture, for example, can have an alienating effect on girls that is reflected in ICT use in general (Carr 2007).

Regarding the present article, which focuses on the operations of learning solution providers, of principal interest are the various aspects associated with the success of learning solutions. An issue constantly raised is technology integration taking place at the cost of pedagogical and contextual considerations (Lee et al. 2013; Romiszowski 2004; Watson 2001). For example, in his article Romiszowski (2004) illustrates the pitfalls that occur when technology starts to dominate at expense of pedagogy. The failure of many earlier technology-based educational innovations has been that the technology is selected first and then the content; making it functional in the educational context is considered only after these phases. Failure was defined as student dissatisfaction, high dropout rates, and puerile and irrelevant learning material, problems associated with remote, computer-based courses. Earle (2002) provides a conceptualization that aptly summarizes the challenges of technology integration into a holistic guideline: establish appropriate conditions by converting restraining forces to support elements of the design process. For us, this implicates the idea whether the solution providers under study here identify crucial elements in their environment and have a strategy for transforming each of the elements from being a constraint into being a possibility.

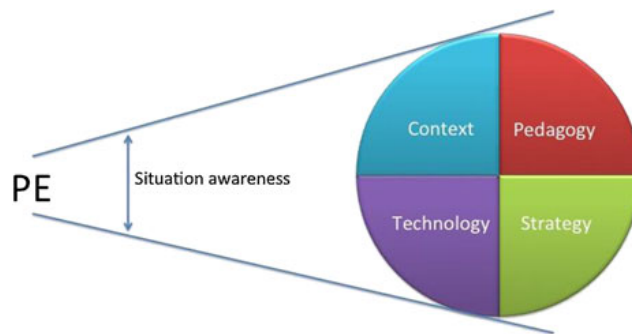
As a technological prerequisite it is assumed that companies are skilled enough to implement the product. However, limitations set by school environment should be highlighted including such elements as a restricted equipment base, dated versions of browsers and other applications, lack of infrastructure, such as the missing network or WIFI connections and the defects of ICT support and maintenance. Knowing technological constraints is essential in order to deliver adaptable solutions. Physical restrictions are one side of development, the other side is the methodological approach, whether plan-driven or an agile approach that seeks a more organic and user friendly and flexible design. Also cooperating with other companies is encouraged to strengthen technological and strategic competence and benefit the product development.

## 2.2 The conceptual framework

The conceptual framework of this study, based on the literature discussed in this section, is depicted in Fig. 1. The concept of awareness is a particular focus. Specifically, we refer to ‘situation awareness’, which originates from aviation psychology and has been extended to other complex systems (Durso et al. 1999). Endsley (1995, p. 97) paraphrases the idea as “knowing what is going on” and defines it as: “The perception of elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future.”

While classical human factors research has largely focused on human information processing, situation awareness is a more holistic concept drawing attention to meaning (Endsley 1995; Flac 1995). The meaning refers to both the interpretation of a message and the actual significance of a message. There are thus both a cognitive agent and an objective reality (real situations) involved and skillful performance in complex systems depends on the correspondence between the two. In sum, it is insufficient that elements in the environment are perceived, but an understanding about them must be developed to take appropriate action.

Situation awareness is affected by prior experience and cognitive abilities and can be trained, and thus is linked to the development of expertise (Durso et al. 1999; Endsley 1995). Experts are able to rapidly develop an integrated understanding about a



**Fig. 1** The conceptual framework for data analysis. We study how the participant enterprises (PEs) relate to their environment in terms of awareness, whether they recognize and develop understanding about the relevant elements in their environment that can have effect on product development. We have predefined a set of aspects that were retrieved from literature and confirmed the set by the pre-analysis of interviews

situation, which is assumed to occur by matching a novel situation to models and experiences that have been established in long-term memory. Novices, on the other hand, may need a great amount of effort to focus and use meta-cognitive strategies to interpret the situation. Endsley (1995) also proposes intervening factors such as a stress and a workload affecting situation awareness.

The concept of situation awareness gives us a general lens for the data analysis implying questions such as what elements participant enterprises (PEs) identify in their environment and whether they develop understanding about these elements for the benefit of their operations, and for designing learning solutions. The linkage between situation awareness and expertise suggests that it is reasonable to consider the questions of this kind in relation to the phase of PEs' businesses. The potential intervening elements in situation awareness in turn prompt the question whether these elements distract the PEs' examination of the environment.

As depicted in Fig. 1, we observe certain aspects through the concept of situation awareness: strategy, technology, pedagogy, and context. These dimensions emerge from the literature and are discussed in the following sections.

### 2.2.1 Strategic awareness

Hannon and Atherton (1998) provide a more domain-specific treatise of the situation awareness. The authors concentrate on the value of strategic planning processes in small firms and associate effective planning with the strategic awareness capability. Instead of cutting the concept into separate subcomponents, they figure out means to enhance strategic thinking by planning activities, whether formal or informal. The writers acknowledge the lack of resources as one obvious reason to drift in the direction of more informal and light planning, or to engage in no planning at all. However, they emphasize that a diligently done business plan as an essential tool of critical, objective, and thorough review. Thus the business plan is the means to enhance strategic awareness, whereas, an entrepreneurial narrative, for example, enhances the organizational coherence.

The general theme of Hannon and Atherton's analysis is 'openness'. On one hand, the effectiveness of planning is linked to the cognitive processes of those involved. This implies the need for caution against a defensive mindset that would inhibit operators



from exposing their ideas to the critique within the network where they function. In general, Hannon and Atherton refer to attitudes towards environment, the capability of being flexible and implementing the business ideas. This is called strategic awareness capability and attached to the development of expertise.

The above implies us that we should monitor whether PEs are engaged in critical thinking about their product development although they would not implement this through the production of formal plans. In describing the motives, they might use informal language instead of an academic one. With regard to ‘openness’, we could think of PEs’ reliance on subjective common-sense pedagogy, which ignores the socio-cognitive perspective and is not informed by learning theory.

### *2.2.2 Technological awareness*

Technology is evolving in a rapid space and its trajectory is influenced by various paradigm shifts. Shifts represent disruptions that offer business opportunities. In our data, the emergence of tablet devices and to a lesser extent switching from Flash to HTML5, exemplify such disruptions. When new technologies are introduced, innovators and early adopters will benefit the most providing that the trend will be favorable later on. Dutta and Crossan (2005) note that “the window of opportunity” is open only for a specific period, after which the opportunities start to fade away. Regarding the hype cycle of various phenomena, the most fruitful period is the beginning of the hype curve.

Sainio and Puumalainen (2007) and Sainio et al. (2012) divide the orientations of successful companies into two main categories: technological and customer-relationship oriented companies. That the technologically oriented company makes more technically radical innovations is an obvious result. Radicalness of an innovation means that the solution differs significantly from rivals’ solutions and is based on superior technological knowledge. Innovativeness may be partly explained by awareness of scientific breakthroughs and technological development.

However, Sainio remarks that the radical technological innovations are rare in the top-sale category and largely risks are bigger. Uncertainty of the market moderates too radical solutions. Brush (2008) highlights that only the technical orientation and awareness do not suffice in making good use of opportunities, but the company must have a clear vision, money to pursue, and social skills to persuade. At education, the usability plays a more important role than technical radicalness. Sainio et al. (2012) state that customer-relationship orientation results in better business models, however, having enough technical skills and resources is still critical for successful design of learning solutions.

### *2.2.3 Pedagogical awareness*

Too dominant technological orientation seems to threaten the pedagogical quality. The underlying pedagogical model is left unparsed or pedagogical aspects are not considered at all. According to Romiszowski (2004) developers do not seem to be aware of how people learn and use flawed instruction models.

In recent years, technology-enhanced-learning (TEL) and game and simulation based learning especially have been examined extensively. Being aware of the main research findings enables considering the most obvious pitfalls and implementing such pedagogical models that are efficient. In studies, constructivist pedagogy is embraced

and traditional, instructional pedagogical style is considered inefficient (Lee et al. 2013; Romiszowski 2004; Salden et al. 2010). Cognitive and affective learning outcomes have been reported, to a lesser extent also social skills (Connolly et al. 2012). Pedagogical awareness may be fostered also by developing the product with a focus group that includes pedagogical experts. Later, interviewing teachers that have used the product enhances awareness of its pedagogical usability.

The learning targets that are set shape the pedagogy, too. In order to deepen the learning and providing students with the skills they will need in the future we should look forward. For example, the Assessment and Teaching of Twenty-First Century Skills (2012) project has envisioned the path ahead and depicted the KSAVE model that consists of various future skills such as creative and innovative thinking, communication and collaboration skills, the literacy of information and ICT and global citizenship. Socio-constructivism as a learning view backs this skill set. Appropriately, also social media usage has features that foster the development of the 21st century skills. According to Biggs' principle of constructive alignment (2003) setting clear targets enhances reaching them.

#### 2.2.4 Contextual awareness

By context, we mean the situation, in which the learning solution is used, most commonly in a classroom during lessons. According to Wenger (2000), social learning consists of three modes of belonging: an engagement, imagination, and alignment. He argues that the companies that dig deep to the context are capable of designing better products for the industry and will succeed better: engagement fosters imagination and aligning with the context and user needs improves usability. To be successful in designing learning solutions, knowing technology is not enough but knowing the context is essential.

We participated in a project targeting to improve the quality of software used at schools. By organizing pilots, interviewing students, teachers, and companies our research team worked as a link between these groups. Having company interviews as the main source of data we started to analyze the material trying to find features that align with the success. We anticipated that companies should be aware of several domain and technology related dimensions.

In this study, we aimed to find out:

1. Which kind of conceptualizations describe the degree of the situation awareness of the company?
2. Which dimensions of awareness do the entrepreneurs emphasize most when talking freely about company's business?
3. Can we point out recommendations by contrasting the awareness profiles and narratives of examined companies?

We addressed our research goal by means of a qualitative study and more specifically a narrative analysis. Twelve companies, who either develop or deliver learning solutions, were interviewed and the resultant data was analyzed based on a conceptual framework defined based on related work and discussions among project researchers. The framework draws on the situation awareness (SA) concept, which describes a

phenomenon of an operator examining its complex environment for being able to make an appropriate decision (Endsley 1995). The situation awareness concept aptly underpins our research interests in studying how our participant enterprises navigate in the business environment of education.

### 3 Method

The participant enterprises ( $N=12$ ) with anonymized names are listed in Table 1, categorized under four product families. Each company is presented with information on the mission, the approximated business phase, size, and technologies in use. Nine of the companies design and develop their own products (LG-1, LG-2, LG-3, MS-1, MS-3, MS-4, AP-2, AP-3, IS-1), while the remaining three deliver solutions by selling hardware and platforms and training in their educational use (MS-2, AP-1, IS-2). The phase of the business is approximated with terms ‘Incubating’, ‘Accelerating’, ‘Mature’, and ‘Global’. ‘Incubating’ indicates that the company is implementing its business idea, on its way toward having a clientele, and may still depend on external funding. ‘Mature’ indicates that the company has found a place in the market or that its operations include externally funded educational activities that are not critical to its business, giving it a rather stable position. ‘Accelerating’ in turn indicates growth while the ‘Global’ adds that the company has entered the international market. These approximations were made only for research purposes, to characterize ‘advancing years’ of the companies, implying that they do not estimate business capability. In characterizing the size of the companies, we used scales 1–10, 10–20, and 20+ for the approximation of human resources and scales 0–500 k€, 500 k€–1 M€, and 1+ M€ for the approximation of turnover.

#### 3.1 Participant enterprises

#### 3.2 Interviews

The interviews lasting 1–2.5 h each were recorded and transcribed. During the interviews, some prompting was done such as, how the company was started and with which mission, and in reference to the conceptual framework given in Fig. 1 different dimensions of it were covered, while the principal method was to give interviewees as much time as they were willing to take for voicing their entrepreneurial narrative.

#### 3.3 Narrative approach

Riessman (1993) describes the narrative as the refraction of the past, rather than a camera-copy of the reality. She points out two important features, imagination and strategic interests, which also shape the narratives told. The narrative provides for a teller a way to re-imagine not only the past, but also future steps. Gartner (2007) highlights entrepreneurial narratives as “the science of imagination”, where the future of business area is echo-sounded with various “what if?” hypotheses, and thus may be

**Table 1** Participant enterprises

	PE	Mission	Phase	Personnel, volume	Technology
Learning games	LG-1	Math game for 6–10 years	Incub.	1–10, 0–500 k	PHP, JavaScript
	LG-2	Business games for graduate students	Global	10–20, 1+ M	Java
	LG-3	Collaboration skills games	Mature	20+, 1+ M	-
Mobile solutions	MS-1	GPS-based POI orienteering for learning	Incub.	1–10, 0–500 k	C++/Java
	MS-2	Tablet delivery and support	Mature	10–20, 1+ M	-
	MS-3	Activation engine for drama pedagogy	Accel.	1–10, 500 k–1 M	LAMP, DHMTL, PHP, Facebook, mobisites
Assessing, portfolios	MS-4	Scalable widgets for learning	Incub.	1–10, 0–500 k	HTML5, CSS3, node.js
	AP-1	Integrated e-learning platform	Accel.	10–20, 500 k	C#, .NET, other MS tools
	AP-2	Portfolio for nursery and primary school children	Accel.	1–10, 0–500 k	PHP, JavaScript
Infrastructure	AP-3	Assessment of ICT skills	Mature	10–20, 1+ M	subcontracted
	IS-1	Web analytics for education	Incub.	1–10, 0–500 k	browser plugins, analytics
	IS-2	Distant learning solutions	Mature	20+, 1+ M	Adobe Connect

assigned as “the language of opportunities” (Gartner 1993; Hjorth and Steyaert 2004). For example, evaluating technological disruptions with other stakeholders having similar interest clarifies and crystallizes the road ahead.

Gartner raises reflectivity as the lowest common denominator of narrative research approaches in various scholarships. The narratives should be told back by listeners with their views to add perspectives and reflect the story at the teller. By bridging the past, the present and future narratives help in creating the continuity and the permanence as a synthesis of single episodes into unbroken intrigue. In organizations, telling stories helps individuals to build collective with common values, beliefs, and norms. Extendably, this leads to the construction of cultural models. Boje complements the rationale for entrepreneurial narratives with the following: encouraging enchantment (2011), flowing, and networking (2001). Riessman also emphasizes the organizing nature of narratives and in accord Kearney (2002) states that the narratives have sovereign significance in explaining something that is otherwise inexplicable, for example, the values of the company.

In this study, the narratives are not claimed to capture the truth, but to convey faithfully enough the aspirations and vicissitudes of the SysTech member companies, so that profiling of the companies to a certain extent is possible. In profiling, we are using the conceptual framework and defining which of the dimensions (strategy, technology, pedagogy, and context) is the most dominant. The narratives are also tied to the general developmental courses of technology and pedagogy. Such contextualizing is one means to enable generalizability, as the individual narratives are positioned in the larger landscape of technological and pedagogical trends (Starr 2010).

The use of the narrative approach in the present study is actually twofold. Firstly, the data is both collected and reported in the form of entrepreneurial narratives; secondly, the methodical choice of the study is the narrative research, which means that the narrative is not only reporting means but used to teach a lesson. An attempt to depict a generic metanarrative of a strategically aware and successful company characterizes the analysis phase. In narratives, we are especially interested in noticing aspects that led to success.

A pattern coding process where themes of the qualitative data are identified based on regularities (Miles and Huberman 1984) was conducted on the data extracts that related to the situation awareness concept or its more specific dimensions. As a tool of the narrative analysis, ATLAS.ti software was used for actual coding.

## 4 Results

### 4.1 Strategy & technology dominate the design of learning solutions

Strategic reflections take place, especially in cases where the company is still incubating or unsettling. In the interviews, entrepreneurs feature the variety of alternative routes for the future in order to pull through the financial valley of death (Osawa and Miyazaki 2006). The reflection focuses particularly on funding and customer acquisition. Sometimes strategic plans will receive up to opportunistic features: the company has technological expertise that is transferred in developing learning solutions without necessarily any special attention to the application area and on occasion even reluctant

to invest in pedagogical expertise. The big public investments and a possibility to go global utilizing Finland's good PISA (OECD's Programme for International Student Assessment) reputation have tempted companies to transfer to the area of education, as the interviewee within one of the SysTech member companies (IS-1) described:

“I strongly believe that technology-enhanced learning will provide excellent business opportunities globally and we might try out to find fair deals and get some good feedback. Success in PISA tests combined with excellent teacher training will open a lot of doors to further opportunities.”

Most of the SysTech member companies exert from a technological basis (10/12). Previously, for example, they have developed various marketing, analytics, and mobile entertainment applications and the knowledge and software components have then been reused and composed as new learning tools. Only a few applications have emerged through a user-centered design and been developed from scratch through user need analysis. In implementing, the developers' own visions and experience have served as starting point for the development (IS-1):

“We should have collected more feedback from the users, but we didn't have this kind of focus group service (such in SysTech) readily at hand.”

Technological activities of member companies are contextualized to the bigger picture of technological development, where the Internet plays the most centric role. The pioneer of the internet, Tim Berners-Lee (2010), continues promoting internet best practices such as open standards and egalitarian principles and in his opinion net neutrality should apply not only desktops but to mobile connections as well. The emerging World Wide Web Consortium standards, such as HTML5, and rocketing browser support, the ease of development and deployment attract developers to web applications. The trend is exemplified by SysTech member companies the majority of which (10/12) develop web applications, however not purely with standard tools, but also using asynchronous JavaScript and XML (AJAX) and rich Internet applications (RIA).

The support for web interaction was added to many languages, for example, parsing the web pages for easy handling as separate elements were added. PHP, a server-side scripting language, went furthest; it allowed the free and flexible injections of PHP inside HTML code, resulting in an unorthodox hybrid that got native programmers and promoters of clear design patterns on their toes. The interviewees took a stand on the credibility of the programming language used, for example:

“The first technology choice was quite poor. I chose the PHP language, which was considered technologically extremely bad and I could barely implement a software with it in the early 2000s. Since then it has developed considerably and nowadays it has become a significant programming language. Even today, it is quite common that professionals underestimate the capabilities of PHP.”

Nevertheless things may change rapidly, when the positive strategic examples clear the path for others:

“While pondering a few years ago whether PHP is credible enough as a platform, Facebook came and changed the whole situation. Such a big system as Facebook has been made by using PHP.. and by a company, which is listed in stock exchange.”

However, PHP was not the only language suffering from a bad reputation:

“The second matter was the use of JavaScript. I believed that it will be an important tool at some point but at the end of the 90s, using JavaScript was considered as a joke. 3–4 years ago I decided to switch the whole system to Java, because I saw the benefit of having it as a server mainframe. It was a safe solution. Later on, the focus has shifted from languages and frameworks to scalable infrastructures and services such as Amazon Cloud, where the intermediate language is not so important.”

In addition to languages, new device categories, such as tablets, have been introduced in accelerating pace and proportions of old devices have changed rapidly:

“The outlook for Symbian was very promising at that time, new technologies were emerging.. It was clearly our main focus, Nokia had just released Xpress Music and the product portfolio in general was strong .. Then there were the first signs of iPhone...”

When implementing mobile solutions for all the noteworthy platforms, it is imperative to follow technological development to detect whether certain thresholds and tipping points have been reached, for example, following reflections on decision is made:

“We cannot decide to support e.g. Windows Phone just like that, but it has to grow.. to 10 %.”

Not only languages and devices but also digital distribution has over gone radical revolution. In the beginning, downloading an application was more cumbersome and done through different channels such as marketplaces provided by operators taking a remarkable share of the profit. Easy access to applications is crucial:

“The problem was that you had to know quite a lot to be able to even search applications. Nokia did not have any particular site for the applications that people would have known. Along with iPhone the situation changed suddenly when Apple started to announce that “this and this many applications in App Store”.”

However, as appealing the web application development might be, it contains its own challenges, such as composing relevant style sheets and media queries to enable scaling to different display sizes, taking different nonstandard features of special browsers into account, especially earlier versions of Internet Explorer (<8.0) seem to cause problems.



## 4.2 Pedagogy and context, new opportunities not capitalized

Desktop applications have shifted extensively from standalone to web applications and learning solutions naturally share the trend, thus the evolution of web is central also for learning solutions. The certain resemblance of development of web and pedagogy may be perceived, let us parallel evolution steps as a metaphor of different phases of pedagogy to illustrate the progression:

<b>Web evolution step</b>	<b>Corresponding learning perception</b>
Web 1.0	Behaviorism, cognitivism
Web 2.0	Socio-constructivism
Web 3.0	Learning analytics, user-tailored learning

During Web 1.0, the content was fixed and a user was not able to influence it. In behaviorism, learning is teacher-centered, based on stimulus–response pairs and iteration; solutions with drilling activities may be categorized into this genre (e.g. LG-1, AP-2). In static HTML pages, the biggest wow was a hyperlink that linked data forming tree-like structures. Similarly in cognitivism, concepts form schemas and the thicker the schema, the deeper the knowledge. Yet recognized effective, this type of linking, e.g. concept maps and alike do not appear among SysTech solutions. Dynamicity of the web has increased even further and gradually users have taken a role as content providers, new Web 2.0 tools, such as blogs and wikis, accelerated the progress. Web 2.0 facilitates socio-constructivist learning, the examples of which exist also among SysTech learning solutions, for example, making decisions as teams (LG-3) or a short message guided drama (MS-3). Along with a more user-centric web also evolves smarter by gathering increasingly information about users. Learning analytics (IS-1) and user-tailored user experience demonstrate these new Web 3.0 features.

Yet these developmental trends are evident both to ‘techies’ and ‘teachies’ apart, rare are the successful combinations of state of the art. The majority of SysTech companies belong to the technology-oriented group; some of the companies have reflected their own functions and observed the phenomenon of ‘hype-from-hype’ ramping, as LG-3, one of the learning game developers states:

“The development is driven by technology, not by the content or user needs and for knowing the user needs examining the context is crucial. New waves of technology follow each other quickly leaving no time to stabilize or use all the possibilities of the new technologies. The development goes from hype to hype at the expense of pedagogical aspects.”

Lacking pedagogical knowledge shows in inadequate curriculum considerations and a moderate engagement of students, in case of too straightforward a functionality of behaviorist stimulus–response pairs founded on iterations. Lessons learnt from social media are mainly lost; interaction and common content creation are not fully exploited. In addition to students, solutions must be agreeable to teachers and approved by the education authorities that invest in learning technologies. Aligning the solution with the



curriculum and proving its efficiency are the keys to convince. A viable option of the entrance into education would be co-operation with schoolbook publishers.

The curriculum is becoming more mobile (Mylläri 2012), which means that mobile devices are used more during lessons and informal learning is raised next to formal learning practices. Mobility detaches a student from a predefined learning time and place. The student has an access to information constantly and it is possible to form teams and work remotely, everybody does not need to be online simultaneously, messaging may also be asynchronous. Mobility is otherwise closely connected with the idea of informal learning, where the learning is not always via formal set-up in the school context, but can happen in a free time, while playing or surfing in the net. Tablets have made mobility as convenient for the user as possible; there are also signs that teachers experience the increased sense of competence, which is illustrated by MS-2 with the following excerpt:

“Here we have, for example, Erkki, 59, who is retiring soon. With iPad, Erkki there is confident that he still can learn new and decides to come to a tutorial course, the threshold of which could otherwise prove to be too high.”

By and large the competence aspect is important for educators. One interviewee (IS-1) compared the lesson context to warlike conditions. An immediate response is essential, hence easy to use, easy to access solutions function the best. Piloting early and often enough to sanity-check the product status and functionality in real context enables to build the product from the ground up. Paving the way for the pilots by helping in setup and providing training is a good practice to make a smooth start. In the real context also constraints such as old and various hardware, old browser versions, and problems with the network infrastructure may cause surprises.

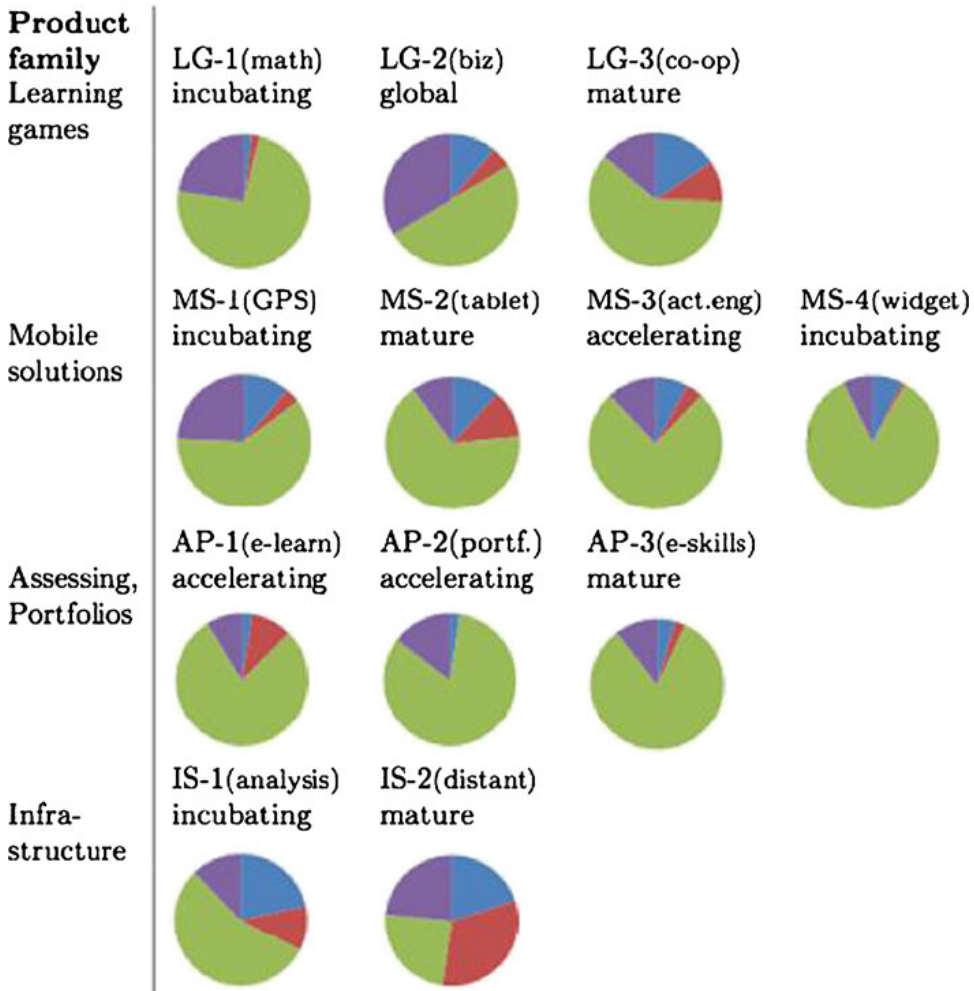
#### 4.3 Summary

The pie diagrams in Table 2 illustrate distributions of different dimensions of awareness in interviews. Before the ATLAS.ti coding, each dimension was defined more in detail with sub-codes, which were complemented during the coding phase if needed.

Based on the diagrams it can be seen that the more mature the company, the more balanced is the awareness distribution (e.g. LG-2, LG-3, MS-2, IS-2. AP-3 is an exception), whereas incubating and accelerating enterprises are very strategy-centric. Two thirds of the time these enterprises are handling such themes as funding, gaining customers, and internationalization. Whether a balanced profile is a cause or consequence to a more mature situation, it is indistinct based on our data. If it were the cause, two more balanced incubating/accelerating companies, MS-1 and IS-1 should survive and succeed better in continuation. However, to be able to prove this hypothesis we should have a longer observation span and a bigger sample.

### 5 Discussion

In this paper, we have studied 12 companies by exploring their entrepreneurial narratives in order to illustrate the influence of being situation aware on successful learning solution design. Transcribed interviews were interpreted using the conceptual

**Table 2** Situation awareness distribution

Strategy = green, technology = dark blue, pedagogy = red, context = light blue

framework containing four aspects: strategy, technology, pedagogy, and context. The former two represent the general entrepreneurial competencies of an ICT company, whereas the latter relate to the pedagogy as the domain, to which the learning solutions are targeted. The codes were derived from the conceptual framework and the actual coding was done by using ATLAS.ti qualitative data analysis software.

The conceptual framework was used as the tool to define the degree and dimensions of situation awareness of SysTech companies. The results show that business and strategy are emphasized in expense of pedagogy and context. The phase of the company is nominal, whether incubating, accelerating, or mature: the more established companies are less keen on talking about funding, marketing, and gaining customers, which are counted as strategy related things, whereas these are in the center of attention of beginning companies: “the mouth speaks what the heart is full of”. While incubating, the entrepreneurial narrative is also more blurred and inconsistent as there are many alternative paths to the future (e.g. MS-1, MS-4, and AP-2).

Narratives told by the entrepreneurs themselves are more vivid due to first-hand experience and often the narratives also steep in the features of tragedy, especially periods of survival struggle through economic downturns produce the scenes of suffering. Employees identify themselves more as performers than executives and R&D issues are in front in place of strategic considerations. In some cases commitment to the product schedule is so extensive that it creates a myopic attitude towards feedback and development initiatives (AP-3) so that keeping milestones comes before being sensitive to client needs.

The orientation (education and contacts) defines the technological interest. When interviewed, people who are software developers and coders themselves (i.e., LG-1, LG-2, MS-1, MS-3, AP-2) stress both strategy and technological viewpoints; also entrepreneurs with a technical background made perspicacious observations. Technology-oriented interviewees link their descriptions to examples of technological companies such as the Apple, Nokia, and Facebook, whereas the focus of business-oriented entrepreneurs is in success and failure. In general, companies identify the need for situation awareness regarding especially technology and follow its development e.g. from the news and by discussing to other stakeholders.

Previous experience as an entrepreneur (LG-1, MS-2, MS-3) gave perspective also for future thinking and scenario creation (Dutta and Crossan 2005; Politis 2009). The firmest future views were set out by the people with the technological expertise combined with the entrepreneurial experience. A profound enough grip on technological understanding and previous background helped in evaluating disruptions and opportunities provided. The advent of tablet computers and Apple's ascendant market dominance were the disruptions pondered in most interviews because of their direct influence on the device distribution and development. MS-2 said it was clear from the beginning that the iPad has potential in pedagogical context. Similarly, LG-1 stated that disruptive nature of iPad was obvious to them right from the start. MS-2 has profited from the tablet penetration, resulting in expansive growth of the company. However, many successful SysTech companies such as MS-2 diminish the share of far-reaching strategic thinking and highlight the importance of serendipity instead. However, fortune favors the fit, albeit only with a bit of strategic wit. Exploiting the opportunity, MS-2 managed to occupy the virgin territory of school environment lacking easy-to-use, up for it devices. In the case of LG-2, sticking at web application in 1995 was a critical decision that has later on proven to be a fortuitous choice.

In the interviews, pedagogy and context are mentioned surprisingly infrequently. Few of 12 enterprises (IS-2, MS-2, LG-3) have clearly considered pedagogy, the rest rely more or less on their own understanding and experiences gained, for example, with entertainment games and web applications (LG-1, LG-3). In some cases, the impression is that school context has been selected almost by accident or business-wise, after deducing where best to focus investments (IS-1, MS-1, MS-3, and MS-4). User-centered design practices encompass having a teacher as a pedagogical expert in the group, evaluating solutions with the focus group, and observing the praxis in real context. Some companies (IS-2, LG-3) did argue for putting content before technology and criticize overhyped waves of technology innovations that repeat iteratively and prevent from deepening the pedagogical know-how and concentrating on essentials. The essential is manifested in the curriculum; additional demands may be gathered from teachers.

## 6 Conclusions

Being aware of the situation and context in its entirety is essential for success in all phases, but especially for the incubating companies. Interviews and the situation awareness framework were used as profiling tools. Focus on business strategy and technology is obvious among the 12 studied companies; however, pedagogy and context are not taken into account to the true extent of the need: mature companies were better balanced, but otherwise pedagogy-aware companies seem to be rare. Previous entrepreneurial experiences and the company's own R&D input increased awareness and facilitate skills to evaluate disruptive potential of new technologies and proper timing, i.e. improved strategic and technological awareness, whereas having close relationships to schools, a background in teaching or pedagogy explained good pedagogical awareness of the minority of enterprises that took such perspectives into consideration. Furthermore, a lack of sites for piloting and proper feedback mechanisms led to inadequate conceptions of user needs, preventing companies from developing learning solutions to their full potential.

Agency for Technology and Innovation (TEKES) set up SysTech project to foster interaction between developers, teachers, and researchers. Developers are released from arranging pilots, teacher needs are better taken into account through focus group participation, and researchers have an opportunity to bring research findings and curriculum awareness to companies by collecting material from real-life projects, and analyzing it for research purposes. The ultimate goal is to develop a co-creation model and principles with mutual benefit for all participants.

The best narrative has a happy ending. The definition may be better products, pleased customers, deeper learning outcomes, the improved sales of participant enterprises, a thicker value network, and finally as a bonus, the next phase of the project, piloting abroad and aid in internationalization. As a good PISA achiever Finland has already now a good reputation in education, with systematic development the reputation may be maintained and improved still. Further study should include evaluating the process, improving it iteratively, and developing a standard for good learning solutions with proven effectiveness.

**Acknowledgments** The research reported in this article has been funded by TEKES, SysTech programme. Grateful acknowledgement for proofreading and contributing valuable insights goes to Jennifer von Reis Saari

## References

- Bennett, S., Maton, K., & Kervin, L. (2008). The 'digital natives' debate: a critical view of the evidence. *British Journal of Technology Education*, 39(5), 775–786.
- Berners-Lee, T. (2010). Long Live the Web: the call for continued open standards and neutrality. *Scientific American* November (22).
- Biggs, J. B. (2003). *Teaching for quality learning at university: What the student does* (2nd ed.). Philadelphia: Society for Research into Higher Education.
- Boje, D. M. (2001). *Narrative methods for organizational and communication research*. London: Sage Publications Limited.
- Boje, D. M. (2011). Our organizations were never disenchanted: enchantment by design narratives vs enchantment by emergence. *Journal of Organizational Change Management*, 24(4), 411–426.
- Brush, C. G. (2008). Pioneering strategies for entrepreneurial success. *Business Horizons*, 51(1), 21–27.
- Carr, D. (2007). Computer games in classrooms and the question of cultural baggage. *British Journal of Educational Technology*, 38(3), 526–528.

- Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012). A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, 59(2), 661–686.
- Dalgamo, B., & Lee, M. J. W. (2010). What are the learning affordances of 3- d virtual environments? *British Journal of Educational Technology*, 41(1), 10–32.
- Daniel, D. B., & Woody, W. D. (2013). E-textbooks at what cost? Performance and use of electronic v. print texts. *Computers & Education*, 62(0), 18–23.
- Durso, F. T., Nickerson, R. S., Schvaneveldt, R. W., Dumais, S. T., Lindsay, D. S., Chi, M. T. H. (Eds.). (1999). *Situation awareness* (Ch. 10, pp. 283–314). West Sussex, England: John Wiley & Sons.
- Dutta, D. K., & Crossan, M. (2005). The nature of entrepreneurial opportunities: understanding the process using the 4I organizational learning framework. *Entrepreneurship: Theory and Practice*, 29(4), 425–449.
- Earle, R. S. (2002). The integration of instructional technology into public education: promises and challenges. *ET Magazine*, 1, 5–13.
- Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37(1), 32–64.
- Flac, J. M. (1995). Situation awareness: proceed with caution. *Human Factors*, 37(1), 149–157.
- Gartner, W. B. (1993). Words lead to deeds: towards an organizational emergence vocabulary. *Journal of Business Venturing*, 8(3), 231–239.
- Gartner, W. B. (2007). Entrepreneurial narrative and a science of the imagination. *Journal of Business Venturing*, 22(5), 613–627.
- Hannon, P. D., & Atherton, A. (1998). Small firm success and the art of orienteering: the value of plans, planning, and strategic awareness in the competitive small firm. *Journal of Small Business and Enterprise Development*, 5(2), 102–119.
- Hjorth, D., & Steyaert, C. (2004). *Narrative and discursive approaches in entrepreneurship: A second movements in entrepreneurship book*. University of Illinois at Urbana-Champaign's Academy for Entrepreneurial Leadership Historical Research Reference in Entrepreneurship.
- Kearney, R. (2002). *On stories*. London: Routledge.
- Krentler, K. A., & Willis-Flurry, L. A. (2005). Does technology enhance actual student learning? The case of online discussion boards. *Journal of Education for Business*, 80(6), 316–321.
- Lee, E. A.-L., Wong, K. W., & Fung, C. C. (2010). How does desktop virtual reality enhance learning outcomes? A structural equation modeling approach. *Computers & Education*, 55(4), 1424–1442.
- Lee, Y.-H., Waxman, H., Wu, J.-Y., & Lin, G. (2013). Revisit the effect of teaching and learning with technology. *Educational Technology & Society*, 16(1), 133–146.
- Linder, S. P., Abbott, D., & Fromberger, M. J. (2006). An instructional scaffolding approach to teaching software design. *Journal of Computing Sciences in Colleges*, 21(6), 238–250.
- McKinsey. (2009). *Shaping the future: How good education systems can become great in the decade ahead*. Singapore: McKinsey & Company.
- Mikropoulos, T. A., & Natsis, A. (2011). Educational virtual environments: a ten- year review of empirical research. *Computers & Education*, 56, 769–780.
- Miles, M. B., & Huberman, A. M. (1984). *Qualitative data analysis: A sourcebook of new methods*. Beverly Hills: Sage.
- Mylläri, J. (2012). Towards mobile curriculum with systemic learning solutions. In M. Specht, J. Multisilta, & M. Sharples (Eds.), *mLearn 2012 Conference Proceedings: 11th World Conference on Mobile and Contextual Learning* (pp. 280–283). Helsinki: University of Helsinki.
- OECD (2011). *PISA 2009 at a Glance*. OECD Publishing. [http://www.oecd-ilibrary.org/education/pisa-at-a-glance-2010\\_9789264095298-en](http://www.oecd-ilibrary.org/education/pisa-at-a-glance-2010_9789264095298-en). Accessed 25 Aug 2011.
- Osawa, Y., & Miyazaki, K. (2006). An empirical analysis of the valley of death: large scale R&D project performance in a Japanese diversified company. *Asian Journal of Technology Innovation*, 14(2), 93–116.
- Politis, D. (2009). Entrepreneurs' attitudes towards failure: an experiential learning approach. *International Journal of Entrepreneurial Behaviour & Research*, 15(4), 364–383.
- Riessman, C. K. (1993). *Narrative analysis* (Vol. 30). Newbury Park: Sage Publications, Incorporated.
- Romiszowski, A. J. (2004). How's the e-learning baby? Factors leading to success of failure of an educational technology innovation. *Educational Technology-Saddle Brook Then Englewood Cliffs NJ*, 44(1), 5–27.
- Sainio, L.-M., & Puumalainen, K. (2007). Evaluating technology disruptiveness in a strategic corporate context: a case study. *Technological Forecasting and Social Change*, 74(8), 1315–1333.
- Sainio, L. M., Ritala, P., & Hurmelinna-Laukkanen, P. (2012). Constituents of radical innovation—exploring the role of strategic orientations and market uncertainty. *Technovation*, 32(11), 591.
- Salden, R. J., Koedinger, K. R., Renkl, A., Aleven, V., & McLaren, B. M. (2010). Accounting for beneficial effects of worked examples in tutored problem solving. *Educational Psychology Review*, 22(4), 379–392.
- Starr, L. (2010). The use of auto-ethnography in educational research: locating who we are in what we do. *Canadian Journal for New Scholars in Education*, 3(1), 1–9.

- Stutsman, N. (2013). BYOD: one year later. *Technology & Learning*, 33(7), 36–39.
- Watson, D. M. (2001). Pedagogy before technology: re-thinking the relationship between ICT and teaching. *Education and Information Technologies*, 6(4), 251–266.
- WEF. (2010). *The global competitiveness report 2010–2011*. Geneva: World Economic Forum.
- Wenger, E. (2000). Communities of practice and social learning systems. *Organization*, 7(2), 225–246.



# Publication II

Niemelä P., Di Flora C., Helevirta M. and Isomöttönen V., “Educating future coders with a holistic ICT curriculum and new learning solutions”, *Journal of Systemics*, 2016

DOAJ: <http://www.ingentaconnect.com/content/doaj/16904532/2016/00000014/00000002/art00004>  
Niemelä et al. (2016)



# Educating future coders with a holistic ICT curriculum and new learning solutions

Pia NIEMELÄ  
Computer Science, Tampere University of Technology  
Tampere, Finland

Cristiano DI FLORA  
Rovio  
Helsinki, Finland

Martti HELEVIRTA  
Tampere, Finland

and

Ville ISOMÖTTÖNEN  
Mathematical Information Technology, University of Jyväskylä  
Jyväskylä, Finland

## ABSTRACT

Technology-orientation and coding are gaining momentum in Finnish curriculum planning for primary and secondary school. However, according to the existing plans, the scope of ICT teaching is limited to practical topics, e.g., how to drill basic control structures (if-then-else, for, while) without focusing on the high level epistemological view of ICT. This paper proposes some key extensions to such plans, targeted to highlight rather the epistemological factors of teaching than talk about concrete means of strengthening the craftsmanship of coding. The proposed approach stems from the qualitative data collected by interviewing ICT professionals (N=7, 4 males, 3 females), who have gained experience of the industry needs while working as ICT professionals (avg=11.3 y, s=3.9 y). This work illustrates a holistic model of ICT teaching as well as suggests a set of new methods and tools.

**Keywords:** ICT curriculum, teaching ICT in primary and secondary school, concept maps, UML, holistic ICT model

## 1. INTRODUCTION

The new curriculum with information and communication technology (ICT) as its focus is currently being reviewed and prepared for publication. The need of more ICT experts in industry has been recognized in decision-making by governing bodies. Not only are various domestic directions promoting ICT education but also the EU and multinational corporations have been actively pursuing new instructions and assessment of e-skills. For example, the EU has outlined a strategy for improving e-skills for the 21st century to foster competitiveness, growth, and jobs.

Moreover, in Finland distinguished pedagogues of especially the University of Helsinki [1, 2] are promoting more student-centered, informal learning: tablets for all students, using online material and social media to co-create in order to gain better ICT and multi-literacy skills. Future needs have guided the planning of the becoming 2016 curriculum. The way of working

and living is rapidly changing, and the need for curriculum change is acknowledged. Familiarizing students with technology and learning the basics of coding will be started already in primary school and the skills gained are further strengthened at the secondary level.

In the ICT curriculum, digital literacy and ICT skills are meant to be built gradually, starting from visual coding and tactile learning followed by a more formal approach at the secondary level, where ICT is integrated into math teaching. Hence, programmable calculators and other computational features are well represented in the curriculum plans. The introduction of new ICT concepts by experimenting relies on “Learning by doing” methodology. Graphical and other high-level languages with additional libraries meant for education are utilized.

Learning goals are divided as learning packages that consistently build up the basics of computer science a grade by grade at the secondary level. For example, the 7th grade aims at acquainting pupils with such computing fundamentals as statements, data types, the sequential execution of the program, ‘if-then-else’ flow control structures, and finding errors in syntax and correcting them. Basics of logic are introduced, starting from a truth value of a sentence. In the 8th grade, variables and functions are introduced. State machines are used and visualized concretely by playing with the construction kits (e.g. the switch states ON/OFF). Logic continues with deduction and reasoning. In the 9th grade, new variable classes such as collections, conditional iteration (while, do, for), and recursion are introduced. At a more general level, the learning goal is to model a problem and divide it into smaller executables. For gaining the craftsmanship of coding the planned approach sounds viable, but regarding of the whole palette of needed ICT skills the view is regrettably narrow.

## 2. RESEARCH QUESTIONS

The current proposal for ICT curriculum emphasizes gaining the craftsmanship of coding with small and valid incremental steps. The order of propagation is well-justified, but still arguable. Instead of addressing all the possible aspects of coding and computer science learning primitives in detail, the epistemology

of the ICT teaching as a whole should be discussed to consider all essential higher level needs. It is important to ensure that these findings are sanity checked and that the discussion is not lead by pedagogues only, but validated and augmented by ICT professionals, who know the industry needs. The research questions to be replied are:

1. Which kind of model would give a more holistic view of ICT epistemology?
2. How to support the learners in their becoming not only good coders but also good software architects and designers?
3. Which kind of learning solutions would support the ICT teaching model proposed?

### 3. INDUSTRY NEEDS RULED IN

To get a better grasp of the current ICT landscape we interviewed seven ICT professionals by email, six of whom are software developers and one a program manager. The email questionnaire contained the following questions: What are the ICT skills needed today/in the future? Which are your best ICT courses/informal learning experiences? How should ICT be taught in the primary school? With seven replies, we are far from scientific significance and based on the data only rough recommendations can be given. However, the anticipated holistic model can be verified by referring to the answers. Based on the replies, we classified ICT related capabilities to four categories: the craftsmanship of coding, modeling, user-centered design, and project management. More generic skills such as critical thinking, future working qualifications, and global citizenship were also mentioned, however, not taken into account here as they were regarded more as all-encompassing, general capabilities to be taught in other subjects as well.

Since the majority of the interviewees represented the implementation side, the craftsmanship of coding perspective was well pronounced. In a list of needed skills, web computing was mentioned six times, followed by data structure & algorithms (3), testing (2), and mobile coding(1). Among the most common computer languages Java (2), JavaScript (2) and C++ (1) were listed, but also specialties such as Rust (1), Clojure (1) and Go (1) received votes. High-availability engineering (1) and the ability to develop games (1) were seen as useful system level capabilities. Modeling was mentioned four times: design, UML, architecture, and being able to recognize meaningful entities were listed. User-centered design occurred once in the form of “*understanding the needs of the customer and managing them*” emphasized by the only project manager involved, who considered also project management and selling as important skills.

Our interviewees regarded hands-on experience as the main building block in learning. They would include in the ICT lessons of primary school e.g. team work and pair-programming exercises, increase motivation and inspiration by providing good examples, combine ICT with sports, and have students build their own e-portfolios. Working in teams or in pairs kids would learn informally scaffolding each other in the zone of proximal development [3]. Regarding good learning experiences the importance of teamwork was emphasized (3), especially pair programming with friends having a similar level was considered rewarding.

From the formal side the basic courses in the beginning were found the most meaningful (2), and those teaching techniques that remain the same regardless of the language such as data structures and algorithms (3) were valued high. Nevertheless, we also received critical views regarding ICT teaching, for example: “*At high school I never attended any good ICT courses. But all the math and physics at school helped me to learn problem solving and how to break down a problem in multiple pieces.*” According to the interviews, the future is drifting in the direction of HTML5 (2), robots (1), internet of things (1), and visualizations (1). These findings were classified into four main categories.

### The craftsmanship of coding

In the discipline of handicrafts and craftsmanship, learning happens through doing by hand, which is seen as a way of leveraging innovation and the creativity. Theories such as intelligent hands [4] and learning by doing, are the basis for the tactile learning language. In maths, the tactile exercises such as fraction pieces and decimal system learning tools are used while approaching the symbol language more in-depth. In ICT, bridging the connection between electronics and coding may be achieved with the help of different assembly kits (e.g. LEGO MindStorm and Robots, Arduino, Lilypad, littleBits). Electronic components, such as light emitting diodes, buzzers, and couplers can be controlled by coding and give a more concrete and clear response than visible feedback on a computer screen. As one of our interviewees puts it, “*Learning by doing simply cannot be beaten in efficiency.*”

In addition to construction kits, visual programming languages may be used as primers. Scratch, for example, provides graphical support for a user preventing the faulty code or the connecting of incompatible code sequences. Control structures (such as if-then, for, while) are ready-built, a user only has to adjust parameters, such as counters in iteration loops. Visual programming languages are limited in freedom of degrees, which at the initial learning stage will be good to minimize the cognitive load: time is not wasted hunting syntactic errors. In the long run, the conciseness of such languages starts to restrict freedom and creativity indicating the due date to expand to more expressive programming languages.

### Conceptual modeling as a software architect

On the authority of our interviewees, the development of ICT talent requires strong modeling and conceptualization skills. The highlighted modeling skills were designing, mastering UML modeling language, being able to separate relevant entities and build an architecture of systems. Regarding thinking skills mentioned ‘logical and critical thinking’ overlap partly with the conceptualizing skills, too. Therefore, we propose conceptual modeling as one of the key expertises and concept and mind mapping as its preceding preparatory skill. However, having good conceptualizing and modeling skills is not useful only in ICT, but in deep learning in general, the biggest difference between expert and novice thinking being the consistency and density of underlying concept schemes. Deeper learning implies linking atomic details as bigger and more robust constructions. Tying new knowledge to relevant concepts and previous propositions makes learning more meaningful.

## User-centered design to take into account real user needs

The domain of user-centered design was duly emphasized by the only project manager interviewed (“*understanding customer’s needs and use*”, “*selling*”). A real innovation takes the user’s needs into account, it focuses on the user and his context and incorporates his perspectives during the whole design process and new applications may be seen as innovations. Future coders need user-centered design tools to be able to innovate apposite applications. When the design is to be more user-centered, an essential prerequisite is to know how a customer acts by perceiving the typical work sequence and processes. This can be done by observing the environment and interviewing users i.e. becoming more aware of user needs.

In software projects, when the snapshot of the situation in its entirety has been obtained and user needs are detected as detailed manner as possible, the needs are dressed as formal use cases and requirements, which are a starting point to the following implementation. Ultimately, the final project achievements are checked against the use cases. When successful, the intended new solution provides added value and more efficiency by going beyond current practices.

## Project management

The golden rule of project management consists of planning, organizing and controlling, efficiency being the ruling principle both time and money-wise, as our project manager states: - Line out, what is needed and how to get the most efficient (both workload and budget) solution. Less is needed in direct implementation, more having the whole picture with technical knowledge about possibilities.

Project management is also seen as an interface between the customer and the team and good communication between different stakeholders is his responsibility. In addition to taking care of the good communication within the team it is also crucial to involve the customer in the communication loop to ensure that the user needs are fulfilled. Often the needs get more detailed - or even changed - during the process. The current de facto standard in project management is an agile methodology that embraces self-organizing teams that are capable of managing themselves. More and more, in agile projects people are working in remote teams that are self-directed.

Agile methodologies target flexibility in taking into account the moving target. A customer may change his mind during the implementation that is called requirement volatility. By iteratively ensuring that the direction is right and the product will better respond to the need, the project management is linked with the user-centered design, too. Albeit of the biased sample (only one project manager interviewed) the interview data confirmed what was expected i.e. that the project management is more concerned about the user needs and communication with the user than developers. Interviews also suggest that the project management level is more situation aware and concerned about retrieving the big picture, whereas developers think more in terms of technical solutions.

## 4. THE HOLISTIC ICT TEACHING MODEL AND NEW LEARNING SOLUTIONS

The proposed teaching model is depicted in Figure 1. The most elementary building block, the craftsmanship of coding, starts

with elementary exercises that combine both the visual coding and tactile learning objectives in order to provide a robust hands-on experience to build the base. When targets gradually grow more complicated or a bigger team is involved in coding, students will have to learn how to model and easily communicate the system structure. To this aim we propose the building block of conceptual modeling. The third thread illustrates the need for user-centered innovations. To be able to innovate, the student has to make observations in order to become acquainted with the customer needs, and to depict the underlying processes. After that optimizing and improving them is enabled, and ultimately, students come up with new, more efficient approaches to the problems and challenges.

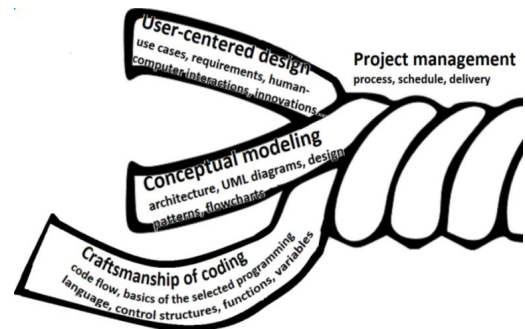


Figure 1: The proposed holistic model for ICT curriculum in primary and secondary school.

Together the threads form the cord of an ICT professional more capable of handling ICT projects successfully. The entwined threads are woven together by project management which includes controlling the process (e.g. using agile methods), being able to divide the project in smaller tasks, scheduling and staying within the deadlines. Next we will examine how these missing threads of current ICT curriculum of modelling, user-centered design and project management may be introduced in the school environment.

## Various concept map techniques as knowledge building tools

In computing good software architects are good modelers, whereas in school excellent students are good conceptualizers. In recent years, concept maps have been recognized as an effective visual learning tool that helps learners memorize and organize knowledge. Åhlberg [5] recommends maps in situations requiring data parsing and argues that they illustrate the conceptual and propositional structure of written text. He also considers conversions to both directions, from text to a map and vice versa, as a good way to work on and elaborate meanings. In addition to visually appealing and easy to use concept map applications (bubbl.us, MindMup, CMapTools), the tools of ICT professionals such as the Unified Modelling Language (UML) may be introduced to students as a modelling tool that is applicable in ICT teaching in particular.

Modeling skills are important for communication purposes, too, both between developers and other stakeholders. Nowadays, as the development is often geographically distributed, communications skills are extremely necessary. Even if one is not going to be a developer himself, a better understanding

about the overall structure with a map representation gives a quick generic overview and more means to communicate with developers.

In education, we also promote the use of concept mapping as an assessment tool. Currently, assessment is based on the student's test results and activity during lessons. This method is good at measuring whether smaller learning objectives have been reached. However, a more detailed development of conceptual understanding and getting a bigger picture is less frequently examined. Meaningful learning results are achieved when a person consciously and explicitly ties new knowledge to relevant concepts and previous propositions they already possess. Deeper learning implies linking atomic details as bigger and more robust constructions. A concept map can be understood as a visualization of a mental concept construction.

#### **Authentic tasks facilitating user-centered design**

Often real world problems are open ended, complex, require extended knowledge and emergent solutions. According to the Engagement Learning Theory by Kearsley and Shneiderman [6], a valid way of getting students committed is to provide meaningful, creative, and authentic tasks, which reflect the interests of students themselves, as much as possible. Finding an achievable yet interesting project target can prove to be the hardest part of the whole project, but once the goal is set, the group may start working backward from it [7], e.g. planning, sharing the goal into smaller executables, scheduling these parts, and finally moving to implementation.

User-centered design implies the participatory methods of inquiry of real - not self-determined - customer needs, modeling the situation, and means of improving current practices. For younger students, gaining the required level of situation awareness is a meta-cognitively demanding task and requires such future working skills as communication and collaboration. In software projects, when the snapshot of the situation has been obtained, the demands are dressed as formal use cases or lighter user stories and requirements, which are a starting point to the following implementation.

The authenticity of the project is increased with a real customer, who is interested in getting the thing done, is waiting for the end product and giving feedback. A real and useful innovation takes user's needs into account; it focuses on the user and his context and incorporates his perspectives during the whole design process, hence providing an opportunity of producing what Scardamalia et al. [7] call emergent outcomes, i.e. through new considerations, creativity and sustained work something brilliant might emerge. However, the emphasis here is not on the outcome but rather to familiarize with project work practices, learn how to co-operate, communicate, and take responsibility.

Nousiainen [8] emphasizes that the continuous involvement of users is a goal as such but also a way to empower users and promote workplace democracy and the means to practice the working life skills of participation, collaboration, and communication. In some schools, students have already started to innovate e.g. new means to recycle [9]. If no customers are available, suitable activities such as school projects could be e.g. doing your own online textbook or encyclopedia or pilot tests of innovative new technologies and rating them. Often the publicity of putting the end result on the web provides a sufficient incentive for the students to do their best. Strategically

oriented students might see this also as a sweet spot of exposing their skills for potential employers and polish their CV.

#### **Engaged students as self-directed project managers**

Several educators agree along with industry representatives that there is a gap between current education and future work skills. Moylan [10] identifies "Project Learning" as a key methodology for closing this gap between current curriculum and developing their necessary knowledge and skills essential for success in the 21st century. Among other forces, President Obama also instructs schools to transfer from trivial bubble filling exercises to such 21st century skills like problem-solving, critical thinking, entrepreneurship and creativity: the school has to train children for future challenges and work [11].

Project management may be practiced by open assignments i.e. with school projects that have to be scheduled and delivered in a timely fashion. In such projects, students should be responsible for the acquisition of information needed, for example, by interviewing their intended customer or searching data from the net. It would be good to attempt to raise the abstraction level by modeling. For schools to adopt agile project working style, students would come together to work on an ICT project that they have selected by themselves, plan and share the work, take on roles that play to their strengths and interests and then implement and solve problems together till the project is done.

Being ultimately learner-centered this type of learning assignment requires a very different approach of the teacher compared to traditional classroom instructions. The teacher is no longer the lecturer and the decider, but rather a facilitator or coach learning alongside students; sometimes to give up the control may feel like a farmer that is selling his farm. However, the effectiveness of instructor-led lessons and lecturing has long been questioned and as a subject ICT is one of the best suited for applying project oriented learning style. However, we note that often transferring to an open-ended working mode requires some practicing and external pressure to get things done, whether the agent generating the pressure is a teacher or preferably an external customer.

## **5. CONCLUSIONS**

Finland is planning to enrich its primary and secondary school curriculum with coding in order to prepare students for the future working life. Instead of settling on only the basics of coding, e.g. code flow and control structures, we claim that the ability to innovate and design software systems is at the very heart of software engineering. By adding the key areas of modeling and user-centered design to instruction schemes, we create a more holistic ICT curriculum. The conception and modeling ability is needed not only in ICT but in knowledge building and conceptual thinking in general. User-centered design improving the practices of one's own environment may also be seen as a tool of empowerment. An empowered member of the society, who is aware of user needs, will also become more innovative.

Hands-on experimentation was also considered beneficial among our interviewees: games, pair-programming, and learning from others informally were seen as ways to foster learning and engagement. Innovativeness and creativity are buzzwords used in curriculum planning, often in accordance with arts and crafts, which are assumed to enhance them. With

new methods and learning solutions, creativity may be fostered with STEM subjects as well: building robots, making animations, and playing and even developing games (e.g. Angry Birds Space to assimilate gravity basics) are new, engaging and motivating ways of learning.

Since many students are passionate about playing, games as new learning solutions have proven to be very powerful. High motivation and engagement appear as a “flow” while playing. Fu et al. [12] examined engaging games and listed properties such as immersion, the clarity of the goal, autonomy, feedback, challenge, and social interaction as the ingredients of flow. Moreover, it has been reported that games can have a positive impact on pupils’ perceptual templates, knowledge acquisition and affective outcomes [13]. By including the suitable features of games in learning environments, serious education may transform to edutainment. The level of shared fun increases interest in ICT in general and ICT learning objects may be gamified, too.

Coding starts in Finnish schools in the autumn of 2016. ICT classes become a laboratory of new learning tools and methodology. By documenting experiments and applying continuous development cycles, we may iteratively improve the learning results. In addition to teaching to code, it is necessary to introduce new tools for modeling and user-centered design.

## 6. REFERENCES

- [1] K. Hakkarainen, L. Hietajärvi, K. Alho, K. Lonka, K. Salmela-Aro, "A Paradigmatic Analysis of Contemporary Schools of IS Development", **International Encyclopedia of the Social & Behavioral Sciences**, pp. 918-923. 2015.
- [2] L. Krokfors, "Learning. Creatively. Together", **Educational Change Report 2016**. 2015.
- [3] L. Vygotsky, "Interaction between learning and development", **Readings on the development of children**, 23.3: pp. 34-41. 1978.
- [4] L. Hyde, **Making it**, New York Times, 6. 2008
- [5] M. Åhlberg, **Concept maps as a research method**. 2002.
- [6] G. Kearsley and B. Shneiderman, **Engagement Theory: A Framework for Technology-Based Teaching and Learning**. 1999.
- [7] M. Scardamalia, J. Bransford, B. Kozma and E. Quellmalz, "New assessments and environments for knowledge building," in **Assessment and Teaching of 21st Century Skills**. Springer, pp. 231-300. 2012.
- [8] T. Nousiainen, **Children's involvement in the design of game-based learning environments**. 2009.
- [9] M. Aineslahti, **A Journey in the Landscape of Sustainable School Development**. 2009.
- [10] W. A. Moylan, "Learning by Project: Developing Essential 21st Century Skills Using Student Team Projects", **International Journal of Learning**, vol. 15. 2008.
- [11] B. Trilling and C. Fadel, **21st Century Skills: Learning for Life in our Times**. John Wiley & Sons. 2009.
- [12] F. Fu, R. Su and S. Yu, "EGameFlow: A scale to measure learners' enjoyment of e-learning games", **Computers & Education**, vol. 52, pp. 101-112. 2009.
- [13] T. Connolly, E. Boyle, E. MacArthur, T. Hainey, & J. Boyle, "A systematic literature review of empirical evidence on computer games and serious games", **Computers & Education**, 59(2), 661-686. 2012.

# Publication III

Niemelä P., “All Rosy in Scratch Lessons: No Bugs but Guts with Visual Programming”,  
*Frontiers In Education Conference Proceedings, 2017*

DOI: [10.1109/FIE.2017.8190612](https://doi.org/10.1109/FIE.2017.8190612)

Niemelä (2017)

# All Rosy in Scratch Lessons: No Bugs but Guts with Visual Programming

Pia Niemelä, Pervasive Computing, Tampere University of Technology

**Abstract**—This case study addresses motivational issues in the elementary computing that follows UK National Curriculum of Computing (UKNC) at one of the international schools in Asia. The study examines different motivations and their impact on learning outcomes. Started in Year 8, Scratch was used as a computing primer, followed by the Khan Academy’s JavaScript, and Python basics. In order to study the learning process, surveys, interviews, and the analysis of the Scratch coursework were employed. Based on the results, Scratch provides a useful tool for scaffold programming basics and for fostering motivation in all student groups. The discontinuity point from visual to textual programming appears to be problematic: textual programming with JavaScript and Python seems to engage mathematically talented students who developed intrinsic motivation, and disengage several others, mainly because of felt incompetence. A few students with authentic interest areas, such as design, animation, or social media, engage inadequately after transition. In planning the syllabus, it is crucial to address motivational aspects as well.

**Keywords**—K-12 computing syllabus; visual programming; Scratch; JavaScript; Python; SDT theory; intrinsic motivation

## 1. Introduction

Our way of working and living is rapidly changing and school curriculum must adapt accordingly. Familiarizing students with computing basics should begin already in primary school. The secondary level then strengthens the gained skills further and enables differentiation. In learning, the attitude toward the subject and the perceived sense of self-efficacy are crucial in terms of good learning outcomes. In contrast, students’ attitudes towards computing tend to become adverse in many early adapter countries, which reflects in the learning motivation. After including the subject in the elementary school curriculum, the enrollments of computing courses decreased alongside the attenuation of attitudes; examples include South Korea [1], the UK [2] and U.S. [3]. Although, U.S. has recently managed to switch the declining trend with determined actions, such as initiatives of Hour of Code, CS4All, and Hack Clubs [4]. In addition, an appealing tool selection, such as visual programming, seems to lure more students into CS classes, and to increase motivation and self-efficacy [5].

Negative attitudes are common not only among students, but also among teachers [6]. Rapid changes in society and new requirements induce resistance to change. Were the attitudes favorable, low computing competence would prevent

from progressing [7]. Well-designed tools for novices lower the learning threshold by hiding complex syntax of programming languages. Brown (2013) lists the need for more substantial support for computing teachers at school context, and the shared vision with clear subject goals among school personnel [8]. In the United Kingdom, computing teachers have founded a Computing-At-School (CAS) community, where they can share their concerns and get advice [9].

As an active member and contributor of the CAS community, Crick (2011) noted that the increasing amount of disengaged students calls for more excitement and wider perception in the whole science-technology-engineering-math (STEM) domain [10]. Implementing visually rewarding projects in other STEM subjects, demonstrates applicability of programming skills also outside computing classes. The CAS guide for secondary school (2014) states that creating presentations, websites or videos, are a viable way to add excitement providing public access [11]. Demonstrating the utility of skills in concrete applications feeds motivation. [12]. Since being motivated and engaged correlates with good learning outcomes [13], attempts to construct a good intrinsic motivation are essential for education.

Visual programming circumvents many disadvantages of traditional textual programming languages, such as susceptibility to errors. Fewer errors and a robust environment increase productivity, which in turn adds the self-efficacy of students. However, after the students have tasted the convenience of visual programming, they might need some extraneous assistance in moving to textual programming [14], [15], [16]. The present study examines the development of motivation and focuses on transition from visual to textual programming.

### 1.1. Research questions

In this study, I focus on the motivational factors behind the differences of learning outcomes in computing. By analyzing the surveys, interviews and coursework, the study answers the following questions:

- 1) What motivation categories can be found?
- 2) What is the role of visual programming in engaging students?
- 3) What are the means to maintain motivation with textual programming?

First, I review the motivation theories and correlation between motivation and learning outcomes. The Research Methods section opens up the context of this study and application of mixed methods. The Results section describes

the motivation categories based on the content analysis and the interview results. Analysis of Scratch coursework completes the Results. To conclude, the motivational aspects of learning are emphasized.

## 2. Development of Intrinsic Motivation

This study examines the development and manifestation of the motivation and early computing craftsmanship among school students. Differences in performance tend to grow during school years. The Self-Determination Theory (SDT, [17]) justifies this tendency by explaining the mechanisms of the development of the intrinsic motivation and partly by referring to the Intentional Learner Theory [18].

The innate psychological needs and structure of the person's psyche affect the development of intrinsic motivation, whereas external factors affect extrinsic motivation. External factors consist of environmental pressures, such as control and expectations, and after completing a task its rating and other anticipated feedback. Alternatively, the reward may also be more abstract or remote, for example, the appreciation of the community or a study place in the desired field. The Intentional Learning Theory [18] considers the long-term objective of gradual knowledge building more comprehensive than the intrinsic motivation, and denotes a "serious student" that sets subject-specific lifelong learning objectives. Regardless of motivation type, long-term goals sharpen the learning.

Extrinsic and intrinsic motivations are not mutually exclusive but complementary. Motivation type varies on each stage of learning. For example, in the beginning and at junction points, the extrinsic motivation is a trigger for the development of an intrinsic motivation so that a successful intrinsic learning process gradually takes over from the extrinsic one. Counter-intuitively, a number of studies [19], [20] claim that external rewards may even slow down the development of intrinsic motivation. For example, a student may start to think that a subject itself is not worth studying, if a teacher must reward the effort. In addition, rewards are seldom open-handedly distributed throughout the process. Was the motivation dependent on rewards, it would start to decline when rewards tail away.

According to the SDT theory, motivation is at its strongest in the overlap of three main components, competence, autonomy and relatedness. A motivated student feels more knowledgeable, being in control of his learning, and connected to other students. Competence is understood holistically to entail not only computing basics and specific tools, but being able to solve problems by decomposing them into smaller subproblems, identifying patterns, modeling systems and then gracefully implementing a desired artifact as a well-managed and timely delivered project [21].

Being able to set authentic goals supports autonomy, i.e., authenticity relates to autonomy. Authentic goals enhance engagement and empowerment. However, the goal must be achievable, so that a student either possesses or can acquire necessary skills to succeed on schedule.

Lastly, relatedness helps in transforming from extrinsic to intrinsic motivation. While intrinsic motivation is yet in its infancy, a student's relations to the significant others (i.e. friends, classmates, parents) may trigger and uphold motivation. Additionally, peers with the same level of competence scaffold the student. Relatedness is especially helpful in the beginning when the intrinsic motivation is still weak. For example, Haarala-Muhonen et al. (2011) recommended investing in the students' preparedness from the very beginning by using both student and teacher tutors [22]. Haatainen et al. (2013) decreased computer science dropouts by scaffolds, tailored for the first computer science course (CS1) at university [23]. The components of intrinsic motivation function in co-operation and influence each other in a spiral manner: competent students show more initiative and are eager to take on tasks, which again increases autonomy – autonomous learners take more responsibility for their own learning – which develops competency.

## 3. Research methods

The research questions are answered by surveying and interviewing Year 10 students of Hope International School in Cambodia in year 2016, and by analyzing their Scratch coursework. The students had followed National Curriculum computing syllabus [24] for two years by the time of conducting this research. They started with Scratch and continued with textual programming languages of JavaScript and Python. To get a grasp of the executed syllabus, the computing teacher was informally interviewed. His approach to teaching resembled genetic epistemology that necessitates progressions from concrete to abstract exercises (e.g. from Legos to Minecraft then to exploiting Python APIs that enable a programmatic construction of Minecraft artifacts) [25].

The UKNC syllabus defines the overarching learning targets, such as '*think creatively, innovatively, analytically, logically and critically*' and '*analyse problems in computational terms through practical problem solving, including designing, writing and debugging programs*' [26]. The syllabus does not dictate the use of any specific languages. Nevertheless, the selected programming languages hold an established position supported by Computing-at-School UK, as a course organizer and certifier. The author of this study did not influence teaching instructions, but carried out the interviews and examined the coursework retrospectively.

### 3.1. Qualitative analysis of surveys and interviews

The survey consisted of two parts: in the first part, the students answered open-ended questions regarding language preferences and the most useful computing skills. In the second part, they completed sentences to reveal the attitudes towards computing: "Coding is like...", "Scratch is...", "Scratch works best in...", "I like to code because...", "I do not like to code because...", and "Anybody needs ICT skills because..." The selection of survey informants was made by simply exploiting the group which was the easiest available,



i.e., the Extended Math class (N=16) that the author was teaching.

The content analysis of survey data aimed at identifying the underlying motivations. Similar arguments were grouped and the most descriptive words were selected as identifiers of each motivation category. The analysis led to the appropriate theoretical framework: the SDT theory and its intrinsic motivation part appeared to best explain the suggested categorization. Because of the content-rooted attachment of the theory, the analysis had a flavor of a grounded theory.

After these preparatory phases, the focus group (N=6) of the most skilled computing students were interviewed. The selection applied the snowball method: the teacher selected only the first student in the chain, and each student in turn named the next student of the opposite gender until a group of six was in size. During the interview, one of the students acted as a moderator and ensured everyone's contribution; no other people but the group were present. The questions were distributed beforehand, the first task being the review of initial motivation categorization. The review feedback shaped the final motivation categories.

The focus group interview was recorded and transcribed. The moderator proofread the transcription and completed the parts not caught by the audio typist. Consisting of the most prominent and intrinsically motivated programmers of Y10, the focus group and its interview could shed light on the genesis of intrinsic motivation. The most illustrative comments were quoted.

### 3.2. Quantitative analysis of Scratch coursework

Furthermore, the Scratch coursework provided means for triangulation in order to test the accuracy of intrinsic motivation predicting good performance. In this study, activity in Scratch is thought to reflect intrinsic motivation pursuant to the Free-Choice Paradigm [27]. The paradigm measures a student's intrinsic motivation based on his behavior when he considers not being observed. In this context, free-choice activities comprise extra projects, sent messages, comments, favorites, and followers/followees as a demonstration of intrinsic motivation. Similarly to the grading system with A-E that assumes the Gaussian distribution of human qualities, only one-fifth of the students are anticipated to have an 'A-level' intrinsic motivation. The activity score was reduced as intrinsic motivation simply by defining a threshold to result in one-fifth of the students. Activity that exceeded the threshold was regarded as an intrinsic motivation of value one; other users were marked as not intrinsically motivated, the value of zero. In comparison, the binomial data of intrinsic motivation correlated with Scratch scores illustrated with logistic regression. Logistic regression is apt for classifications, defining a student as intrinsically motivated or not.

In addition to the activity level, coursework is analyzed based on quality of the code. Initially, Dr.Scratch was the choice for scoring. Handily enough, the <http://drscratch.org/> website requires only the project id as an input. The output

comprises seven separate modules scored 0-3, resulting in the total of 21 points that divides into three levels: 'basic', 'developing', and 'mastery'. Nonetheless, Dr.Scratch did not provide a batch-processing mode necessitated by the number of projects. To execute analysis in larger quantities, an underlying Python plug-in, on which Dr.Scratch was built on, was exploited directly. This plug-in, called Hairball, claims to be lint-inspired [28]. Instead of a single indicator, it provides detailed information about the code. To reduce the amount, defining rubrics was indispensable, see Table 1.

Table 1. SCRATCH COURSEWORK RUBRICS

Max score	Block	Definition
3	BlockCounts	Divided to basic and advanced: Basic: Max score 2. Advanced blocks consist of broadcast-message, clone and touch Max score 1. Totals 3
1	DeadCode	$1 - \frac{\text{lines of dead code}}{\text{total lines}}$
1	Variables	Max 0.5 from sprites as variables Max 0.5 from other Totals 1.

Scoring should favor the utilization of desired and more advanced code structures, such as broadcast-recv message passing that enables concurrency combined with loops, e.g., a forever loop [29]. In concurrent programming, several agents are functional and may access the same resources simultaneously, which adds complexity. Agents and their interplay are central practices in agent-based programming, where Scratch categorizes in [30]. Scratch provides a clone operation for copying sprites. By cloning and setting rules for interactions, by sensing other agents (e.g. if touched), a student can implement simulations or games. Mastering advanced computing practices, such as concurrency and cloning, demonstrates progress in computational thinking skills of automation and algorithmic thinking, and prepares for multi-agent, event-driven programming with textual programming languages as well.

## 4. Results

The results are introduced chronologically, in the order of the survey (N=16), the focus group interview (N=6), and the Scratch coursework (N=54). In compliance with Scratch scores, other grades are reflected to widen the perspective. Based on the content analysis of the surveys and interviews, motivation categories are represented in the form of a diagram. From the categories, intrinsic motivation, a linchpin in developing expertise, is studied more closely by interviewing a focus group and analyzing Scratch coursework.

#### 4.1. The survey and initial motivation categorization

The survey consisted of sixteen Year 10 students. The content analysis of the survey data divided motivations inherently into four categories characterized with following descriptions:

- 1) Intrinsically motivated students find coding rewarding per se, enjoy challenges, and problem solving
- 2) Intentionally motivated students are willing to invest in skills needed for further studies, jobs, and career. Good grades, positive feedback, and merits to show in a CV are highly valued
- 3) Design/art students pursue self-expression. The students aim at achieving something amusing or visually attractive that might be used as a design for industrial products, e.g. T-shirt, ads, and architecture
- 4) Internet/social media users are eager to connect, browse, and share ideas with their peers

The intrinsic motivation correlates most with the computing preference. Intrinsically motivated students enjoy programming, whereas students with other motivations need external triggers to ignite interest. A number of intrinsically motivated students are also keen on math. In sentence completions, these students often expressed their enthusiasm in compliance with highlighting the applicability of new skills, such as 'Computing...':

Girl, intrinsic/intentional: ... *opens doors for me and enables me to make my way out of problems.*

Boy, intentional/intrinsic: ... *you can better understand what makes computers work and its logic.*

Boy, intrinsic: *I get the knowledge of how a computer works.*

In the interviews, disjoint categories of intrinsic and intentional motivations were clearly visible, but frequently manifested in the same students. Yet intrinsic motivation is more desirable from the subject's point of view, motivations are complementary: benefiting the skills excludes no genuine interest. However, a few intentionally (and not intrinsically) motivated students were capable of overlooking their dismay and frustration because of foreseen benefits, exemplified with the following comments that demonstrate this ambivalence:

Girl, intentional: *It is complicated to learn and has many different functions, but on the other hand: I know it will be an important skill to have in the future. ICT skills are important to have in your career.*

Girl, intentional: *Coding is very difficult and annoying when you are writing it but it is very relieving once you succeed. It is hard, you need to type a lot. But it is the trend nowadays and needed in society jobs, colleges, ..helps users think. People who use ICT skills have very developed thinking.*

The design and social media oriented students found their own territory of asserting themselves through

Photoshop exercises, tuning images and animations, albeit sometimes these students indicate lower intrinsic motivation. This group with the negative computing attitudes gave the following statements:

Girl, design: *It is too hard and I do not understand computing.*

Boy, design: *It is a separate language that I do not understand. I find it to be boring, and it doesn't interest me.*

The teacher checked the identified categories and acknowledged them with 'Nail on the head!'

#### 4.2. A focus group interview and final categorization

Our focus group consisted of five intrinsically motivated and one design-oriented student. The group got the initial diagram for review; see Fig. 1. The globe with speech bubbles intended to illustrate the internet category, i.e. being connected and sharing ideas. However, the focus group struggled the most with this category. They associated internet with browsing, considered a no-brainer, which raised the question of a role of computing in it, 'Personally, I don't think number four (internet) applies very well if you know how to program, because if you know how to program, internet browsing shouldn't be a problem.' The group was also slightly puzzled with 'design', 'design goes with the internet, cause the internet is in the end, and everything in the internet has been designed'.

Due to no saturation, the categories of 'Design/art' and 'Internet/social media' do not cover all the variety of potential interest areas. Were the students keen on audio/video editing or robots, the survey topics might contain 'Audacity', 'iMovie' or 'Arduino', which fit poorly in the current categories. By combining design and internet, and labeling the new category more generically as 'authentic self-expression' enables tackling the whole range of interests; the revised diagram below.

Intrinsically motivated students found coding (or the final product) so rewarding that an effort was made even if extrinsic motivation factors (support, feedback, rewards) were minor or squeezing. A remarkable part of the motivation was the foreseen applicability of computing skills in further studies and career, which complies with the intentional motivation. In accordance, external motivation factors played a role in shaping the motivation such as family pressure, good grades and the pressure of being tested. The next excerpt illustrates this: *I probably wouldn't stick with computing if I wasn't being tested and pushed by the fact that I'm doing it for the school as well.*

The focus group did not believe that becoming intrinsically motivated happens by accident. A computing course, an enthusiastic peer, or a tutor that paves the way was anticipated to trigger it. Even if the interview did not pronounce any submissive reasons for developing an intrinsic motivation, personal qualities seem to play a significant

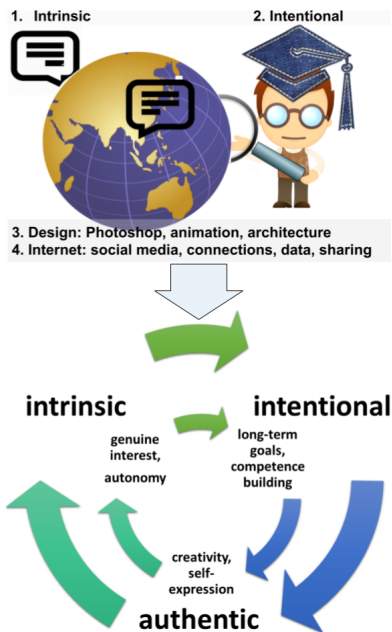


Figure 1. Motivation categories: the initial and revised versions

role. Math competence appears to correlate strongly and imply computing interest, which hypothesis is supported by grade correlations, the section 4.4, in particular between Additional Math and computing, as well as with intrinsic motivation indicators, such as activity.

In regard to motivation, the focus group was happy with moving from Scratch through JavaScript to Python, as addressed by a member *I would recommend using Scratch first, to show people how to think logically in an easy to learn environment. Then they should learn JS, as that's more like real coding, and can free people up from the limits of Scratch, while still being visual. Finally they should learn Python as that is more powerful than JS, but is easier to write without syntax errors. Scratch prepares for textual programming without 'overwhelming' them. In contrast, the survey group preferred Scratch to JavaScript and Python 'definitely more exciting/fun than Python or Khan Academy'.*

### 4.3. Scratch activity predicts good performance

The quantitative part focuses on the effect of intrinsic motivation on learning outcomes by comparing Scratch activity and scores (N=54), where high Scratch activity is considered to reflect high intrinsic motivation. In addition to the number of projects, factors such as sent messages, comments, followers, and followees, increment the activity score. In determining intrinsically motivated students, the activity threshold was tuned to include approximately one fifth of the students. This threshold was raised three times

higher with random MIT Scratchers revealing higher social activity of a typical user. Rather than for socializing, the Y10 students exploited Scratch for learning: they seldom contacted anyone else but their classmates or the teacher, and for any other reasons but advice. Compared with his students, the teacher was exceedingly active in giving them advice and preparing examples. In addition, he followed prominent Scratch contributors, thus scoring the highest in an activity-based evaluation.

One target of the Scratch coursework was to contribute and share projects publicly in the MIT database, where the visibility can be selected either public or private. Publicly shared projects expand the ever-increasing remix resources. Examining expert examples is an apt way of adapting new coding tricks. Demo selection provides opportunities of vocational growth for teachers as well. In addition to the Scratch community, the coursework provided cross-curricular contribution as well. It had agreed clients, who were, in essence, the STEM teachers of the school. They used projects to enliven their lessons, for instance, by showing simulations of laws of physics and chemistry.

The correlation between the intrinsic motivation and Scratch score was clear (see Figure 2). In the figure, x-axis

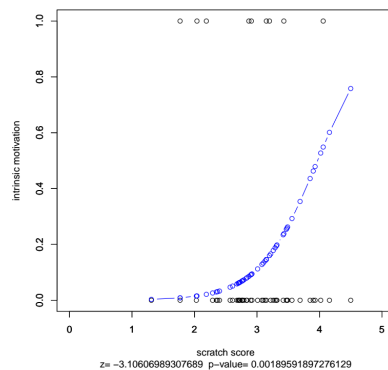


Figure 2. Intrinsic motivation (0,1), in x-axis the Scratch scores,  $p = 0.0019$

shows Scratch scores within range 0-5. The distribution of scores was Gaussian, checked graphically with a histogram 4. The students' coursework was assessed by utilizing the Hairball plugin with the criteria defined in Table 1. The correlation between intrinsic motivation and Scratch scores was statistically significant ( $p=0.0019$ ).

The Scratch projects in the MIT database can be employed as a reference point and a means to review the validity. In excess of projects (currently over 20 mill.) a random sample of N=1000 was considered adequate.

In contrast with the Y10 students, random data indicated no correlation between activity and Scratch score; see Figure 3. The typical S-curve straightened, and the significance of  $p\text{-value} = 0.1$  was less than the limit of 5 %, thus the correlation was lost.

Unlike the examined Y10 students, a typical MIT Scratcher focuses more on his social connections. In ad-

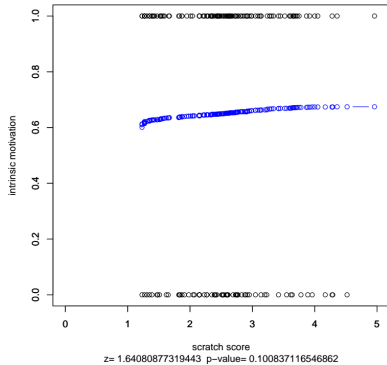


Figure 3. Random sample from MIT database revealed a different behavior,  $p = 0.1$

dition to a number of messages, comments and ratings, the followship of prominent contributors increases the activity score. The follower pattern of Scratch complies with preferential attachment theory [31], where followers distribute unevenly and cumulate in large quantities to few popular contributors. Including these experts, the Scratch user base is intensely heterogeneous, from kids to adults, many of whom just try out the tool. Evidently, a random Scratcher abandons a project easier without external pressure, that is, a teacher demanding the completion, getting a score, having a client, or sharing publicly. The amount of trash projects is high, thus selecting randomly results in a large amount of low scores. In this sample, the number of trash was remarkable.

In order to prove the significance of the quality difference, vertical lines illustrate the means of the samples. Moreover, a two-sample t-test determines the p-value. In the t-test, the null hypothesis states that the means are not significantly different, whereas the alternative hypothesis argues the opposite. The t-test gave p-value of  $7.1E-11$ , i.e., indisputably significant. Fig. 4 reveals the bimodal nature of the random sample illustrated by the pink histogram, where the first peak resides between scores 1 and 2. This peak consists of trash projects that are not necessarily meant to be shared; the latter peak represents the decent projects. To improve comparability, the private projects should have been omitted, even if the web store allowed the download.

#### 4.4. Other grades checked

For a validation of activity-predicted performance, the Scratch scores were contrasted with grades of other academic subjects. Letter grades of A-F were digitized in order to calculate the correlations: A+ equals to 4.33, A to 4, and F as “failed” equals to 0. Other grades linearly distribute in between. Being a member of the focus group was coded with the value of one, for non-members the value was 0. These substitutions enabled converting from nominal to interval data to calculate correlations.

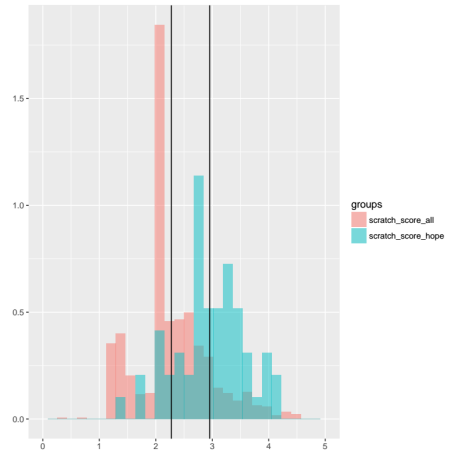


Figure 4. Histograms of Scratch scores; light blue bars describe the Hope School, whereas pink bars the random MIT sample

The correlation between formal computing grades and global perspectives grades was the strongest ( $r=0.52$ ), followed by geography( $r=0.50$ ). These two non-STEM subjects preceded correlations of science ( $r=0.495$ ) and math ( $r=0.40$ ). However, an essential aspect lost in calculations was the exclusiveness of Additional Math, selected only by five Y10 students, of whom four were in the focus group. The school provided three math levels in ascending order of complexity: Core, Extended, and Additional Math. To choose Additional Math, a student had to be an A-level student, thus the grades given in the Additional Math are not commensurate with the Core or Extended. Consequently, Additional Math participation was encoded as a dichotomy of on=1, off=0 to calculate the correlation,  $r=0.54$ , which was the strongest correlation with formal grades. After these correlation, activity, Scratch score, and focus group membership were checked. Activity and Scratch score correlated with computing grades poorly, activity  $r=0.05$ , Scratch score  $r=0.23$ ; whereas with AddMath excellently, activity  $r=0.60$ , Scratch score  $r=0.56$ ; and moderately with other math,  $r=0.31$  and  $r=0.29$ . Being part of the focus group correlated with AddMath ( $r=0.66$ ) the strongest, with other math strong as well ( $r=0.64$ ), and quite strong with computing ( $r=0.54$ ). Thus, correlations suggest a close linkage between intrinsic motivation and math as a subject.

## 5. Conclusion

**Which motivation categories can be found?** Categories include 1) intrinsic 2) intentional and 3) motivation based on authentic self-expression goals, often related to social media or design. Intrinsically motivated engage computing without special entertainments such as gamification or external rewards. Significantly, challenges that may hinder others have an opposite effect concerning intrinsically mo-

tivated students: hard as they struggle, the more motivated they become. As the teacher puts, 'Once they are engaged, you can get them to follow you through broken glass'.

Intentional motivation comprises a flavor of strategic thinking and opportunism mixed in, pronounced by such excerpts in interviews as: 'An easy A for my paper', 'Pushed by the tests', 'Computing is applicable to the future', and 'It's good for my CV'. The group with authentic goals have special interest areas and distinct requisites for which they needed computing skills. In this sample, these interests varied from Photoshop, animation and architecture to social media and connectedness. By providing sharing, remixing, rating, and searching options, Scratch increases the anticipated community feel and adds social aspects to engage its users. These features induce a flavor of social media to Scratch. Besides socializing, however, the database of existing Scratch work offers means for learning from experts, when intentionally used in this manner. In addition to the Scratch on-line community, various APIs (such as [api.scratch.mit.edu](http://api.scratch.mit.edu)) proffer programmatic ways of handling Scratch data and tailoring new kind of applications.

Lack of any afore mentioned motivations appears as disengagement, which was not common among the examined students.

**What is the role of visual programming in engaging students?** The students appreciate Scratch as a tool. They value it high, because of no bugs and thus feeling more competent. The following snippet of the focus group interview demonstrates this: **Alpha:** *If you do Python or JavaScript, right, you have more chances of failing. When you have Scratch, you can have later that... super-multiple layers, you know what I mean? And when you finish it, it works, it is like (\*a remarkable sigh\*). Yeah, I'm the best! (pause). You know that feeling...? **Drakvor:** *We get that feeling with Python, not with Scratch. **Alpha:** *It is like if you have a wrong indent in Python everything goes wrong.***

As robust as it is, Scratch has its pitfalls. Meerbaum-Salant et al. (2011) discovered that Scratch might lead to bad programming habits, such as a bottom-up development and fine-grained programming [32]. The authors emphasized that Scratch requires appropriate teaching methods and materials to reach its full potential. Nonetheless, programming with Scratch appears to be a success story, whereas further steps with JavaScript and Python need adjustments to avoid decline in motivation.

### **What are the means to maintain motivation with textual programming?**

In this study, the focus group (excluding the 'design' student) was eager to get grasp of more powerful and unconditional textual programming, whereas the motivationally heterogeneous survey group desired to keep Scratch, qualified as 'one of the fun things in your life', 'easy and visual', and 'the lego of coding'. To boost motivation, the focus group advised to highlight the benefits, for instance, inspiring outcomes, **Alpha:** *If he (the teacher) shows what the final outcome, if we do it in that class, could end up like.. it increases the motivation. **ActiveInFantasy:** *But you need**

*show people achievable things. **Alpha:** *Yeah, what they can achieve when they learn at that point.. **ActiveInFantasy:** *So if you start people on Python by saying 'Look at the Google Search Engine,' then they might be a bit overwhelmed.***

Applicability of the skills demonstrates in other school subjects as well, 'Our chemistry teacher had us make simulations of scientific experiments, which I think helped people see how their programming knowledge can be made useful in real life situations.' In addition, higher employability demonstrates applicability, as the computing teacher emphasized, 'More and more in the world today, you can get jobs easier, if you have these sorts of skills.'

Scratch may raise the threshold to transfer to textual programming: achievements may remain frustratingly modest due to frequent errors. To smooth the transfer, peer tutors are advantageous, and in addition to human help, tools and IDEs could provide extra scaffolding. Currently, active research focuses on tools bridging visual and textual programming with various angles of approach. In bridging, the mediated transfer between modalities of blocks and text can be either unidirectional or bidirectional. In a unidirectional tool, only the visual code, e.g. blocks, is editable (e.g. Blockly [33]); changes reflect automatically in the textual representation. A bidirectional tool modifies one representation based on the modification in another; examples include Droplet editor for Blockly [34], Pencil Code, and Code.org's App Lab [35].

In addition to textual code, illustrative visualizations scaffold gaining understanding of difficult concepts, such as lists and their indexing; difference of class and object; and control structures or the whole control flow of a program: Patch as a successor of Scratch provides such list visualization [36]. Frame-based coding offers an intermediate stepping stone in proceeding from visual to textual by structuring the code into blocks and by providing implementation hints [16], [37]. Greenfoot makes also such visualizations by elucidating e.g. the difference between an editable class and an instantiable objects [37]. Furthermore, visualizations need not restrict to code only, but block design can expand to cover electronic design as well, e.g. Scratch provides an Arduino expansion, Blockly goes with Picaxe that offers input/output simulations of circuits and flowcharts to visualize the control flow [38], and Microsoft MakeCode serves as a web-based programming platform similar to Scratch enabling the control of micro:bit components, which are embeddable and affordable microcontroller devices [39]. With the syntax guidance only, a student remains on the first steps of semiotic ladders, from where one should climb higher, to semantics and pragmatics. In climbing, the textual representation ought to be exploited to its full potential by bidirectionally connecting the block-based and textual representations and by explicitly transferring the gained programming experience, lest left untapped as a learning resource.

Climbing higher and striving for bigger and error-free systems necessitates design and testing. However, current Scratch supports poorly these attempts. In analysis phase, debugging tools should be as informative as possible: current state of variables and visualization of the control flow

enables finding errors effortlessly. Error messages should be descriptive and help in fixing. From new emerging transition tools, e.g. Patch addresses these issues by providing integrated tracing/debugging to assist algorithm development and implementation [36]. In preventing errors, test-driven development targets error-free code by mandating tests first, that students may first regard as work in vain. Hence, establishing the practice may require a disciplined approach from the teacher. The teacher should also focus on the most common misconceptions caused by block programming, such as 'loops are forever' [5]. Misconceptions include indentation-related issues that seem to cause problems in moving to Python. In assessing the visual programming outcomes, automatic analysis could be exploited, e.g., Dr.Scratch to be passed with the level of 'developing' or 'mastery' before getting a grade. In addition, cross-debugging is worth trying.

**Future directions** Tailoring and differentiating the computing syllabus based on the skill level and authentic interest areas would provide meaningful learning goals for all student groups. The syllabus could have a separate and concise core part common to all computing students. Optional modules would comprise e.g. in-depth computing for intrinsically motivated students. Intentional learners would appreciate modules for study and future work skills including e.g. efficient data search and visualizations, whereas students with authentic interests would benefit from more autonomy and space to demonstrate their creativity by implementing assessable digital artifacts. With more fine-grained offerings, computing frustration was partly preventable.

Intrinsic motivation seems to be a remarkable asset in developing expertise in computing. The computing syllabus should take into account motivational aspects and attempt to foster intrinsic motivation in particular. The emerging trends (such as Internet of things, cloud services, coauthoring tools) expose the ubiquitous and pervasive nature of computing; in the future jobs will be more and more digitized. Everybody needs technological skills that are emerging as new citizenship skills, society should allow no digital dropouts. It is essential to take into account different motivations and engage students with authentic interests that may trail computing enthusiasm. This is important to note: clever engagement exploits students' authentic interests to increase their motivation.

Curriculum planners should pay more attention to motivational aspects also here in Finland, where the new curriculum including computing has been in effect since autumn 2016.

## Acknowledgments

I thank Hope International School, both the personnel and students, for opting me the interviews and Scratch coursework, and Pervasive Computing Department and my supervisors for their support of interdisciplinary research topics. In addition, special thanks to the Academy of Finland (grant number 303694; *Skills, education and the future of work*) for their financial support.

## References

- [1] Jeongwon Choi and Sangjin An and Youngjun Lee, "Computing education in Korea — current issues and endeavors," *ACM Transactions on Computing Education (TOCE)*, vol. 15, no. 2, p. 8, 2015.
- [2] Gillian M. Bain and Graham Wilson, "Convergent pathways in tertiary education. What makes our students succeed?" *ACM Inroads*, vol. 8, no. 2, pp. 37–40, 2017.
- [3] Azad Ali and Charles Shubra, "Efforts to reverse the trend of enrollment decline in computer science programs," *The Journal of Issues in Informing Science and Information Technology*, vol. 7, pp. 209–225, 2010.
- [4] Greg Thompson, "Coding Comes of Age: Coding Is Gradually Making Its Way from Club to Curriculum, Thanks Largely to the Nationwide Science, Technology, Engineering and Mathematics Phenomenon Embraced by So Many American Schools," *THE Journal (Technological Horizons In Education)*, vol. 44, no. 1, p. 28, 2017.
- [5] Armoni, Michal and Meerbaum-Salant, Orni and Ben-Ari, Mordechai, "From Scratch to "real" programming," *ACM Transactions on Computing Education (TOCE)*, vol. 14, no. 4, p. 25, 2015.
- [6] Balanskat, Anja and Blamire, Roger and Kefala, Stella, "The ICT impact report," *European Schoolnet*, vol. 1, pp. 1–71, 2006.
- [7] Bingimlas, Khalid Abdullah, "Barriers to the successful integration of ICT in teaching and learning environments: A review of the literature," *Eurasia Journal of Mathematics, Science & Technology Education*, vol. 5, no. 3, pp. 235–245, 2009.
- [8] Neil Christopher Charles Brown and Michael Kölling and Tom Crick and Simon Peyton Jones and Simon Humphreys and Sue Sentance, "Bringing computer science back into schools: lessons from the UK," in *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013.
- [9] S. Krpan, Divna and Mladenović and G. Zaharija, "Mediated Transfer from Visual to High-level Programming Language."
- [10] Tom Crick and Sue Sentance, "Computing at school: stimulating computing education in the UK," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. ACM, 2011, pp. 122–123.
- [11] P. Kemp, Ed., *Computing in the national curriculum. A guide for secondary teachers*, 2014.
- [12] Inkpen, Kori and Upitis, Rena and Klawe, Maria and Lawry, Joan and Anderson, Ann and Ndunda, Mutindi and Sedighian, Kamran and Leroux, Steve and Hsu, David, "" We Have Never-Forgetful Flowers In Our Garden": Girl's Responses To Electronic Games," *Journal of Computers in Mathematics and Science Teaching*, vol. 13, pp. 383–383, 1994.
- [13] Maarten Vansteenkiste and Joke Simons and Willy Lens and Kennon M. Sheldon and Edward L. Deci, "Motivating learning, performance, and persistence: the synergistic effects of intrinsic goal contents and autonomy-supportive contexts," *Journal of personality and social psychology*, vol. 87, no. 2, p. 246, 2004.
- [14] Weintrop, David and Wilensky, Uri, "Between a Block and a Type-face: Designing and Evaluating Hybrid Programming Environments," in *Proceedings of the 2017 Conference on Interaction Design and Children*. ACM, 2017, pp. 183–192.
- [15] D. Weintrop, "Minding the gap between blocks-based and text-based programming," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. ACM, 2015, pp. 720–720.
- [16] Brown, Neil CC and Altadmri, Amjad and Kölling, Michael, "Frame-Based Editing: Combining the Best of Blocks and Text Programming," in *Learning and Teaching in Computing and Engineering (LaTICE), 2016 International Conference on*. IEEE, 2016, pp. 47–53.

- [17] Richard M. Ryan and Edward L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being," *American psychologist*, vol. 55, no. 1, p. 68, 2000.
- [18] Carl Bereiter and Marlene Scardamalia, "Intentional learning as a goal of instruction," *Knowing, learning, and instruction: Essays in honor of Robert Glaser*, pp. 361–392, 1989.
- [19] Martin V. Covington, "Rewards and intrinsic motivation," *Academic motivation of adolescents*, pp. 169–192, 2002.
- [20] Edward L. Deci, "Effects of externally mediated rewards on intrinsic motivation," *Journal of personality and social psychology*, vol. 18, no. 1, p. 105, 1971.
- [21] P. Niemelä, C. Di Flora, M. Helevirta, and V. Isomöttönen, "Educating future coders with a holistic ICT curriculum and new learning solutions," 2016.
- [22] Anne Haarala-Muhonen and Mirja Ruohoniemi and Sari Lindblom-Yläne, "Factors affecting the study pace of first-year law students: In search of study counselling tools," *Studies in Higher Education*, vol. 36, no. 8, pp. 911–922, 2011.
- [23] Haatainen, Simo and Lakanen, Antti-Jussi and Isomöttönen, Ville and Lappalainen, Vesa, "A practice for providing additional support in CS1," in *Learning and Teaching in Computing and Engineering (LaTiCE)*, 2013. IEEE, 2013, pp. 178–183.
- [24] English Department for Education, "National Curriculum in England: Computing programmes of study," 2013.
- [25] J. Piaget and E. Duckworth, "Genetic epistemology," *American Behavioral Scientist*, vol. 13, no. 3, pp. 459–480, 1970.
- [26] Department for Education, "GCSE subject content for computer science," p. 6, 2015. [Online]. Available: [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/397550/GCSE\\_subject\\_content\\_for\\_computer\\_science.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf)
- [27] Deci, Edward L., "8: Ryan, RM (1985). Intrinsic motivation and self-determination in human behavior," *New York and London: Plenum*.
- [28] Bryce Boe and Charlotte Hill and Michelle Len and Greg Dreschler and Phillip Conrad and Diana Franklin, "Hairball: Lint-inspired static analysis of Scratch projects," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. ACM, 2013.
- [29] John Maloney and Mitchel Resnick and Natalie Rusk and Brian Silverman and Evelyn Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, 2010.
- [30] van Krevelen, D., "Intelligent agent modeling as serious game," *Agents for Games and Simulations*, pp. 221–236, 2009.
- [31] D. J. Price, "Networks of scientific papers," *Science (New York, N.Y.)*, vol. 149, no. 3683, pp. 510–515, Jul 30 1965.
- [32] Orni Meerbaum-Salant and Michal Armoni and Mordechai Ben-Ari, "Habits of programming in scratch," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 2011, pp. 168–172.
- [33] Code.org, "Learn to Code Javascript and Blockly."
- [34] Bau, David, "Droplet, a blocks-based editor for text code," *Journal of Computing Sciences in Colleges*, vol. 30, no. 6, pp. 138–144, 2015.
- [35] Bau, David and Gray, Jeff and Kelleher, Caitlin and Sheldon, Josh and Turbak, Franklyn, "Learnable programming: blocks and beyond," *Communications of the ACM*, vol. 60, no. 6, pp. 72–80, 2017.
- [36] Robinson, William, "From Scratch to Patch: Easing the Blocks-Text Transition," in *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. ACM, 2016, pp. 96–99.
- [37] M. Kölling, N. C. Brown, and A. Altadmri, "Frame-based editing: Easing the transition from blocks to text-based programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*. ACM, 2015, pp. 29–38.
- [38] Alimisis, D. and Moro, M. and Menegatti, E., *Educational Robotics in the Makers Era*, ser. Advances in Intelligent Systems and Computing. Springer International Publishing, 2017.
- [39] T. T. Ball, "Physical computing for everyone," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 2017, pp. 3–3.

# Publication IV

Niemelä P., Helevirta M. “K–12 curriculum research: The chicken and the egg of math-aided ICT teaching”, *International Journal of Modern Education and Computer Science*, 2017

DOI: [10.5815/ijmecs.2017.01.01](https://doi.org/10.5815/ijmecs.2017.01.01)

Niemelä and Helevirta (2017) © MECS Publisher



# K-12 Curriculum Research: The Chicken and the Egg of Math-aided ICT Teaching

**Pia S. Niemelä**

Tampere University of Technology/Pervasive Computing, Tampere, Finland  
Email: pia.niemela@tut.fi

**Martti Helevirta**

Tampere, Finland

## I. INTRODUCTION

**Abstract**— In this article, we examine the relationship in K-12 education between Mathematics and Information and Communication Technology (ICT). The topic is reviewed from various angles, based on both a literature study and by directly contrasting the Finnish National Curriculum (FNC) of 2014 (effective since autumn 2016) with the National Curriculums of the UK (UKNC)[3] and the US (USCC)[2].

Finland has chosen a cross-curricular approach to developing the new curriculum for teaching ICT, which involves integrating it mainly with math, but also with handicraft, and various other subjects. This is in direct contrast to the UKNC, for example, which teaches ICT as its own field, to be taught through the Computing and Design/Technology syllabi. This poses a question for this research study, namely, how well do teaching math and ICT fit together? The first step towards answering this question is to establish which ICT concepts and domains are directly supported by math and which are left uncovered. As a theoretical research paper, the rationale for the inter-connectedness of math and ICT is based on the work of many researchers. To illustrate our comparison of the two subjects, in this article we concentrate on clarifying math's and ICT's shared concepts of variable and function.

The results of this study indicate that transfer between the subjects happens bi-directionally, which might suggest that teaching ICT in combination with particular branches of math, notably algebra would be of benefit to our students. In order to pursue this approach, extra modules for logic, basic linear algebra and set theory would also be required. The fundamentals of basic algebra, the function and the variable, and their significance as synthesizers in both algebra and ICT are highlighted. In addition, the use of calculators as function tutors is explored in an instructional classroom setting. The conclusion of this study is that although there are certain benefits to the currently chosen approach of teaching ICT in combination with mathematics, these are not enough to outweigh the advantages of adopting a more versatile dedicated ICT syllabus, such as that provided by the UKNC.

**Index Terms**— ICT curriculum comparison, computing fundamentals, variable, function, math-aided ICT, transfer, computational thinking

Being able to handle a computer and the internet is not only the new norm, but nowadays it is a new necessity. Nearly all social and commercial enterprises now conduct their business on-line, and the face-to-face meetings which our parents went through in order to, for example, pay a utility bill or even take out a bank loan are now rare. Our society expects and needs every independent citizen to be able to function on the internet. ICT has become a basic life-skill, and as such it is essential that we incorporate ICT skills into our education system at an early a stage as is practicably possible.

All of this necessitates that our schoolchildren have to be able to use the net with confidence. In order to be able to follow any kind of further education (FE) our school-leavers have to be able to search for information on the net instead of an encyclopedia. Nearly all students have to consult e-books and e-articles, which can be ordered at low cost, or are even free to read online, unlike in the 'old days' when a student's budget for buying study books often exceeded their budget for food.

Much has been made of the negative social effects of the net, but encouraging our future citizens to utilize the net, and giving them all the skills to do so could actually have significant social benefits, let alone the economic ones. People who can use Instagram or Pinterest to store their favorite pictures can meet like-minded people (albeit virtually), which can encourage them to expand their interests and form bonds with others. The same is true for researchers, i.e. Google Scholar, Mendeley and RefWorks are not only used to search for and store interesting articles for further reading, but they can also lead researchers to establish contact with other researchers, which gives another facet to the concept of networking.

The demand for an expansion of ICT education at all levels of the FNC is clear [1]. The gradual immersion of ICT in all areas of society and the need to facilitate new innovation and productivity require this [7]. We currently have a shortage of skilled manpower to fill the employment needs of our burgeoning ICT industry, so we need to immerse ICT into all areas of society in order bring about a new era of innovation and productivity. Two recently published UK House of Commons' reports, The Digital Skills Crisis [8] and The Digital Skills Gap [18] in fact quantified the cost of the shortage of skilled

ICT personnel by stating that the digital skills gap costs the British economy £63 billion a year in lost GDP.

In the U.S., the act ‘No Child Left Behind’ (NCLB) was continued in 2015 by the renewed congress bill ‘Every Student Succeeds Act’ (ESSA). Its goal is to leverage the affordability and quality of education, especially in STEM subjects. The acts stipulate the necessity of improving access to the wealth of freely available information which the net has given us by equipping schools with up-to-date ICT devices, high-speed connections and with 100,000 highly-trained new STEM teachers. In accordance, the STEM education coalition is advocating STEM education as a national priority. The economic justification for this is summarized in its one-pager [37] that states, for instance, that “60 percent of U.S. employers have difficulties finding qualified workers to fill vacancies in their companies”, and “Jobs in computer systems design and related services – a field dependent on high-level math and problem-solving skills – are projected to grow 45 percent between 2008 and 2018”.

In Finland, the strategy for the development of the information society 2007-2015 emphasizes good ICT-skills for accessing services, for increasing overall productivity and for renewal. However, our industrial base is currently going through a period of upheaval as it adapts to the stagnation of economy caused by the fall in Nokia’s market share, and turmoil in the paper industry. This has led to structural changes in the nature of our economy, and in some cases, to massive layoffs of skilled ICT personnel as well. Because we currently have this pool of unemployed ICT-trained personnel there is certain reluctance on the part of our economic planners to increase the provision of ICT. Rather, the National Digital Agenda of 2011-2020 emphasizes of the societal impact of ICT, such as organizing public services for improving sustainability, transparency and civic participation, where social networking contributes to strengthening the dialogue between the public and private sector. According to this agenda, ICT should be an integral part of education. From the earliest age, well-designed digital services and learning materials (e.g. games and simulations) should be developed and deployed, and distance learning options should be provided for students living in remote areas.

In addition to individual countries setting out their own policies for the promulgation of ICT, several international coalitions and organizations, such as the EU and the OECD, have stressed the importance of ICT education. The EU states that shortages of e-skills in the European workforce will result in an excess demand for ICT practitioners. In their recent visionary reports in 2016, European Committee foresaw the Digital Single Market (DSM) growing to its full potential. This would be based on common ICT standards and initiatives, such as the eGovernment Action Plan, the European Cloud Initiative, Public Private Partnerships and the Fourth Industrial Revolution. This revolution will utilize the potential of integrated cyber-physical systems and technologies in which Finnish research and industry is already leading

the way. We must equip our future workforce with the skills and confidence to operate in the world of big data and the Internet of Things. Although the EU has facilitated the growth of ICT education by funding research with its Horizon 2020 programs, many EU countries lag behind these initiatives and increasing differences in ICT skills are slowing down the development of EU-wide standards and procedures. For example, as the administrator of the EU-wide PISA tests, the OECD is in a unique vantage point for reviewing the education systems of various countries. In this context, it is worth noting that the OECD’s recent report [28] makes it abundantly clear that all students first need to be equipped with basic literacy and numeracy skills in order to be able to participate fully in the hyper-connected, digitized societies of the 21st century.

This article is not only concerned with showing the importance of teaching ICT in schools, but also aims to show the importance of ICT for all members of society. Therefore, we first examine the relevant pedagogical literature aimed at justifying combining ICT with math, as this is the approach currently favored by the Finnish education authorities for the (new) Finnish curriculum. The idea is to teach ICT as a cross-curricular subject, starting with building hands-on assembly kits and electronic experiments in craft subjects at the primary school level. However, the bulk of the ICT syllabus will be integrated with math. Therefore, the initial and primary ICT learning goals, i.e. the generic requirements for algorithmic thinking and the ability to write simple command sequences, are inserted into the math syllabus.

In the following chapters we will go through the ICT and math syllabi in more detail, viewing their potential as a combined syllabus, and as separate ones. In the Results chapter, we will evaluate and compare the two approaches, either math-aided ICT or ICT as a separate subject. This paper’s contribution to the field of curriculum research is its focus on the differences between the reviewed math and ICT syllabi. The paper highlights the expected benefits of each approach to teaching ICT, taking into account not only the knowledge gained from academic research, but also ICT curriculums of other countries. The paper concludes with a summary of our findings and some recommendations for the future.

#### A. Research Question

This study asks:

- How does ICT fit in with the mathematics curriculum?
- What are the fundamental concepts of computer science, and how do they interact with the corresponding concepts of math?
- What ICT topics are left uncovered in the current FNC, when compared with the discrete computing curriculum of the UKNC?

## II. RELATED WORK: PEDAGOGICAL CONSIDERATIONS

From the pedagogical standpoint, combining math and ICT is well justified and can be expected to have a positive effect on the learning of both subjects. The following sections will describe the links between ICT and math in terms of transfer, explicit abstraction, and the areas in which ICT can be used as a scaffold for learning math skills, and vice versa.

### A. Transfer of Learning

Transfer of learning is based on the principle of transferring a skill from one domain to another [36], in our case from math to ICT and vice versa. Learning Transfer explains why learning by analogy is easy. For example, the driver of a car has many of the skills needed to drive a truck, although jumping into the cockpit of an airplane would still be quite a challenge. Successful transfer correlates closely with the current level of acquired expertise: the greater the expertise, the more flexible are one's mental models for adopting new knowledge. An expert finds correspondences and analogies by exploiting previously constructed schemas, identifies without extraneous effort the significant features of new material and hence learns easily in new situations, whereas a novice is stuck with the amount of data and concentrates on irrelevancies. According to venerated model building theories, such as Piaget's genetic epistemology [31], learning is perceived as modifications to the existing schemata; termed as assimilations, if little or no reorganizing is needed, and as accommodations, if the existing schema needs reconstruction. In defining the concept of expertise, Gestalt psychologists (e.g. [21]) refer to the insight experience that helps one find the right solutions intuitively and enables the subject to predict outcomes in new situations.

Transfer may occur either laterally or vertically [17], implying hierarchical learning steps. Transfer can also be near or far [30] within one nearest domain or extended to other further domains. Lateral transfer is more likely to occur and quicker to perform than vertical. Transfer which occurs at only one level is lateral. For example, in math, stepping from addition to subtraction only involves a small cognitive gap, whereas jumping to reordering the equation is a significant step. In pedagogic terms, one level is called a "learning set", and proper and robust learning means progressing consistently from one level to another. Stepping to the next level requires complete mastery of the previous level, in which case, the subsequent vertical transfer can be made without too much difficulty. If sudden vertical jumps are made in learning, however, the variation in learning outcomes among the students grows, and poorer students will suffer.

There are two options for fostering successful learning transfer, and they have been described as the low road and the high road of education. The low road prescribes strengthening routines by iteration, as a result of which responses develop to become more reflexive and automatic. The high road means mindful and explicit abstraction and an active search for connections [30]. In

teaching and learning, this requires that teachers should explicate the underlying principles and point out connections to prior knowledge. As for the learners, they should become more aware of concepts, their relations, and ultimately, they should metacognitively recognize the necessity of making associations in order to enable deeper learning. Nowadays, in Finland at least, explicit abstraction is the accepted approach to mindful, conceptual elaboration which fosters learning transfer.

Transfer has been studied in the context of learning new languages. As a base language, the usefulness of teaching Latin is recognized. In addition to the positive correlation between knowing Latin and learning Romance languages [19][35], the favorable effect also applies to learning other, linguistically unconnected languages. This shows that if learning transfer is successful, a student is capable of finding the common underlying conceptual basis of different topics [17]. Finding such analogies requires a certain level of intellectual maturity at which the student is able to elaborate the material conceptually in order to reach a deeper understanding. In this respect, a positive correlation between ICT and mathematics does appear to exist, so learning transfer is a central theoretical concept of this study.

#### 1) Transfer between ICT and Math

The transfer of learning between languages is analogous to that of math and ICT. As Dijkstra [10] claims, *'Programming is one of the most difficult branches of applied mathematics'*. Syslo and Kwiatkowska [38] argue that discrete mathematics is central in developing algorithmic thinking, which is one of the key skills in ICT, whereas Flatt in the panel discussion [42] states that, in fact, programming is an extension of algebra. It has long been recognized that good math skills are helpful in learning ICT. Conversely, ICT is known to benefit algebraic, logic and problem-solving skills needed in math. The transfer from ICT to math is straightforward. For example, a student trying to master the basic concepts of function and variable in algebra, would be helped if he can practice with an interactive 'shell' or programming environment and writing small programs. On a larger scale, programming means solving problems by dividing them into smaller solvable elements, often implemented as functions, which is similar to problem-solving in math.

**Hello World!** Usually, becoming acquainted with a programming language is begun with this brisk greeting: a programmer calls the simple print function and the computer shows the greeting on the screen. One can still obtain a lot of information from this short first meeting, such as, whether the *main()* function was needed, how parameters were given, how commands were finished, whether indents were needed, how errors were communicated to the coder etc. "Hello World!" is also representative in illustrating the very fundamentals of coding. In most languages, the command drawing the text on the screen, i.e. *print()*, is a built-in function that is called with a text string as a parameter. The parameter is

handled as an anonymous variable instantiated for the duration of the function call, after which it is passed to the function and then destroyed. Even if it is not apparent, passing a string parameter gives the first glimpse of a variable. Meeting the profoundest programming fundamentals, the function and the variable, even in the simplest test program, is an achievement that underlines the importance of these two concepts. Coincidentally, they happen to be the basic concepts of algebra as well. In the section *Variables and functions as common fundamentals*, we analyze their importance in more detail.

**Hello Algebra!** Among the syllabus areas, students struggle most with algebra and functions [23]. Instead of plainly giving an answer to a problem, a student should examine the properties of a function, such as gradients, maxima and minima. Progressing gracefully anticipates a shift of paradigms from the arithmetic to algebraic. In algebra, not only is the concept of a function causing problems, but also getting the variable is difficult. As a surrogate concept, the unknown is used as a bridge in order to learn the concept of the variable. At primary level, the unknown, often represented as a gray box, question mark or an empty space in a calculation, is at secondary level replaced with 'x'. However, the unknown unlike the variable, is understood as a static value, which does not change after it has been figured out. The relation of  $x$  and  $y$ , i.e., a function  $y = f(x)$ , and analyzing its properties both algebraically and graphically, is the new core. With a function "machine" each input of a new  $x$  value will produce one (and only one) output value of  $y$ .

Multiple representations serve as a cognitive aid to learning by providing a means to switch the point of view. Functional thinking can be defined as a type of finding a relation between two varying quantities; hence it has its applications in science, especially physics, the core of which is to depict the laws of nature. Rakes et al [33] have studied the challenges of developing the algebraic thinking, and they have found that especially symbolic notations and abstract reasoning are causing problems, and that students think that they are not properly prepared for making the headway in abstract thinking and generalizations. Rakes recommends conceptual instead of procedural learning for the sake of better transferability that would result in more flexible learning.

Wilkie [44][45][46] discusses the challenges of functional thinking and promotes gradual development by using multiple presentations and bridging arithmetic and procedural knowledge more cautiously. For the primary school, Wilkie has illustrated a pathway of generalizations of sequences as a preparation for algebra; these exercises are labelled pre-algebraic. Development steps include figuring out patterns out of geometric sequences, growing these patterns by defining next items, and visualizing the increase of amounts. In generalizations, it is important to gradually grow recursively step-by-step to defining the  $n^{\text{th}}$  term that determines the relation and general solution. In sequences  $n$  implicitly represents a variable. In addition, Wilkie emphasizes the linguistic means of describing problems

by using pronumerals. By a pronumeral she means the verbose name of a variable in contrast to one letter notation allowed in math. The pronumeral may be called, for instance, *number\_of\_tiles*. This kind of naming complies with ICT coding conventions. In addition to textual representation, also the visual representation provides a beneficial view of the function. The graph gives lots of information for finding the solutions and about the behavior of the function, e.g., the maxima and minima. In addition, the general knowledge of polynomials (continuity, the dominance of the greatest degree) and rational functions (optional discontinuities and asymptotes) helps in depicting the nature of the function.

## 2) Transfer between ICT and other subjects

Deep as the linkage between math and ICT might be, the rigorous mastery of one's mother tongue is a prerequisite for graceful performance in academics overall. Similarly, reading disability predicts disabilities in other subjects and in math, and in the light of this comorbidity, poor performance in ICT is to be expected [47]. In addition to the one's mother tongue, knowledge of the English that constitutes the base for computer languages is an advantage.

The logic of a sentence in verse is to be parsed before understanding algebraic notations in symbols. Philosophers dating back to Aristotle have regarded language as the source of logic and creativity. The task of education would thus be to stimulate all the faculties and nourish young minds. In addition to logic contained in one sentence, constructing a plot or chain of arguments in factual writings should form a logical path. At some point, logic has been taught as part of the optional subject of philosophy - however, logic overlaps also with language and math. As a scaffold of improving logic, a teacher could introduce new tools, such as argument mapping [9], which belongs to the same mapping family together with mind maps and concept maps.

Modeling and abstraction skills are beneficial for 'learning to learn' purposes. In studying academic subjects, concept mapping might become a handy tool. Strengthening conceptual learning is never a waste of time. In general, study skills are worth investing in: knowledge can change and things tend to be easily forgotten, but study skills remain. Metacognitive skills refer to a student's awareness of his means of learning and allow him to plan good strategies for learning, which implies that a student possesses strategies to choose from, such as concept mapping. However, learning to learn should be a cross-curricular goal involving all academic subjects.

In terms of ICT suitability, other STEM subjects besides math are also fit for ICT applications, such as the simulations and videos of science experiments. For example, science demonstrations are sometimes high risk, require expensive ingredients, or happen too quickly to be clearly perceived. With simulations and videos, more cognitive capacity is available to the student to make observations, and an experiment may be repeated as

many times as feasible. STEM has been recently enhanced as STEAM, art included. In the discipline of handicrafts and craftsmanship, learning happens through doing by hand, which is seen as a way of leveraging innovation and creativity. In visual and musical arts, ICT provides fancy tools that facilitate and increase productivity. Theories, such as intelligent hands and learning by doing, are the basis for tactile learning language. In ICT, bridging the connection between electronics and programming may be achieved with the help of different assembly kits (e.g. LEGO MindStorm and Robots, Arduino, LilyPad, littleBits). Electronic components, such as light emitting diodes, buzzers, and couplers can be controlled by executing code commands, and they also give a more concrete and clear response than visible feedback on a computer screen.

### B. The fittest programming paradigm

From the three most prominent paradigms of imperative, functional, and object-oriented, the imperative paradigm is often considered the simplest, based on easy-to-explain concepts and a low level of abstraction. Also, it is easy to visualize with the help of control flow diagrams. Some traditional programming languages, such as Basic, FORTRAN and C, are examples of an imperative paradigm, and command shell scripts employ this paradigm, too. C is by far the most popular imperative language today. However, because it is a low-level language, it allows direct access to HW resources, such as memory and sensors, which forces the programmer to pay a lot of attention to memory management. This raises the pitfalls of handling memory, which is why low-level languages are far from simple for beginners.

Especially in the early history of ICT, imperative paradigm was appropriate, since program instructions were executed in a sequence and the end result was predictable. Functional paradigm and languages were being developed in parallel, however. With the advent of graphical user interfaces (Mac, Windows), the event-driven model started to change mainstream programming. The imperative programming paradigm was enhanced with classes and objects resulting in object-oriented paradigm (OOP) that was especially fit for bigger systems. Then the web took over the world and programming paradigms had to adapt to that. The event-driven model of programming was extended to web applications. Well-defined and often strict programming languages were gradually substituted by the looseness of internet languages, such as HTML, JavaScript and PHP, and finally with all sorts of tag-based extensions (such as JSP and ASP) mixing static and dynamic web content at will. The latest developments are transferring JavaScript-based technologies also to the server (Node.js). Thus, the worst nightmares of programming purists have become true.

The programming paradigm should support the desired style of writing code. While advancing in skills, a student is expected to internalize good coding conventions, such as modularity, documenting, testing, and, in the later stages of studies, also saving HW resources or speeding

up response times. Modularity is achieved by splitting the system into suitable structural components, which can be done at different levels in different programming paradigms. In the OOP, the system is constructed by interfaces and classes, the relations of which may comply with design patterns (such as Visitor, Strategy). In modelling such a system, UML class diagrams are often used. Classes define member variables and methods (functions) and hence encapsulate that class related data.

In regards to transfer between math and ICT, i.e., in bridging algebra and programming, lambda calculus is the missing link [36]. In its conciseness and execution of algebraic operations, such as reductions, lambda calculus conforms to the symbolic expression characteristic of math. It is also categorized as a functional language.

Since it possesses the highest purity and hence a special added value in pedagogy, it appeals to ICT teachers and theoreticians. In addition, being applicable in proving and other theoretical studies in ICT, the lambda calculus and its pure derivatives are willingly used as an introductory university course for functional languages. As the first language in the primary and secondary school, lambda calculus is definitely overkill.

Regarding the functional programming paradigm, the complexity issues have been addressed in educational initiatives targeting at primary and secondary levels. The functional programming camp has tried to satisfy the wishes of ICT educators and provided suitable courses and material: WeScheme, TeachScheme!, Logo Turtles to practice algorithms [16] for example, and DrRacket and also Bootstrap which uses the Racket programming language (prev. PLT Scheme) [23][24][36]. The Bootstrap course targets ultimately at game design. In using Bootstrap, promising results among K-12 students have been reached, Felleisen and Krishnamurthi [13] state boldly that “*Bootstrap provides the strongest evidence yet that teaching functional programming directly affects the mathematics skills and interests of K-12 students*”, and along with them researchers have long regarded programming as a mightifier in learning mathematical concepts (e.g. [35]). Moreover, Schanzer [36] highlights the low threshold of transfer, “*Bootstrap uses algebra as the vehicle for creating images and animations. That means that concepts students encounter in Bootstrap behave exactly the same way that they do in math class. This lets students experiment with algebraic concepts by writing functions.*”

Levy [24] implemented the Racket course for elementary mathematics teachers by adding the consecutive principles of algebra of images and targeting good coding conventions and discipline through using test-first design and documentation. Algebra of images uses images as variables in function calls and prepares for mathematical variables and functions in an entertaining and creative way.

Complexity-wise OOP sets a certain threshold for learning as well. Inheritance, polymorphism and virtual functions, for example, are regarded to belong to the Top-10 most difficult items [27]. Object oriented paradigm is an extension of the type concept found in procedural

programming languages. Furthermore, it is hard to understand methods without first understanding functions. Therefore, OO-paradigm is not appropriate as a first programming paradigm. However, some visual programming tools, for example Scratch may be interpreted as object-oriented, after which the paradigm is not complex at all. Got this way, sprites are objects, whose methods and events are defined, as well as variables set by dragging and dropping.

### C. BYOD (Bring-Your-Own-Device) and forget not to BYOB (Bring-Your-Own-Brains)

Teaching mathematics in Finland is in a transitional stage due to the use of symbolic calculators (allowed since 2012), and shortly also computers, in the matriculation examination. Symbolic calculators bring an excessive competitive advantage to their users. In the spring of 2016, the exam was split in two parts: run with and without a calculator, and the gain of having a symbolic calculator was compensated by making some problems more difficult. In the spring of 2019, the math exam will become electronic as last of the exams. The applications allowed at the work station are the following: LibreOffice for text editing, spreadsheets, vector graphic, GIMP, Pinta, InkScape, Dia, wxMaxima, Texas Instruments N-spire, Casio ClassPad Manager, Geogebra and LoggerPro [49]. Especially scriptable tools are interesting in regard to ICT teaching.

According to the Finnish curriculum 2016, calculators and computers are first to be familiarized with during Y3-Y6. In Y7-Y9, they are applied as learning, assessment and creativity tools. Regarding function keys, the first math concepts are squaring operation  $x^2$  and powers of 10 (e.g. when typing a standard format), since powers and indexes are taught in Year 7. If we consider the squaring operation  $x^2$  as a function,  $x$  is a variable or a function parameter. The user enters the value of  $x$ , after which the calculator prints the value of the function ( $f(x) = x^2$ ) on the screen. Does the student understand that he is using a function? It is unlikely, if this is not especially emphasized, otherwise its meaning is simply reduced as an action button. By pressing it a student gets the desired result, a number squared, as a procedural outcome.

The affordances provided by calculators could be exposed and the function behavior explicated, e.g., the teacher could point out the existing function key  $f(x)$ , and as a concrete example demonstrate the use of the simple function of  $x^2$ , first assign the  $x$  value and retrieve the value of  $y$  as the end result of squaring. As a visual hint, the point  $(x, y)$  could be positioned to the coordinate plane. By inputting sequential integers (1, 2, 3 ...) and plotting points to the plane, the quadratic curve starts to be recognizable. The same exercise may be reused to deduct next numbers of the quadratic sequence. Visualizations of functions with the calculator are beneficial as multiple representations without additional computational overhead. For instance, Desmos as an on-line tool and Mathematica, Maxima and Maple as installable ones are handy in building and exploring functions. However, neither calculators, nor tools nor

games are necessary in teaching mathematics. Actually, adapting to the use of a certain device implies a risk of conceptual welding [35], after which a user is not capable of fully functioning without the device. Expediently, a calculator should remain an optional tool and not a necessity.

Not even learning ICT requires using computers. Many complementing skills necessary in ICT may be practiced well without them, such as computational thinking (CT). Abstraction is one of the three 'a's of computational thinking according to Jeannette Wing [48], who launched the term. The remaining 'a's are automation and analysis. As Wing puts it, "*Computer science is not only computer programming. Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.*" Abstracting systems may be sharpened with mind mapping / concept mapping exercises; in particular, tighter-syntax concept mapping approaches the UML class diagrams used extensively in OOP system design in the industry. Automation, in turn, merges mastering control structures, divide-and-conquer of the problem domain and finding the right algorithms [22].

## III. RESULTS

In order to facilitate comparisons, the ICT syllabi of FNC and UKNC are illustrated as concept maps. As maps, the approaches of math-aided ICT versus ICT as a separate subject are more easily comparable. We first focus on the computational thinking that is common for both syllabi, and after that evaluate the importance of areas that are omitted from math-aided ICT compared to the dedicated ICT syllabus.

### A. Math-aided ICT teaching

Mathematics as a subject is constructed based on spiral progress to more advanced topics. The iterative visits at each math topic at different levels will deepen the knowledge and add details. In merging ICT with math, it is justified to follow the well-established order of math and include corresponding ICT topics where feasible. We took the Finnish math syllabus as the basis, and Figure 1 demonstrates how mathematics as a subject is constructed chronologically and how it expands in a cyclic manner. The concentric gray circles demonstrate different school levels from primary to high school. The further away the subject is located from the center, the later it will be introduced. The figure appears to divide the math syllabus into four major subject areas: arithmetic, geometry, algebra, and the newest addition, computational thinking (CT). CT will lead students to learn how to decompose and solve problems by dividing them into smaller sub-problems, as well as algorithms and modelling.

The Red parts represent topics that are considered especially opportune for ICT teaching: the darker the color, the stronger the emphasis. In addition to CT (algorithms, logic, modelling), the variables and functions of algebra are marked in deep red. Topics marked with lighter red are optional, and proposed as nice-to-have features. For instance, statistics and probability could

lead students towards the fields of data visualizations and data science. In addition to these red-scale areas, the reader should note the UK and US flags. The Flags represent those math additions that are missing from FNC but present in UKNC or USCC, and are anticipated to prepare the students for ICT. Such key areas include sets and matrices. Interestingly, USCC explicitly also defines modeling as one syllabus area that is helpful for both math and ICT.

### 1) Logic, sets and matrices

The initial “hello worlds”, programming evolves into writing control structures such as if-then-else sentences and loops. An if-sentence requires the truth value of its condition to be evaluated, thus presuming skills such as propositional and Boolean logic. The same skill applies in handling iterations when reaching the loop termination condition – typically an equivalence or end of an inequality. These are just the simplest cases where logic in programming is needed. Logic primitives, including truth values as computational entities, seem to be missing in every math syllabus. However, in the UKNC, Boolean logic is currently included in the computing curriculum [3]. Considering the importance of logic to all computer science, this is a distinct lack. The basics of logic, including Boolean logic (true, false, AND, OR, NOT) could very well be included in the math syllabus.

When the amount of data increases, bigger data stores are needed as instead of single variables arrays and other collection types will be introduced. The set is the basic mathematical construct for containment. Sets are highly relevant for programming, as they are the basis for abstract data types called collections and the relational database, among other things. An array may be introduced as a set, a vector or a matrix, and the same operations apply. Therefore, set theory would be useful in any mathematics curriculum designed to support ICT and programming. Currently, set theory is part of UKNC, but is absent from USCC and FNC.

Linear algebra is included in the USCC as matrices and as the syllabus domain of vectors and transformations in UKNC, but matrices and transformations are missing from the FNC. We consider it important for computer science, as matrices enable easy manipulation of data and often simplify computational logic. Matrices are extensively exploited, e.g. for 2D- and 3D-transformations in game development and in data analysis and pattern recognition.

All suggested new math syllabus areas remain at the preliminary level in UKNC and USCC and we propose the same: in logic truth tables and Boolean logic in order to simplify several simultaneous conditions; in sets, Venn diagrams and basic operations of union, intersection and cut with at most three sets; and in matrices, transformations of translation, reflection, rotation and enlargement and finding an inverse matrix. This new math knowledge should be carefully bridged with the prior knowledge with lots of visual exercises and by starting early enough.

Table 1 illustrates in which order these topics, logic, sets and matrices are handled in the UKNC and the USCC.

Table 1. MATH SYLLABI OF UKNC AND USCC

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

### B. ICT as a separate subject

Instead of teaching ICT together with math, it can be taught as a separate subject, as has been shown by the way computing is taught in the UKNC. The computing syllabus was reviewed to discover uncovered topics of the math-aided approach to FNC, see the blue nodes in Figure 2. The Red nodes illustrate overlapping topics found both in UKNC and FNC, where the all-encompassing skill of computational thinking that consists of such sub-domains as algorithms, logic, problem-solving and abstraction is well represented. Algorithms and problem-solving start from the very beginning, whereas abstraction and modelling is from Key Stage 3 upwards in the form of pseudo-coding and flow charts, for instance. Modularity as a good coding practice is highlighted.

In addition to computational thinking, the thread of security and safety starts already from Key Stage 1 and deepens throughout the different stages. In UKNC Computing, safety and security areas culminate as cyber security e.g. identifying possible attack types and system vulnerabilities. The safety and security domain includes the ethics aspect covering a person’s own behavior regarding his own and others’ privacy and covering respect issues as well.

In Key Stages 1 and 2 of the UKNC Computing, the subject content is divided into two parts, first ICT from the perspective of a user and secondly of a programmer. In the user part, the fluent use of technology aims at storing and manipulating digital content. The goal is to understand the digital nature of stored media, text, music, videos. In Key Stage 2, networks are included and their properties, e.g. types and connectivity, are studied. From the perspective of programming, new control structures, sequences and repetition are introduced, as well as such fundamentals as variables and I/O.



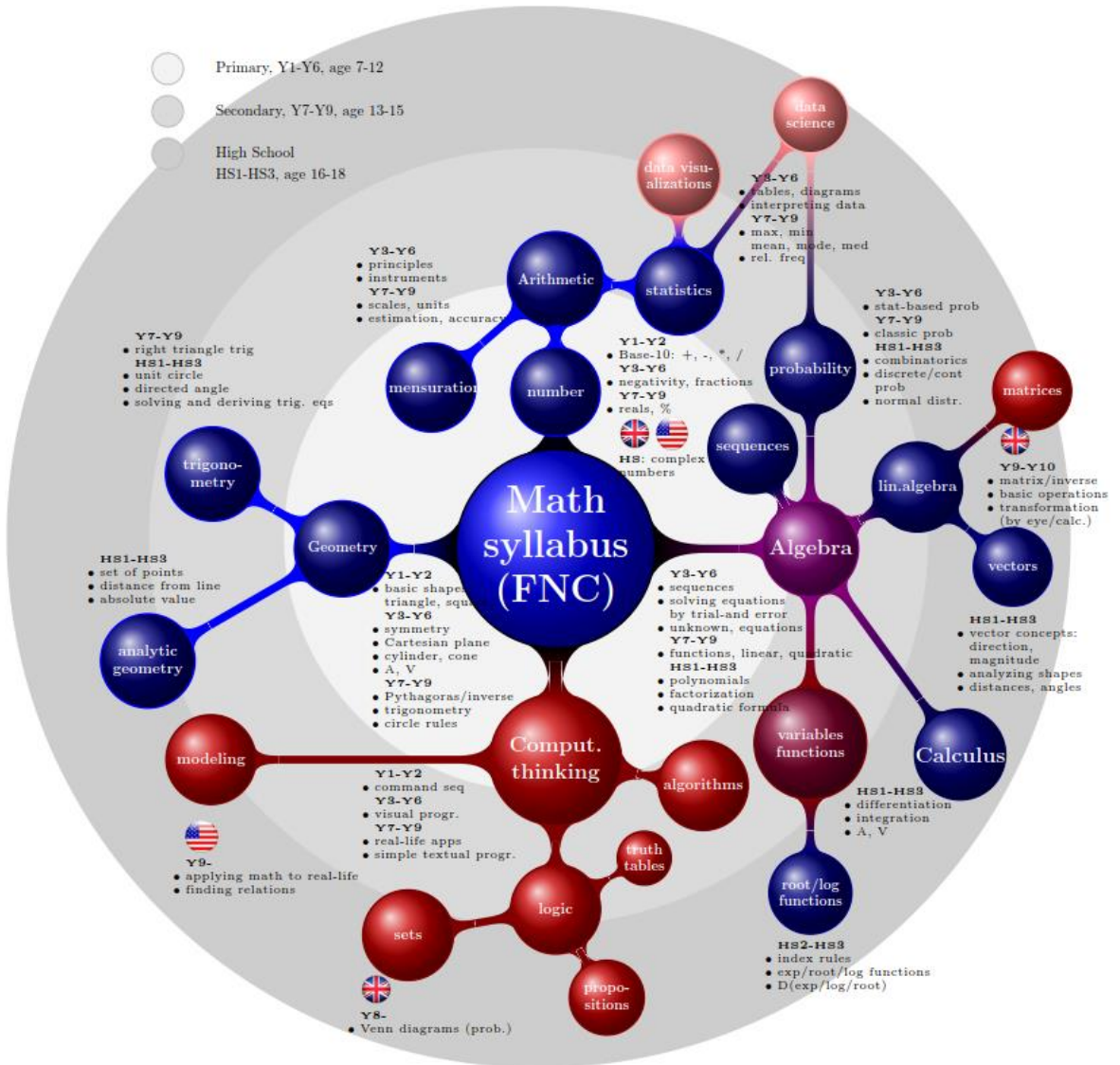


Fig. 1. Combined Math syllabus, based on FNC with additions, computing related items in shades of red,

additions from UKNC marked with  and from USCC with 





From Key Stage 3 onwards, new data structures such as lists, tables and arrays will strengthen the programming skills. Boolean logic (AND, OR, NOT) is brought in and students are familiarized with the binary presentation. Computer systems are reviewed more in depth by introducing HW and SW components and various storages, and the way in which data is stored in memory. In Key Stage 4, the skills are strengthened and analytic thinking and creativity are fostered and applied to one's own interests [4][5]. Students may implement e.g. small applications, portfolios or hypermedia e-books [32] and hence provide material for assessment.

Learning algorithmic and computational thinking is considered as part of mathematics education in FNC as well. On the other hand, issues such as computer and internet architecture, internet of things, robots, big data, cloud computing, artificial intelligence, augmented reality, social media and data privacy and security, are currently omitted. In addition to possible mathematical aspects, these issues involve technical, psychological, societal and legal viewpoints, among others. As Facebook, Twitter, Angry Birds and Pokémon Go phenomena have demonstrated, we live in a world where new ICT inventions can very rapidly take over the whole world - and it would be irresponsible not to give our pupils and students necessary skills to survive in this technological era of wonders. Fluck et al. [14] argue, "*Computer science is rapidly becoming critical for generating new knowledge, and should be taught as a distinct subject or content area, especially in secondary schools*".

### C. Variables and functions as common fundamentals

In this section, we analyze the variable and function concepts in more detail. Variables and functions are the very heart of modern mathematics and science. According to Menger [26], in the development of mathematics and natural science, perhaps the most characteristic concept is that of a variable. Tarski [39] states that the invention of variables constitutes a turning point in the history of mathematics. Kleiner [20] sees the function concept as a distinguishing feature of modern mathematics. In computer science, variables and functions have been an essential part of programming from the very beginning. As computing necessarily involves computer memory, a symbolic reference to a memory location, i.e. variable, is a necessity. Functions were also introduced very early: the second FORTRAN implementation (Fortran II, 1958) already included them. Since then, functions have been part of all major programming languages and have had a major role in various programming paradigms, such as structured programming, procedural programming, and functional programming. Function is the basic means of software decomposition [29], a generally accepted software engineering practice. It directly supports the principles of separation of concerns [11], information hiding, encapsulation and software reuse.

Variables are usually introduced in school mathematics long before functions, e.g. according to the Common Core standard, variables are presented at grade 6 and

functions at grade 8. Table 2 summarizes the differences of variable in school mathematics and in programming. To link variable in math and ICT, Epp [12] advises instructors to draw analogies.

Table 2. VARIABLE IN MATH VS. ICT

	Math	ICT
use case	unknown in equation, a general number, assignment	"bucket"/ memory store, assignment, scope: global/local
alt. use case	function parameter	function parameter passed by value or reference
type of variable	typically numeric (integer, fraction, real number)	numeric or more complex type

Tables 2 and 3 illustrate the interconnectedness of the function and the variable in mathematics and ICT. Especially in ICT, functions and variables are hard to separate from each other - you need both at the very early stage. If functions in mathematics were introduced earlier, together with variables, the mathematical function concept could be used as a starting point for introducing functions in programming languages. Furthermore, the student probably uses some form of calculators, which typically exhibit quite strongly the concept of function in their interface.

Table 3. FUNCTION IN MATH VS. ICT

	Math	ICT
description	relationship between two quantities (usually $x$ and $y$ ) or between elements of two sets	a subroutine that calculates a return value based on input parameters or accomplishes a specific task
number of parameters	typically 1 in elementary math and increasing in advanced math	0 ... n
type of parameters	typically real numbers	numeric or more complex types
number of return values	1	typically 1 (0 ... n depending on language)
return type	typically real number	numeric or more complex type

In analyzing the concepts of variable and function, different meanings were discovered. For example, the model of three uses of variable [41] lists variable as an unknown, a general number, and mutable value of  $x$  in a functional relationship. As unknown, once the value is solved, no reassignment is usually done, so variable is understood as a constant. When the process of generalizing begins, a student starts to transfer from arithmetic towards algebra by identifying patterns [40], e.g. Wilkie [44] uses sequences to facilitate using variables as a pattern generalizer in identifying functional relations. The general number is a midway to actual variables, which are full-blown in functions illustrating the interplay as a relation of the two,  $x$  and  $y$ .

Furthermore, in formulae, the location and naming of the variables define the identity either parametric as a coefficient (constants) or variable as an unknown, for example:

$$f(x, y, z) = a x^2 + b y^3 + c z + d \quad (1)$$

$a$ ,  $b$  and  $c$  are understood as parameters or general numbers, whereas  $x$ ,  $y$  and  $z$  are actual variables.

By the mathematical definition of a function, the ambiguity of output values is not allowed i.e. a function results only one output value per each input. In addition, a domain must include only such input values that produce an output. In programming, the ambiguity of a variable creates confusion. The variable is not its value only, but also comprises a physical location. The address of the variable in a memory is called a pointer in ICT vocabulary. Variable  $x$  is referenced by a pointer  $p$ , see Figure 3, and these two representations are interchangeable by using certain operations. In the following, we use the C-language notations.

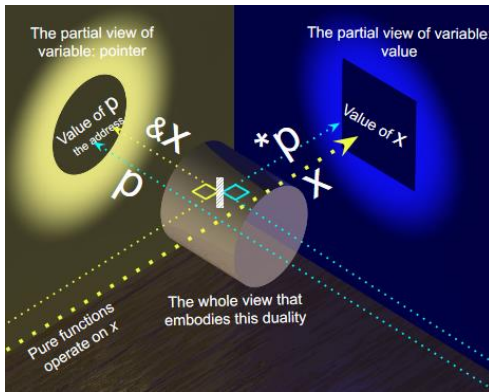


Fig. 3. The duality of variable  $x$  and pointer of  $p$

Note the operators  $\&$ , which is the address-of, and  $*$ , which is the dereferencing operator. Even if you are using variable  $x$ , by adding ‘ $\&$ ’ in front, the address of the variable will be exposed. Analogically, the reference  $p$  may be de-referenced to get hold of the value of  $x$ . Related memory aspect is the data type of the variable influencing the space reserved. For an unsigned integer the space requirement is much less than for a decimal number (*float*, *long*). When declared, variable and pointer are not necessarily assigned a value. In lower level languages, for example in strongly-typed C language, the user is responsible for allocating the memory (*malloc*) of the required data type for the pointer  $p$ , for example:

```
int *p;
p = (int *)malloc(sizeof(int));
```

an operation, which is often forgotten. Both the value and the address may be changed later: a value may be reassigned and the pointer may be redirected to another location. Without special caution, these changes and their side effects may be subtle, go unnoticed and cause nasty and hard-to-trace error situations. Pointers and dynamic memory allocations are among of the hardest ICT topics [27]. In order to fully grasp the concept of variable, unveiling underneath HW structures, i.e., a memory stack, is necessary. Figure 4 demonstrates that in UKNC at the GCSE level, book publishers do not hesitate.

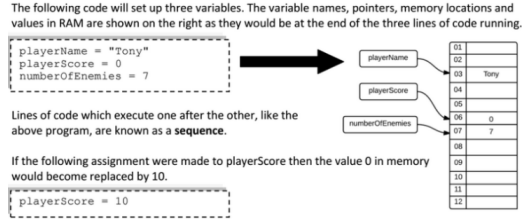


Fig. 5. The GCSE book illustrating variable and its address [15]

Similarly, the concept of function is multifaceted and depends on the used programming paradigm; see the summary of both fundamentals below, in Table 4.

Table 4. VARIABLES AND FUNCTIONS IN SELECTED PROGRAMMING PARADIGMS

	Variable	Function
Imperative	Global and local variables. Pointers exploited in code (in some languages)	Function (returns a value) Procedure (no return value) both may cause side effects
Functional	Variables as constants or unknowns. Once assigned a value is not meant to be changed	Both pure and impure implementations that rely heavily on recursion, sequences and algebraic manipulation
OOP	Member variables encapsulate data inside the object, visibility controlled by access modifiers (private, protected, public)	Object methods that need an instance vs. static methods that need not. Parameters may be passed by value (no changes) or by reference (changes)

Math rules may be violated in all other paradigms but pure functional, which has inspired functional programming advocates to promote its use for teaching algebra as well.

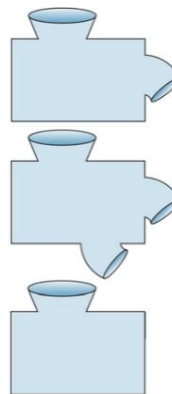


Fig. 4. A pure and an impure function and a procedure

In the adjacent Figure 5, the first function illustrates a valid function in the mathematical sense that takes an input and produces a single output. The next two functions do not follow the rules, e.g. the middle case forks in two different result options based on the inner state of the program. The bottommost case illustrates a procedure: in an imperative paradigm subroutines are split into functions and procedures based on whether they return of value or not.

#### IV. CONCLUSIONS

##### **How does ICT fit in with the mathematics curriculum?**

Programming is heavily based on mathematical concepts. It may be seen as problem solving that requires dividing a problem into smaller solvable sub-problems, modelling a solution and applying algorithmic thinking and logic, as in math. Orientation of problem solving and automation is called computational thinking (CT), which is the most recent addition to the Finnish math syllabus in the 2014 edition. In addition to CT, we recognize algebra, linear algebra and set theory as prerequisites for ICT. From different programming paradigms, functional programming has been found to scaffold learning algebra in particular. Thus using e.g. Racket exercises with image algebra will benefit students, as the move from algebraic function and variable to their computational counterparts complies with the near transfer of learning.

As algebra tutors, calculators and other mobile devices should be exploited to their full potential. Moreover in Finland, transferring into electronic matriculation exams in the spring of 2019 mandates using ICT at earlier school levels. A bunch of ICT tools, such as the computer algebra system wxMaxima, will be available [49]. Practicing with these tools should be started as early as feasible. Math could, for example, be split into normal and laboratory lessons, which would be held in the ICT lab.

##### **What are the fundamental concepts of computer science, and how do they interact with the corresponding concepts of math?**

The basic computing fundamentals are function and variable, which a novice programmer will meet even in the simplest "Hello World" example. In math, algebra and particularly functions are the areas that students find most challenging. Functions should be introduced gradually and by bridging them more closely with prior knowledge. For bridging purposes, Wilkie uses multiple presentations, for example, sequences and visual graphs [44].

By proceeding towards functions and variables in the zone of proximal learning [43], calculators may be used as variable/function tutors by explicitly highlighting the connection between variable  $x$  as an input and function keys as functions producing an output of  $y$ . Probably a student encounters an explicit variable the first time when he is looking at 'x' in a calculator keyboard. To deepen the understanding of variables, one needs information about the memory architecture of a computer. ICT education should contain the basics of computer architecture including data storage and memory. In this context storing variables in memory should be evoked.

Even if syntactic nuances exist, a variable in math is a straightforward concept compared with a variable in ICT, where the dual nature (a value - an address) complicates it. Furthermore, a variable in programming may be of the non-numeric type, such as a string. In mathematical functions, one input value that is typically a real value

results in one and only one output value, also a real value. On the other hand, functions in programming have a wider variety. First, they inherit the ambiguity due to variables as parameters that may be passed by value or by reference, or are of a non-numeric type. Secondly, functions may return a different value with the same input based on the internal state. Thirdly, they may return no value at all.

##### **What ICT topics are left uncovered in the current FNC, when compared with the discrete computing curriculum of the UKNC?**

Compared with UKNC computing syllabus there is a multitude of computer-related skills that are left uncovered without a dedicated subject and teacher. For example, security issues, the basics of computer and network architecture and overall fluency with technology are nowadays comparable with civic competences. In addition, computers may also be used as creativity tools, e.g. design and web-based authoring (blogs, vlogs), and they provide lots of options for developing multi-literacy, for example, content creation, media editing and digital literacy skills. Overall, preparedness for further studies may be strengthened by intelligent searches, source criticism, data analysis and data visualization skills. Explicit knowledge building might be done with mapping tools. Lonka [25] petitions, "*Besides fun and practical activities, it is crucial to facilitate deep learning through guided engagement in scientific inquiry, expert-like designing; in short, students' deliberate efforts to build, create, and synthesize knowledge.*"

Computer science needs skills taught in other subjects as well; mathematics alone is not enough. Still, many areas and new developments in ICT do not fit in to traditional school subjects. It is to be expected that the pace of innovation will continue to speed up in the future – for example, cloud based artificial intelligence is rapidly emerging as a production quality provider of applications of a totally new kind. Since the world is rapidly being digitized, including ICT as a separate subject should be seriously considered. Furthermore, it would also serve as a placeholder for future needs and new developments in technology education.

##### **Future considerations**

Whether ICT should be taught alongside extended math, as a separate subject, or as a combination, it should be studied in practice with various learning experiments. As Lonka [25] points out, "*In Finland and many other countries the availability of technology is adequate, but the primary challenge to overcome is the readiness deficiency for the pedagogically meaningful use of ICT. It is imperative to develop innovative pedagogies that simultaneously support the acquisition of a deep knowledge base, understanding, and 21st Century skills.*"

In addition, different programming paradigms and languages should be compared with novice students in order to find the pedagogically sound and working alternative. For instance, the UKNC curriculum leaves these open and just talks about "low-level" and "high-

level” languages and learning at least two of them [6]. In addition, the short-term hypes of certain programming languages and applications should not influence curriculum planning. Instead, it ought to rest on the fundamentals common to all programming paradigms, whether imperative, functional, or object-oriented.

To lessen the cognitive load in the beginning, visual programming languages such as Scratch, and interactive environments using interpretive languages, should be favored [34]. We advocate progressing from a more disciplined to a looser direction only after orthodox coding conventions, such as modularity, have been internalized. Functional languages are highly disciplined and hence promoted by a few of the ICT establishment. However, the Racket coding school (2016 spring) held in Finland received contradictory feedback from the participating teachers as it was regarded as being too complex [28]. In the UK, the CAS community recommends the path of Scratch-JavaScript-Python, which, for the sake of coding discipline, should preferably be Scratch-Python-Racket. Even if JavaScript were removed to prevent students from developing inappropriate coding conventions, web programming would mandate it. However, if the object-oriented nature of Scratch were recognized and used as a bridge to the most popular object-oriented language to maximize the benefit, then this sequence would stand as Scratch-Python-Java. *If it ain't chickens, it's feathers* - the golden egg of ICT teaching is yet to be determined.

#### REFERENCES

- [1] Finnish National Board of Education. 2014. “Finnish national curriculum 2014”, [Online]. Available: [http://www.oph.fi/download/163777\\_perusopetuksen\\_opetussuunnitelman\\_perusteet\\_2014.pdf](http://www.oph.fi/download/163777_perusopetuksen_opetussuunnitelman_perusteet_2014.pdf) [Accessed: 01- Aug-2016].
- [2] “Mathematics Standards | Common Core State Standards Initiative”, *Corestandards.org*, 2016. [Online]. Available: <http://www.corestandards.org/Math/>. [Accessed: 02- Aug-2016].
- [3] “National Curriculum in England: Secondary Curriculum - Publications - [GOV.UK](http://www.gov.uk)”, *Gov.uk*, 2013. [Online]. Available: <https://www.gov.uk/government/publications/national-curriculum-in-england-secondary-curriculum>. [Accessed: 02- Aug- 2016].
- [4] “Department for Education Computing Programmes of study: Key Stages 1 and 2”, [Online]. Available: [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/239033/PRIMARY\\_national\\_curriculum\\_-\\_Computing.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239033/PRIMARY_national_curriculum_-_Computing.pdf). [Accessed: 02- Aug- 2016].
- [5] “Department for Education Computing Programmes of study: Key Stages 3 and 4”. [Online]. Available: [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/239067/SECONDARY\\_national\\_curriculum\\_-\\_Computing.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/239067/SECONDARY_national_curriculum_-_Computing.pdf). [Accessed: 02- Aug- 2016].
- [6] Department for Education of the United Kingdom. 2014. “Computer science GCSE subject content”, [Online]. Available: [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/397550/GCSE\\_subject\\_content\\_for\\_computer\\_science.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf) [Accessed: 02- Aug- 2016].
- [7] Berger, T., Frey, C. 2016. “Digitalization, Jobs, and Convergence in Europe: strategies for closing the skills gap”, [Online]. Available: [http://www.oxfordmartin.ox.ac.uk/downloads/reports/SCALE\\_Digitalisation\\_Final.pdf](http://www.oxfordmartin.ox.ac.uk/downloads/reports/SCALE_Digitalisation_Final.pdf).
- [8] Blackwood, N. 2016. “Digital skills crisis: second report of Session 2016–17”, House of Commons.
- [9] Billings, D. 2008. “Argument mapping”, *The Journal of continuing education in nursing*. 39(6), 246-247.
- [10] Dijkstra, E.W. 1982. “How do we tell truths that might hurt?”, in *Selected Writings on Computing: A Personal Perspective*. Springer. pp. 129-131.
- [11] Dijkstra, E.W. 1982. “On the role of scientific thought”, in *Selected writings on computing: a personal perspective*. Springer. pp. 60-66.
- [12] Epp, S. 2011. “Variables in mathematics education”, in: *Tools in teaching logic (ed.)*. Springer. pp. 54-61.
- [13] Felleisen, M. & Krishnamurthi, S. 2009. “Viewpoint Why computer science doesn't matter”, *Communications of the ACM* 52, 7, pp. 37-40.
- [14] Fluck, A., Webb, M., Cox, M., Angeli, C., Malyn-Smith, J., Voogt, J. & Zagami, J. 2016. “Arguing for computer science in the school curriculum”, *Educational Technology and Society* 19, 3, pp. 38-46.
- [15] Frankin, J. (ed.). 2015. “OCR GCSE Computer Science 3rd Edition”, 3rd ed. Axsied.
- [16] Futschek, G. 2006. “Algorithmic thinking: the key for understanding computer science”, *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, Springer. pp. 159-168.
- [17] Gagne, R.M. 1965. *The Conditions of Learning*. New York: Holt, Rinehart and Winston. Inc., 1970
- [18] House of Commons. 2016. “Oral evidence: Digital skills gap”, [Online]. Available: <http://data.parliament.uk/writtenevidence/committeeevidence.svc/evidencedocument/science-and-technology-committee/digital-skills/oral/27865.html>
- [19] Jarvis, S., & Pavlenko, A. 2008. *Crosslinguistic influence in language and cognition*. Routledge.
- [20] Kleiner, I. 1989. “Evolution of the function concept: A brief survey”, *The College Mathematics Journal* 20, 4, pp. 282-300.
- [21] Köhler, W. 1970. *Gestalt psychology: An introduction to new concepts in modern psychology*. WW Norton & Company.
- [22] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. 2011. “Computational thinking for youth in practice”. *ACM Inroads* 2, 1, pp. 32-37.
- [23] Lee, R. 2013. “Teaching Algebra through Functional Programming: An Analysis of the Bootstrap Curriculum”.
- [24] Levy, D. 2013. “Racket Fun-fictional Programming to Elementary Mathematic Teachers”.
- [25] Lonka, K. & Cho, V. (ed.). 2015. Report for EU Parliament 2015: Innovative Schools: Teaching & Learning in the Digital Era: Workshop Documentation.



- [26] Menger, K. 1954. "On variables in mathematics and in natural science", *The British Journal for the Philosophy of Science* 5, 18, pp. 134-142.
- [27] Milne, I. & Rowe, G. 2002. "Difficulties in learning and teaching programming—views of students and tutors", *Education and Information technologies* 7, 1, pp. 55-66.
- [28] Organisation for Economic Co-operation and Development. 2016. "Skills for a Digital World".
- [29] Parnas, D.L. 1972. "On the criteria to be used in decomposing systems into modules", *Communications of the ACM* 15, 12, pp. 1053-1058.
- [30] Perkins, D. & Salomon, G. 1988. "Teaching for transfer", *Educational leadership* 46, 1, pp. 22-32.
- [31] Piaget, J. & Duckworth, E. 1970. *Genetic epistemology*. American Behavioral Scientist 13, 3, pp. 459-480.
- [32] Portugal, C. 2014. "Hypermedia E-book as a Pedagogical Tool in a Graduation Course", *International Journal of Modern Education and Computer Science*, Vol. 6(9), pp. 8.
- [33] Rakes, C.R., Valentine, J.C., McGatha, M.B. & Ronau, R.N. 2010. "Methods of Instructional Improvement in Algebra A Systematic Review and Meta-Analysis", *Review of Educational Research* 80, 3, pp. 372-400.
- [34] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. & Silverman, B. 2009. "Scratch: programming for all", *Communications of the ACM* 52, 11, pp. 60-67.
- [35] Rich, P.J., Leatham, K.R. & Wright, G.A. 2013. "Convergent cognition", *Instructional Science* 41, 2, pp. 431-453.
- [36] Schanzer, E.T. 2015. "Algebraic Functions, Computer Programming, and the Challenge of Transfer".
- [37] STEM education coalition One-Pager. 2016. "STEM Education, Good Jobs, and U.S. Competitiveness", [Online]. Available: <http://www.stemedcoalition.org/wp-content/uploads/2016/01/STEM-Factsheet-Updated2.pdf>.
- [38] Syslo, M.M. & Kwiatkowska, A.B. 2006. "Contribution of informatics education to mathematics education in schools", *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, Springer, pp. 209-219.
- [39] Tarski, A. 1994. *Introduction to Logic and to the Methodology of the Deductive Sciences*. Oxford university press.
- [40] Usiskin, Z. 1988. "Conceptions of school algebra and uses of variables", *The ideas of algebra, K-12* 8, pp. 19.
- [41] Ursini, S. & Trigueros, M. 2001. "A model for the uses of variable in elementary algebra", *PME CONFERENCE*, pp. 4-327.
- [42] Van Roy, P., Armstrong, J., Flatt, M. & Magnusson, B. (2003). "The Role of Language Paradigms in Teaching Programming", *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ACM, New York, NY, USA, pp. 269-270.
- [43] Vygotskij, L.S. 1978. *Mind in society the development of higher psychological processes*. Cambridge, Harvard University Press.
- [44] Wilkie, K.J. & Clarke, D.M. 2016. "Developing students' functional thinking in algebra through different visualizations of a growing pattern's structure", *Mathematics Education Research Journal* 28, 2, pp. 223-243.
- [45] Wilkie, K.J. 2016. "Learning to teach upper primary school algebra: changes to teachers' mathematical knowledge for teaching functional thinking", *Mathematics Education Research Journal* 28, 2, pp. 245-275.
- [46] Wilkie, K.J. 2016. "Students' use of variables and multiple representations in generalizing functional relationships prior to secondary school", *Educational Studies in Mathematics* pp. 1-29.
- [47] Willcutt, E.G., Petrill, S.A., Wu, S., Boada, R., Defries, J.C., Olson, R.K. & Pennington, B.F. 2013. "Comorbidity between reading disability and math disability: concurrent psychopathology, functional impairment, and neuropsychological functioning", *Journal of learning disabilities* 46, 6, pp. 500-516.
- [48] Wing, J.M. 2006. *Computational thinking*. *Communications of the ACM* 49, 3, pp. 33-35
- [49] Ylioppilastutkintolautakunta, "SÄHKÖINEN YLIOPIILASTUTKINTO – MATEMATIIKKA", [Online]. Available: [https://www.ylioppilastutkinto.fi/images/sivuston\\_tiedostot/Sahkoinen\\_tutkinto/fi\\_sahkoinen\\_matematiikka.pdf](https://www.ylioppilastutkinto.fi/images/sivuston_tiedostot/Sahkoinen_tutkinto/fi_sahkoinen_matematiikka.pdf)

#### Authors' Profiles



**Pia Niemelä** works currently as a project researcher in the Finnish Academia funded project "Social media supporting Vocational Growth", being a doctorate student in Pervasive Computing Dept. Previously, she participated Helsinki University research projects, Systemic Learning Solutions (SYSTECH) and (TUTLI), which developed educational learning solutions and commercialized them. Her background is in the industrial software development, the longest with Nokia, e.g. as a Java spec lead of Sensor API, JSR-256, and ServiceConnection API, JSR-279, which specifies RESTful web services for the mobile edition. She graduated from the Helsinki University of Technology in 1995, from the department of technical physics, completed pedagogical studies in Tampere University in 2015 and has worked as a STEM teacher both in Finland and Cambodia.



**Martti Helevirta** received his M.Sc. degree in Software Engineering from Tampere University of Technology, Finland in 1986. He has an extensive career of over 25 years in ICT industry as a software developer in the private sector and as a systems analyst/project manager in the public sector.

Manuscript received October the 1<sup>st</sup>, 2016, revised November the 9<sup>th</sup>, 2016; accepted December the 7<sup>th</sup>, 2016.



# Publication V

Partanen T., Niemelä P., Mannila L. and Poranen T. “Educating Computer Science Educators Online: A Racket MOOC for Elementary Math Teachers of Finland”, *Proceedings of the 9th International Conference of Computer Supported Education, 2017*

DOI: [10.5220/0006257800470058](https://doi.org/10.5220/0006257800470058)

Partanen et al. (2017)



# Educating Computer Science Educators Online

## *A Racket MOOC for Elementary Math Teachers of Finland*

Tiina Partanen<sup>1</sup>, Pia Niemelä<sup>2</sup>, Linda Mannila<sup>3</sup> and Timo Poranen<sup>4</sup>

<sup>1</sup>*Tampere City, Tampere, Finland*

<sup>2</sup>*Pervasive Computing, Tampere University of Technology, Tampere, Finland*

<sup>3</sup>*Åbo Academi University, Turku, Finland*

<sup>4</sup>*Computer Sciences, University of Tampere, Kalevantie 4, 33100 Tampere, Finland*  
*tiina.s.partanen@tampere.fi, pia.niemela@utu.fi, linda.mannila@abo.fi, timo.t.poranen@uta.fi*

**Keywords:** Computer Science Education, K-12 Education, Teacher Training, MOOC, Racket, Teacher Professional Development (TPD), Math-integrated Computer Science

**Abstract:** Many countries all over the world are in the process of introducing programming into their K-12 curricula. New Finnish Curriculum includes programming mentioned especially in accordance with mathematics and crafts. Consequently, Finland needs to train teachers to teach programming at elementary school level. In this paper, we describe how elementary math teachers were educated online to teach programming using the Racket programming language. The aim of the course was to increase both content knowledge (CK) and technological pedagogical content knowledge (TPACK). By analyzing the course feedback, questionnaires and exercise data, we present the teachers' views on the course and effects on their professional development (TPD). Finally, we describe development ideas for future online courses.

## 1 INTRODUCTION

Our society is becoming increasingly digitalized, which has also given rise to a global discussion on the role of computer science in education. As a consequence, a number of countries all over the world have introduced computational thinking, programming or computer science in their K-9 curricula. Since 2014, for instance students in England have learned to compute starting at the age of five. In Finland, programming has been part of the national curriculum effective since autumn 2016. It was introduced as a cross-curricular addition, but integrated in particular into the syllabi of crafts (grades 3-9) and mathematics (grades 1-9).

Integrating programming into the basic education was a remarkable change, to which Finnish teacher training departments have not yet fully adapted. Henceforth, both pre- and in-service teachers need to learn to program and obtain an understanding of the core elements of computational thinking. Adding curriculum requirements of this kind retrospectively changes the job description of a teacher significantly. The employer is responsible for taking care of the teachers' training and providing time for sufficient professional development. In addition to new require-

ments, rapid technological disruptions – especially within information and communication technology (ICT) – necessitate the continuous professional development of teachers in order to ensure frictionless career moves in future. By choosing courses that enable them to fulfill curriculum requirements, thus enhancing employability, teachers aim at maximizing their market value. Hence, they are willing to put their own effort into studying.

Although the government recognizes this training need, in-service training resources are still insufficient. Against this background, all voluntary training initiatives are warmly welcome. In this paper, we present the Racket track of Koodiaapinen MOOC, a project initiated informally by a group of volunteer teachers to respond to the gap in formal training. After the voluntary start, the Ministry of Education is currently sponsoring the MOOC by offering the organizers funding according to the number of in-service teachers completing the course. The goals of the course are two-fold: to educate math teachers to learn programming in the first instance, and secondly, to function as a tool in the search for best practices to teach programming.

## 1.1 Theoretical background

Teachers now find themselves in a situation where they need to upgrade their skills and knowledge related to technology, programming and digital competence. This can be seen as a type of transformation, although, it does not fully match 'transformative learning' as defined by Mezirow (1997). As an initiator, Mezirow depicts a 'disorienting dilemma', but the way in which he describes the process can be seen as too intimidating: during disorientation, fear, anger and shame are listed as the driving forces. Consequently, we chose to speak about the 'reorienting dilemma' of teachers instead. In the current reorientation, the most dominant motivation is the external pressure caused by changes in the curriculum and the consequent demands to educate students accordingly. Emotionally, reorientation is also less engaging than disorientation.

Fortunately in Finland, teachers commonly exhibit several types of internal motivation, e.g., their own personal willingness to develop. Teachers consciously build and develop their technological knowledge and expertise as agents of their professional development. In order to attain a better view on motivational factors, we refer to the self-reinforcement and self-efficacy theories of Bandura (2006), where self-efficacy is an important predictor in successful professional development, even more than the actual achievements. On a global scale, the self-efficacy of Finnish teachers is considered high and boosted by excellent PISA results, which teachers strive to maintain. In addition, they are aware of the new standards set by the education authorities as a response to the rapid technological development.

The change in perceived self-efficacy is one metric for assessing the MOOC course learning outcomes. Kennedy (2016) talks about enactment problems in bringing new programming skills into the classroom context after attending a professional development course. She highlights the gap between the course setup and the actual teaching context of the real classroom. Good self-efficacy in math is anticipated to lower this threshold and foster the transfer. In this study, we wish to focus in particular on teaching math and programming together, and examine how math teachers adapt to the change.

## 1.2 Research Questions

- What has been learned about organizing a programming MOOC for teachers?
- How did the teachers evaluate the Racket course?

- How did the teachers describe the effect of the course on their professional development and self-efficacy in teaching programming?

## 2 RELATED WORK

### 2.1 Digital competence in the Finnish curriculum

In December 2014, a new curriculum for Finnish basic education (grades 1-9) was accepted by the Finnish National Board of Education. This curriculum has been in effect since August 2016 and emphasizes digital competence as an interdisciplinary skill throughout all grades. The curriculum excerpts below mention programming explicitly in the objectives of two subjects, mathematics and crafts:

#### **Grades 1-2**

Digital competence: *"Students get and share experiences about digital media and programming in an age-appropriate manner."*

Mathematics: *"Students get acquainted with the programming basics by creating step-by-step instructions, which are also tested."*

#### **Grades 3-6**

Digital competence: *"Students learn to program and become aware of how technology depends on decisions made by humans."*

Mathematics: *"Students plan and implement programs using a visual programming language."*

Crafts: *"Students practice programming robots and/or automation."*

#### **Grades 7-9**

Digital competence: *"Programming is practised as part of various other subjects."*

Mathematics: *"Students should develop their algorithmic thinking and learn to solve problems using math and programming. In programming, students should practise good coding conventions."*

Crafts: *"Students use embedded systems, plan, and apply programming skills in order to create products."*

As the curriculum stipulates that programming is to be taught integrated with math, we start by examining how best to exploit the expected synergy benefits. Compared with programming, math has a well-established syllabus that has evolved into its current state since the very dawn of the educational system. Despite certain minor syllabus areas being dropped from, or reintroduced to, the curriculum, the core content of the math syllabus has remained much the same for decades. In order to ensure smooth transition, the

strong math core should be exploited in order to introduce the analogous and logically progressive steps for programming. It is tentatively assumed that integrating programming into math will move the center of gravity of the syllabus towards computational thinking.

Computational thinking has gained traction since the seminal article by Wing (2006) on the topic. There is no absolute consensus on the definition of the term computational thinking, but many start from Wing's (2011) observation, "[t]he 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." Several operational definitions have been suggested, for instance one presenting a set of cornerstones of computational thinking including data collection, analysis and representation, problem decomposition, abstraction, algorithms, automation, parallel code and simulation (Barr and Stephenson, 2011). Papert (1996) has stated, "*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*".

Math is at the very core of programming that requires algebraic, logic and problem solving skills. Synergy implies mutual benefit between two entities, and although the benefits that a good understanding about math and perceived self-efficacy confer on the learning of computational skills are clear (Lent et al., 1991; Zeldin and Pajares, 2000), the transfer in the other direction, from programming to math, may not be that obvious. In a successful transfer, however, a student should be capable of finding the common underlying conceptual bases of different topics (Jarvis and Pavlenko, 2008). Finding such analogies requires a certain level of intellectual maturity and that a student has elaborated on the learning material conceptually in order to reach a deeper understanding.

In general, successful transfer correlates with already acquired expertise: the greater the expertise, the more well-rounded one is skill-wise and the more flexible one's mental models are for adopting new knowledge (Bransford et al., 2000). An expert finds correspondences and analogies by exploiting the previously constructed knowledge. The expert can easily and without extraneous effort identify the significant features of the new material and is hence able to easily learn in new situations. A novice, on the other hand, can become bogged down by the amount of data and may concentrate on irrelevancies. In defining the concept of expertise, the Gestalt psychologists

(e.g. Köhler, 1970) refer to the insight experience that helps learners find the right solutions intuitively and enables them to predict the outcomes in new situations.

Transfer may happen either laterally or vertically (Gagné, 1965), near or far or by the low road or the high road (Perkins and Salomon, 1988) implying a certain hierarchy of learning. In addition, Rich et al. (2013) state that one of the two complementary subjects tends to be interpreted in learners' minds in a more abstract manner while the other encourages to focus on application. In the case of math and programming, math is more abstract, while programming is understood as applied math (Dijkstra, 1982). In math, educators have long talked about conceptual and procedural knowledge (Gray and Tall, 1994): conceptual knowledge comprises a full possession of the appropriate concepts and the ability to link them together, i.e., the high road to knowledge transfer, while procedural knowledge consists of well-internalized mathematical routines on the low-road. Practicing math routines is anticipated to provide one appropriate affordance for programming interventions.

Transfer between math and programming will be streamlined by bridging the current math syllabus with corresponding programming topics. In addition to students, we note the value of transfer to in-service teachers: the similarity between math and programming of the Racket MOOC is expected to motivate math teachers to learn programming.

## 2.2 Examples of K-12 Computer Science elsewhere

To get a better grasp of the current situation of programming or computer science education EU-wide, European Schoolnet carried out a review of the state of computer science education in 2015 (Heintz et al., 2016). The majority of European countries (17 out of 21) had already introduced or were in the process of introducing computer science concepts in their K-12 curriculum (Balanskat and Engelhart, 2014). Some countries, such as the UK, introduced computer science as a separate subject (English Department for Education, 2013), while others decided to integrate it with other subjects, for instance, Finland (Finnish National Board of Education, 2014). The length of the syllabi varies from K-9 to K-12, and a few countries only include computer science in the upper grades (10-12). However, integrating computer science with math seems risky. For instance, an OECD report has suggested that the higher the degree of computer usage in math lessons, the poorer are the results (OECD,

2015). Thus the need for developing and evaluating a suitable pedagogy for the integration is palpable.

In determining the role of computer science in education, there are various metaphors used, e.g. computer science as literacy, a maker mind-set, or grounded math (Burke and Burke, 2016). If the literacy metaphor is used, then programming as digital literacy emphasizes the same logical skills as are applied in constructing linguistically correct sentences, that is, using e.g. and/or/not in order to get the internal logic of the sentence expressed. From a 'maker mind-set' perspective, the programming language should be as productive as possible, with a low learning curve, which suggests visual programming languages, such as Scratch. Some studies have, however, questioned the benefits of Scratch in enhancing problem solving skills and good programming practices (Gülbahar and Kalelioglu, 2014; Meerbaum-Salant et al., 2011). The grounded math approach highlights the links between programming and math: the transfer between math and programming seems closest to the functional programming paradigm. For example, learning functions in algebra can be practised using functional programming languages.

Combining functional programming with math is not new. Historically, attempts range from the early use of LOGO (Futschek, 2006; Kulik, 1994) to recent experiments employing Racket and Haskell (Alegre and Moreno, 2015). While results from the LOGO initiatives varied (Kulik, 1994), Racket evaluations have consistently been positive and stable (Felleisen et al., 2014; Felleisen and Krishnamurthi, 2009; Schanzer et al., 2015; Schanzer, 2015). The amount of research and the positive results reported convinced our course organizers to choose Racket for the teacher training MOOC.

### 2.3 Teaching Programming Using Racket

The Racket programming language (<http://racket-lang.org>) is a multi-paradigm language, which also supports functional programming. Being a Scheme dialect previously known as PLT Scheme, it has been developed further as an open source project (Flatt and Fandler, 2012). Racket includes a programming IDE, DrRacket, designed especially for teaching purposes (Felleisen and Krishnamurthi, 2009). In contexts where DrRacket cannot be installed, a web-based environment called WeScheme (Yoo et al., 2011) can be used. WeScheme also enables online sharing and remixing of programs.

DrRacket has built-in support for the so-called stu-

dent languages starting with Beginning Student and ending up with Advanced Student Language. Each of these Student Languages gradually introduces new programming primitives and concepts. Simplified syntax and semantics help beginners grasp the core concepts of function design, such as composition and calling. Tool creators have also defined more precise error messages in order to assist novices in debugging and analyzing code (Marceau et al., 2011).

DrRacket comes with graphics and animation libraries (2htdp/image, 2htdp/universe) that are especially apt for beginner level programming. These libraries were developed for more than a decade in the Program by Design project (<http://www.programbydesign.org/>). Along with these libraries, the guide book "How to Design Programs" was written by Felleisen et al. (2014) for high school and college level programming courses. The book emphasizes the advantages of functional programming and introduces Design Recipe to systematize problem solving by dividing it into a chain of smaller decisions. The Recipe also instructs how to construct a program by composing functions and encourages writing tests before an actual function implementation (Felleisen et al., 2014).

To preserve the purity of the functional paradigm, the imperative features of Racket are pushed back. For instance, an assignment operation (set!) and other functions causing side effects (display, read) are not introduced until the student reaches Advanced Student Language level. In the most recent version of "How to Design Programs", these imperative features were removed altogether (Felleisen et al., 2014).

The Program by Design project provides a separate program for middle school called Bootstrap. Its mission is to introduce computer science by teaching algebra by programming a video game using Racket. This algebraic approach has been proved to improve understanding about math concepts, such as variables and functions (Wright et al., 2013). Racket also enables passing numbers, strings and images as parameters. Using images in calculations justifies the description of Racket as "arithmetic with images" (Felleisen and Krishnamurthi, 2009).

A number of articles promote DrRacket as a prominent way of learning algebra (Lee et al., 2011; Schanzer, 2015), especially when special care is taken of the valid instructions and purposefully planned exercises and pedagogical models, such as the Cycle of Evaluation (Schanzer, 2015). The use of design recipes turned out to foster the right order of operations and composition of nested functions. Felleisen and Krishnamurthi (2009) boldly suggests that Bootstrap (functional programming) provides the strongest

evidence of the favorable effects of programming on math skills, along with the fact that researchers have long viewed programming as a promising domain where to practise math concepts (Papert, 1996; Resnick et al., 2009). Bootstrap arranges professional training workshops for middle school math teachers in the USA. In addition, Racket was utilized in the professional training of math teachers in Israel (Levy, 2013). This training was based on the principles of Program by Design, emphasizing test-first development and the featured “algebra of images”.

### 3 Method

The idea for Koodiaapinen MOOC was introduced in 2015 by Tarmo Toikkanen and Tero Toivanen during the annual Interactive Technology in Education conference in Hämeenlinna, Finland. The initial idea was to help teachers learn programming with material that has been prepared especially for them by their peers, for instance, more experienced teachers.

Design based research aims at linking theory and practice in the discipline of education (Reimann, 2011). It stipulates the use of several iterations and re-designs of an educational artifact based on feedback and experience. The beta version of the course was developed and executed without funding, and four voluntary MOOC administration members worked in their spare time. According to the principles of DBR, the course and its content would then be improved course-by-course based on the feedback received.

Figure 1 illustrates the process of two nested design cycles: the outer cycle is the process of curriculum planning that takes place once a decade, while the inner one is the iterative process of developing the ‘Coding at School’ Racket material (<http://racket.koodiaapinen.fi>). Development proceeds in cycles, where different stakeholders give feedback.

Based on the customers, in-service teachers in the present study, the artifact is redesigned together with researchers, whose research interests lie in integrating computational thinking with math education.

First three tracks of the Koodiaapinen course (ScratchJr, Scratch, Racket) targeted at a number of general goals: promoting creativity; presenting programming as a tool for creating something new and inspiring; sharing pedagogical ideas and artifacts during the course; using exercises directly applicable in a classroom context in order to make it easier for teachers to get started; offering course participants sufficient content knowledge so that they would not limit themselves to applying ready-made programming materials but also be able to create their own

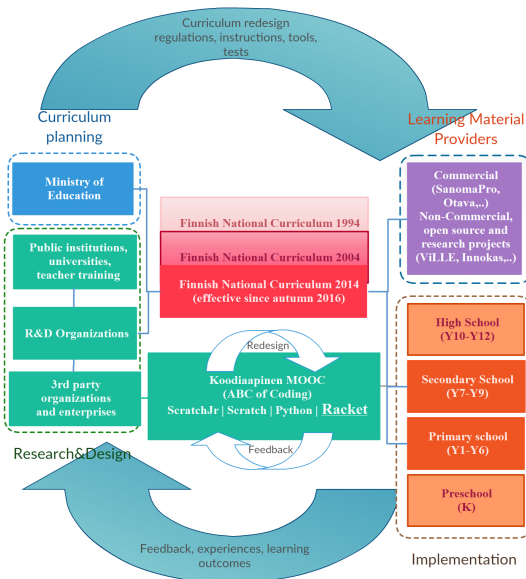


Figure 1: Nested DBR cycles of curriculum updates (update/10yrs) and Coding at School courses (2 updates/yr)

programming exercises; and enabling peer-support by urging participants to help each other on discussion forums. The use of these peer-support channels was crucial, given the lack of resources.

The very core of the ‘Coding at School’ Racket material is to reveal the nature of programming as a sort of applied mathematics and show how mathematics can be taught through programming. The approach is designed to motivate math teachers to adopt programming in their teaching, and to show that programming lessons are not time wasted.

After the course, its potential effects on the participating teachers’ content knowledge (CK) and technological pedagogical content knowledge (TPACK) were evaluated (Voogt et al., 2013). TPACK measures the efficiency of teachers in exploiting technology in their teaching, and this evaluation required suitable rubrics. However, the fluent use of technology during math lessons is not the core goal of the MOOC. Instead, the study aims at building on the existing math foundation and fully exploiting and transferring this knowledge as programming skills in order to create the positive feelings of self-efficacy from the very beginning. Consequently, in this study, the TPACK model has been exploited in an attempt to fill the newly-created space between math and computer science, by focusing in particular on a smooth transfer between these two disciplines.

### 3.1 MOOC Platform Selection

Eliademy, the free Finnish platform, was selected for the autumn 2015 MOOC (<https://eliademy.com/catalog/koodiaapinen.html>). Eliademy comprises such basic features as course editing and management tools, a discussion forum, assignment systems for returning files and support for quizzes. At that time, the platform did not include peer-review. In addition, sharing artifacts and ideas was not functional in Eliademy, and as a result the course was transferred to Padlet (<http://padlet.com>), an on-line notice board system instead.

While Padlet worked nicely for sharing images and code via WeScheme links and essays via Google Drive or OneDrive, an integrated grading system was missing. This lack caused manual work for the instructor. In addition, Padlet did not allow the instructor to contact participants, which prevented her from giving personal feedback on, for instance, their programming style and essays. Hence, an integrated learning environment would have been preferable.

For the spring 2016 MOOC (<https://plus.cs.hut.fi/aapinen-racket/K2016/>), the course platform was switched to A+ (<https://plus.cs.hut.fi/>) developed by Aalto University and used in the university’s own programming courses. In the beginning, A+ did not support showcasing of returned artifacts. As this was found crucial for the Koodiaapinen MOOC, the Rubyric team added the feature.

After this change, Rubyric’s peer-review functionality was used to minimize the workload of the course personnel. The new system enabled the instructor to define grading rubrics and points, so the peer-review was as easy as selecting an appropriate description for the code quality among given options. Peer-reviews were conducted anonymously without using the Padlet-style review wall. Code to be reviewed was allocated randomly to reviewers. Exercises that were not peer-reviewed were put on the Padlet-style wall with the participants’ names so that peers could comment on their work, as shown in Figure 2. Piazza (<http://piazza.com>) was used as the discussion platform. These services were integrated using IMS-LTI protocol.

### 3.2 Course Design Principles

The implementation of the Racket track was inspired by the Systematic Program Design online course offered by edx.org (Kiczales, 2015). Similarly to that course, the Racket MOOC contained weekly exercises with the following introductory material:

1. Short motivational video, in which the lecturer in-

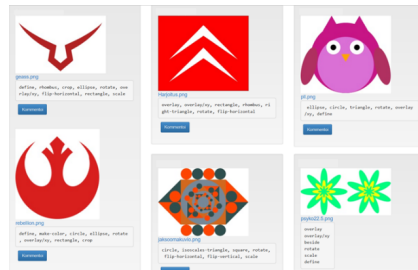


Figure 2: Topic 1 artifacts on the Padlet-style wall (Spring-2016)

troduced the contents and the purpose of the exercise. Some videos also responded to feedback received during the previous week.

2. Tutorial screen capture videos introduced the core concepts. The lecturer used DrRacket for showing programming examples that demonstrated the concepts to be learned during that week. The stepper tool was used extensively in order to explain the evaluation rules. Some written notes were added online, but the course content was mainly delivered in video format. The idea was that the course participants could test the programming examples themselves while watching the videos.
3. The Design Recipe was used to demonstrate the principles of function design, see Figure 3. By using the recipe, a user can solve one detail at a time and proceed step-by-step until the function is ready. One of its noteworthy features is the definition of test cases before implementing the actual function body.

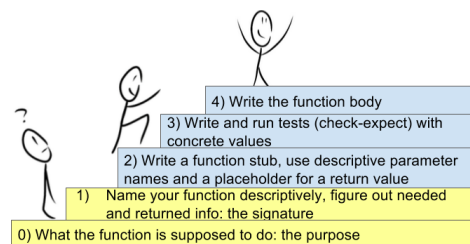


Figure 3: Design recipe presented as a staircase that helps to design a function step-by-step

4. Exercises and their solutions were delivered as both DrRacket and WeScheme files and used as self-tests of the course content presented in the video tutorials.
5. Hands-on exercises differed from the Systematic Program Design exercises as neither peer-review nor multiple choice quizzes were used to check how well the material had been understood.

Lastly, Koodiaapinen had an essay about the pedagogical aspects instead of a programming project as in Systematic Program Design.

The programming exercises and their solutions were taken from the Coding at School material and the Coder's handbook (<http://racket.koodiaapinen.fi/manuaali/>), which contains documentation for the graphics and animation libraries (2htdp/image and 2htdp/universe), Beginning Student Language primitives, and new library additions of Racket Turtle and display-read.

## 4 Results

The first Racket track was carried out on a weekly basis. At the end of each week, feedback was collected first using Google Forms and later Grader, an online survey tool developed at Aalto University. The feedback was saved and analyzed in order to improve the next course. Open-ended textual feedback for theory and exercises was solicited, as well as a time estimate about the workload of a week. In this chapter, we introduce our results in chronological order, first the Autumn-2015 results and corresponding lessons learned, followed by the Spring-2016 results.

### 4.1 Autumn-2015

Promisingly, up to 369 teachers attended the beta version of the Racket track of Koodiaapinen MOOC. The Racket track turned out to be significantly more difficult in comparison with the other tracks (ScratchJr, Scratch), thus preventing participants from maintaining the same pace. Based on the feedback, the course had too much weekly content: the target was 2 h/week, but the actual workload was notably higher, 3-4 h/week. As a result, the MOOC team decided to slow down the Racket track. To complete the course, 80% of the coursework had to be returned, as 140 out of the 369 participants did (completion rate 38%).

The autumn course proceeded in the order of functions-logic-loops. All in all, too many concepts were introduced simultaneously and teachers started to struggle with learning, which in turn resulted in an excessive amount of questions in the discussion area. On the other hand, experienced programmers still lacked a few crucial tools needed in the exercises, e.g., a conditional structure. **Lesson learned:** Topics need to be organized based on their difficulty: simple things first and then proceeding to more advanced techniques in a widening spiral. Exercises must be synced with the introduced topics.

Coupling a function with the design recipe caused confusion: participants did not see the need for test cases and stubs. **Lesson learned:** These topics need to be introduced separately, first functions and manual testing in an interactive window. After this, tests may be automated with check-expect and re-used in designing new functions. Automatic tests will help in understanding how functions should be implemented and in checking that functions behave as expected. A similar order is also used in the guide 'How to Design Programs' (Felleisen et al., 2014).

No major problems due to DrRacket and WeScheme were reported. However, check-expect supported images in DrRacket but not in WeScheme, thus examples worked differently, which left WeScheme users puzzled. In addition, some interoperability issues arose due to a few functions introduced in Racket-lang documents, yet the Finnish Coder's handbook was restricted only to primitives functional in both.

Due to time constraints, some important concepts were left out from the autumn course, e.g., recursion, local variables and more advanced usage of lists. However, these skills were needed when implementing the quiz application in a good programming style without repetition. **Lesson learned:** The quiz was found highly motivating and applicable for school, however, the corresponding lesson is to be complemented with the needed advanced topics.

The final essay worked as expected: teachers found it both motivating and useful. Postponing pedagogical and curriculum considerations to the end of the course was a deliberate design decision: one needs to understand relevant programming and computational thinking ideas as well as challenges involved in teaching before adjusting the curriculum. The essay aimed at highlighting TPACK issues and summarizing the ideas evolved during the course. The main TPACK threads of this MOOC were to ponder how to apply the course exercises to STEM subjects, especially math, and foster creativity, culminating with the final essay. In addition, the accomplished self-designed artifacts were one step towards advancing self-efficacy and enactment.

The course material for Spring-2016 was revised and rearranged and new material was developed based on the lessons learned from Autumn-2015. The style of the beta version was retained: introductory and tutorial videos, PowerPoint slides, exercises with solutions and the programming artifacts to be returned and reviewed. Three returned artifacts were peer-reviewed, and therefore they had fixed return and review deadlines. For all other artifacts, the deadline was the end of the course.

Table 1: Two iterative Racket track development cycles based on the feedback (Autumn-2015/Spring-2016)

w	Autumn-2015	Lessons Learned	Spring-2016
1	<p>Introduction to Racket programming using images (2htdp/image library), problem decomposition, variables as global constants. <b>Artifact:</b> An image shared by participants</p>	<p>The image created positive feelings of achievement: using simple geometric shapes familiarized the teachers with the tool and enabled creativity.</p>	<p>t1: The same exercise as in autumn</p>
2	<p>Using functions and parameters to solve problems (abstraction), the design recipe as a scaffold. <b>Artifact:</b> Definition of a function (screen capture images). The 1st exercise focused on purpose of the function, its signature, a stub and test cases, i.e. on demonstrating the design recipe process. The 2nd exercise was to implement the actual function body and the minimum of two function calls.</p>	<p>Contents from weeks 2-4 from autumn 2015 were divided into topics 2-5 and new content on recursion and broader usage of lists was added.</p>	<p>t2: Earlier introduction: how to use true/false, comparison operators, predicates and conditional structure (if) to control code execution, how to test functions in an interaction window and by writing unit tests (check-expect) <b>Artifact:</b> Definition of a function, which uses if-expression, including the purpose, signature and test cases for all code branches. Peer-reviewed by 3 participants</p>
3-4*)	<p>Boolean operators (and, or, not), comparisons, predicates, and conditional structures (if, else) to control code execution. The animation engine (2htdp/universe), reading a user input (display-read library) familiarizing with WeScheme. <b>Artifact:</b> WeScheme code with conditional structures, the result could be an animation, a simple quiz or an automated calculator for some math formulas. Shared with the group</p> <p>*) Time for the autumn material of week 3 was doubled (week 3 became weeks 3 - 4)</p>	<p>More code skeletons provided for t3-5, so the course participants did not need to create applications from scratch.</p>	<p>t3: The design recipe for functions. Writing tests first, Boolean operators and conditional structure for more complex logic, animation engine (2htdp/universe), WeScheme to share code <b>Artifact:</b> No changes to the animation and the simple mouse app. The quiz and the calculator postponed. t4: Helper and recursive functions, reading user input (display-read library), blocks with side-effects (user interaction), local variables for storing the input <b>Artifact:</b> Defining multiple functions (at least one recursive) i.e. a purpose, a signature and test cases. The end result could be an image, recursive calculation or a simple calculator that asks an input in a loop. Peer-reviews by 3 course participants</p>
5	<p>Looping using higher order functions (map, foldl, foldr) and lists, usage of Racket Turtle library to draw geometric shapes. <b>Artifact:</b> Shared image, which uses a looping structure and either:</p> <ol style="list-style-type: none"> <li>higher order functions + 2htdp/image</li> <li>higher order functions/loops with repeat + Racket Turtle</li> </ol>	<p>As similar exercises were already done in accordance with the recursion, only Racket Turtle option was maintained and foldl/foldr were left out.</p>	<p>t5: Lists to store a set of values, iterating a list recursively and producing new lists or one result value, how to use image files in DrRacket and WeScheme applications <b>Artifact:</b> WeScheme code, which implements a simple list based quiz using a recursive list-eater function, shared with others. t6: Looping using lists and higher order functions (map), usage of Racket Turtle library to draw geometric shapes <b>Artifact:</b> Shared image, drawn using Racket Turtle library</p>
6	<p>Requirements of the Finnish curriculum for the programming, algorithmic thinking/computational thinking, and how to teach and integrate it with other subjects. <b>Artifact:</b> Either a.) an essay (1-2 pages) reflecting the challenges of teaching programming b.) design of a new exercise c.) a syllabus for integrating programming into one's own subject</p>	<p>Participants felt that this exercise was particularly applicable for their work and hence found it motivating.</p>	<p>t7: The same exercise as in autumn, except peer-reviews were added</p>



Some participants complained that it was difficult to create programs from scratch and preferred exercises with given code skeletons. Thus, such skeletons were provided as a scaffold for writing a program in order to support a learning path with distinct use-modify-create steps (Lee et al., 2011).

## 4.2 Spring-2016

The course syllabus for Spring-2016 was designed so that different aspects of algorithmic thinking (abstraction, logic, repetition) were introduced side by side starting from the easier ideas and progressing to more advanced ideas. The course content was divided into seven topics, each scheduled to take 10-14 days. Three topics were almost identical to those in Autumn-2015: topic 1, topic 6 (previously 5) and topic 7 (previously 6), i.e., the final essay was left unchanged. Table 1 illustrates an overview of the course content and exercises, and how the course developed according to the feedback.

The Spring-2016 version of the Racket track had fewer participants (171) than the beta version, as it was competing for the same target group with a newly introduced Python track. Of these 171 participants who started the Racket track, 100 finished, resulting in a 58% completion rate (80% of the coursework was required to pass). The completion rate was 31%, taking into account all teachers (325) who had enrolled on the MOOC. The number of teachers, whose returned coursework was accepted for topics t1 - t7, is illustrated in Figure 4.

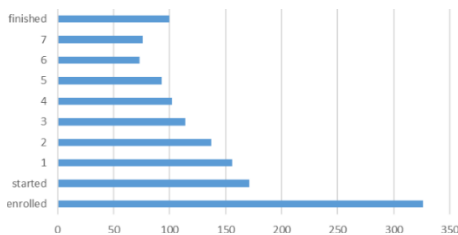


Figure 4: Number of accepted coursework for topics 1-7

### 4.2.1 Pre-course survey

We conducted a pre-course survey to get background information about the participants (N=137) using Grader, which was also used for lesson feedback. Based on the survey, most participants had some previous experience in programming: only 26% had none, and as many as 45% had used more than one programming language/environment. In the order of popularity, the languages mentioned were Scratch,

34%, C/C++ 30 %, Java 26 %, Pascal 22 %, Basic 20 %, Python 15 %, Visual Basic 14 %, JavaScript 10 %, FORTRAN 9 %, LOGO 8 % and C# 3 %. The greatest number of participants were among the 25-to-35 age group (42%) and the majority of them were female (78%). Almost 90% of the course participants were math teachers and a similar proportion (91%) taught in grades 7-9. Almost two thirds (61%) reported that they had never used programming in their teaching.

Compared to Autumn-2015, notably fewer programming questions were asked on the discussion forum. Consequently, the peer support that proved so important during Autumn-2015, was almost non-existent during Spring-2016. The same phenomenon was noted in all four tracks of Koodiaapinen. One possible reason is that the Piazza was too difficult to use, another might be that the joint discussion area of all tracks was laborious to follow and hence distancing. In addition, discussions generated email notifications to all participants, which was found annoying. Moreover, while Autumn-2015 was advertised to everyone, Spring-2016 was marketed mainly to math teachers, who are anticipated to be more fluent with technology by default, thus asking less questions.

### 4.2.2 Course feedback

The teachers' feedback on their level of experienced enthusiasm, suitability and usefulness of the seven topics covered was above average on a scale of 1-5 (1: not at all, 2: a bit, 3: reasonably, 4: a lot, 5: very much). The highest enthusiasm was created by programming images (t1,6) and animations (t3). The final essay (t7) scored the highest on the suitability and usefulness due to its pedagogical and curriculum reflections, whereas recursion (t4) scored the lowest. Overall, however, the scores did not differ remarkably, see Figure 5:

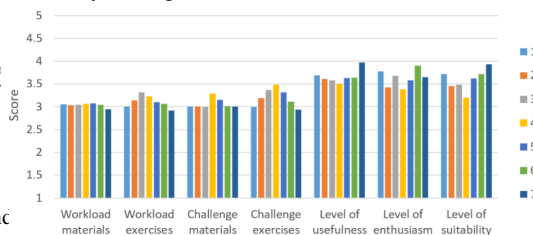


Figure 5: Spring-2016 feedback for topics 1-7

The course feedback indicated a medium difficulty level for most lessons, but recursion was considered the most difficult in all aspects. In similar vein, the workload of most topics scored in the middle, where the exercises using more complex logic and

the animation library resulted in the highest workload scores. The actual hours used per topic are shown in Figure 6. The target for Spring-2016 was 3-4 hours of work per topic, and in fact most participants used 2-6 hours. Hence, the target was reasonably close to the realization.

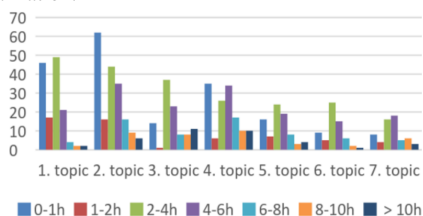


Figure 6: Amount of participants as a function of workload grouped by topics

#### 4.2.3 Post-course surveys for the course development

At the end of the course, the course setup was evaluated by the participants, but the survey gain was notably low at that iteration: only 12 participants answered, out of which 11 completed the course. The teachers were asked, for example, to list aspects that helped in completing the course, the top three reasons being:

1. the tutorial videos of the course
2. the importance of the subject
3. concrete programming exercises

Table 2 and 3 illustrate the claims that the participants agreed on either 'strongly' or 'to a certain extent'. The rejected claims were '*The course did not support my development in becoming a teacher of programming*' (1.75) and '*The course did not offer sufficient knowledge of teaching programming*' (2.25). Most improvement ideas related to the course schedule and the difficulty of the exercises:

1. Only peer-reviewed exercises had deadlines while the rest had to be completed before the course end. This made it possible to complete tasks in the wrong order, causing difficulties. Setting pedagogically adjusted deadlines would improve this.
2. For some topics, the video examples were simpler than the real exercises. This can be remedied either by having the material cover more complex examples, or making the exercises easier to match the difficulty level of the videos.
3. Although the video tutorials were considered helpful and clear, a few teachers would preferred written material: after watching a video, finding specific information caused problems.

Table 2: Claims that participants agreed on

Feature	Score [1..5]
MOOC-style courses are well suited for professional development	4.5
The course provided skills needed for teaching programming	4.4
The course increased my knowledge on how to teach programming	4.3
The course provided methods for teaching programming	4.3
The course worked well for as a MOOC	4.3
The course gave me concrete ideas (tips) for my work as a teacher	4.3
The teaching methods applied enhanced my learning	4.1
The course increased my confidence in programming as a teacher	3.9
I was committed to learning actively by myself during this course	3.9
I will promote the contents that I learned during this course to the other teachers in my school district and my own school	3.6
The course made me excited about programming	3.6
The course increased my interest in learning more about teaching programming	3.6
I received sufficient support during the course	3.6

4. To complete the course, 6 out of 7 topics were required, thus a few participants did not return the final essay. It, however, was considered the most important topic, in particular more important than those covered in topics 5 and 6. Consequently, the teachers suggested that the final topic should be compulsory and either 5 or 6 elective.

The course material and exercises were spread on multiple platforms, such as A+, Eliademy, Rubyruc and Piazza, which was found confusing. Moreover, A+ and Eliademy required separate accounts, which led into problems e.g. when opening solution files in Eliademy. In order to find the exercises more easily, the teachers suggested direct links to be attached to the material. Due to the variety of platforms, following the course execution was also problematic. The status of a delivery was shown in several places, thus getting an overview of each assignment was cumbersome, which hampered the recognition of pending peer-reviews. Only a sufficient number of peer-reviews granted a credit and because of pending reviews a number of credits were missing. Credits were delayed also because the course set-up required

the instructor to accept each return separately. Yet another source of annoyance was Piazza by sending participants an excessive amount of email notifications. Consequently, the teachers proposed a daily or weekly digest instead.

These improvement ideas were taken into account in the later versions of the Racket course; the development of the course is meant to be continuous. After implementing a few of these improvements, multiple benefits could already be listed regarding the new platform and course syllabus. First, reviewing and grading of returned artifacts was much easier using the new Padlet-style wall. Also peer-reviewing decreased the amount of work, since the instructor needed to manually review only the cases that were unclear. Secondly, code reviews provided a new learning opportunity and clarified the requirements of good programming style, for instance, why appropriate naming and written purpose statements for functions are important and why code needs to be tested. Thirdly, the new course syllabus and schedule seemed to work better and the workload for the course participants and the instructor was more balanced.

## 5 CONCLUSIONS

We have developed an online programming course for elementary school teachers, emphasizing the linkage between mathematics and programming, and facilitating creativity and sharing. As the first result, we found that teachers were willing to learn programming and appreciated the pedagogical considerations in particular: the final exercise of writing the essay scored highest of all exercises on both suitability and usefulness. The previous programming exercises aimed at enhancing the content knowledge. As such, the programming exercises were tailored to be fit for teaching in authentic classroom settings, but in conjuncture with learning to program teachers were called to reflect on the exercises and come up with new aspects and brand new tasks as well.

Secondly, the teachers' feedback from the Spring-2016 course iteration was more positive than from the first beta trial, which indicated that the level of difficulty and workload were becoming reasonable. The contents of the course were perceived both suitable and useful. In addition, the course seemed to create a fair amount of enthusiasm, making this type of programming MOOC a motivating and interesting form of professional development for in-service teachers. In the effort to provide effective in-service training, the improvement of the learning platform and fine-tuning the course material should be contin-

uous. Consequently, the course will be incrementally improved based on the participants' feedback: these two subsequent Racket courses prove that this type of agile course development is feasible.

Thirdly, the positive course feedback and reflections in essays seem to suggest that professional development and self-efficacy of the participants increased. However, future research should observe the long-term effects of the course, e.g., how many participants actually started using the learned material and skills in their work. As Kennedy (2016) points out, real enactment in the school context is the final test.

Further studies should also examine more thoroughly the suitability of the material for elementary math and the question whether the course gave a satisfactory enough insight into computational thinking. For the purpose, the final essays provide a plethora of data to review. Systematic research and executing various learning experiments will enable determining the best practices for developing computational thinking and enhancing math syllabus, thus fulfilling the new requirements of the Finnish Curriculum 2014.

## 6 ACKNOWLEDGMENTS

We thank the Aalto University A+ and Rubyric teams for their efforts for the Koodiaapinen MOOC. We express our gratitude to Emmanuel Schanzer for modifying WeScheme to suit our material and to Technology Industries of Finland Centennial Foundation for funding the development of the Koodiaapinen MOOC in Spring 2016. Last but not least, thanks to Tarmo Toikkanen for coordination.

## REFERENCES

- Alegre, F. and Moreno, J. (2015). Haskell in Middle and High School Mathematics. In TFPIE vol. 1,.
- Balanskat, A. and Engelhart, K. (2014). Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe. European Schoolnet.
- Bandura, A. (2006). Guide for constructing self-efficacy scales. Self-efficacy beliefs of adolescents 5.
- Barr, V. and Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? ACM Inroads 2, 48–54.
- Bransford, J. D., Brown, A. L. and Cocking, R. R. (2000). How people learn.
- Burke, Q. and Burke, Q. (2016). Mind the metaphor: charting the rhetoric about introductory programming in K-12 schools. On the Horizon 24, 210–220.

- Dijkstra, E. W. (1982). How do we tell truths that might hurt? In *Selected Writings on Computing: A Personal Perspective* pp. 129–131. Springer.
- English Department for Education (2013). *National Curriculum in England Computing programmes of study*.
- Felleisen, M., Findler, R., Flatt, M. and Krishnamurthi, S. (2014). *How to Design Programs, Second Edition*. MIT-Press.
- Felleisen, M. and Krishnamurthi, S. (2009). Viewpoint Why computer science doesn't matter. *Communications of the ACM* 52, 37–40.
- Finnish National Board of Education (2014). *Finnish National Curriculum 2014*.
- Flatt, M. and Findler, R. (2012). *PLT - The Racket guide 1*.
- Futschek, G. (2006). Algorithmic thinking: the key for understanding computer science. In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* pp. 159–168, Springer.
- Gagné, R. M. (1965). *The Conditions of Learning*. New York: Holt, Rinehart and Winston.
- Gray, E. M. and Tall, D. O. (1994). Duality, ambiguity, and flexibility: A proceptual view of simple arithmetic. *Journal for research in Mathematics Education* , 116–140.
- Gülbahar, Y. and Kalelioglu, F. (2014). The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education-An International Journal* 13.1, 33–50.
- Heintz, F., Mannila, L. and Färnqvist, T. (2016). A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education. *Frontiers in Education* October.
- Jarvis, S. and Pavlenko, A. (2008). *Crosslinguistic influence in language and cognition*. Routledge.
- Kennedy, M. (2016). How does professional development improve teaching? Review of Educational Research .
- Kiczales, G. (2015). *UBCx: SPD1x Systematic Program Design - Part 1 (version 1, summer 2015)*.
- Kulik, J. A. (1994). *Meta-analytic studies of findings on computer-based instruction vol. 1, of Technology assessment in education and training* pp. 9–34. : Psychology Press.
- Köhler, W. (1970). *Gestalt psychology: An introduction to new concepts in modern psychology*. WW Norton & Company.
- Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. and Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads* 2, 32–37.
- Lent, R. W., Lopez, F. G. and Bieschke, K. J. (1991). Mathematics self-efficacy: Sources and relation to science-based career choice. *Journal of counseling psychology* 38, 424.
- Levy, D. (2013). *Racket Fun-fictional Programming to Elementary Mathematics Teachers*. In *TFPIE2013 TFPIE2013*.
- Marceau, G., Fisler, K. and Krishnamurthi, S. (2011). Measuring the effectiveness of error messages designed for novice programmers. In *Proceedings of the 42nd ACM technical symposium on Computer science education* pp. 499–504, ACM.
- Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (2011). Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* pp. 168–172, ACM.
- Mezirow, J. (1997). *Transformative learning: Theory to practice*. New directions for adult and continuing education 1997, 5–12.
- OECD (2015). *Students, Computers and Learning*.
- Papert, S. (1996). An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning* 1, 95–123.
- Perkins, D. N. and Salomon, G. (1988). Teaching for transfer. *Educational leadership* 46, 22–32.
- Reimann, P. (2011). Design-based research pp. 37–50. *Methodological choice and design*. : Springer.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. and Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM* , 52, 60–67.
- Rich, P. J., Leatham, K. R. and Wright, G. A. (2013). Convergent cognition. *Instructional Science* , 41, 431–453.
- Schanzer, E., Fisler, K., Krishnamurthi, S. and Felleisen, M. (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*, pp. 616–621, ACM.
- Schanzer, E. T. (2015). *Algebraic Functions, Computer Programming, and the Challenge of Transfer* .
- Voogt, J., Fisser, P., Roblin, N. P., Tondeur, J. and van Braak, J. (2013). Technological pedagogical content knowledge—a review of the literature. *Journal of Computer Assisted Learning* 29, 109–121.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM* 49, 33–35.
- Wing, J. M. (2011). Computational thinking. In *VL/HCC* p. 3, csta.acm.org.
- Wright, G., Rich, P. and Lee, R. (2013). The influence of teaching programming on learning mathematics. In *Society for Information Technology & Teacher Education International Conference* vol. 2013, pp. 4612–4615, editlib.org.
- Yoo, D., Schanzer, E., Krishnamurthi, S. and Fisler, K. (2011). WeScheme: the browser is your programming environment. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* pp. 163–167, ACM.
- Zeldin, A. L. and Pajares, F. (2000). Against the odds: Self-efficacy beliefs of women in mathematical, scientific, and technological careers. *American Educational Research Journal* 37, 215–246.



# Publication VI

Niemelä P., Partanen T., Harsu M., Leppänen L., and Ihantola P. “Computational thinking as an emergent learning trajectory of mathematics”, *Proceedings of Koli Calling International Conference on Computing Education Research, 2017*

DOI: [10.1145/3141880.3141885](https://doi.org/10.1145/3141880.3141885)

Niemelä et al. (2017a)

# 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>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

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

© 2017 Association for Computing Machinery.

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" [40]. 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 models and ideas introduced by Wing [39] and Cuny, Snyder and Wing [10] encompass all the essential components of CT and nothing superfluous, and they still have enough resolution power to categorize the Racket MOOC participants' views. The combined model 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-orientation 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 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 interactionism [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 interactionism 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,

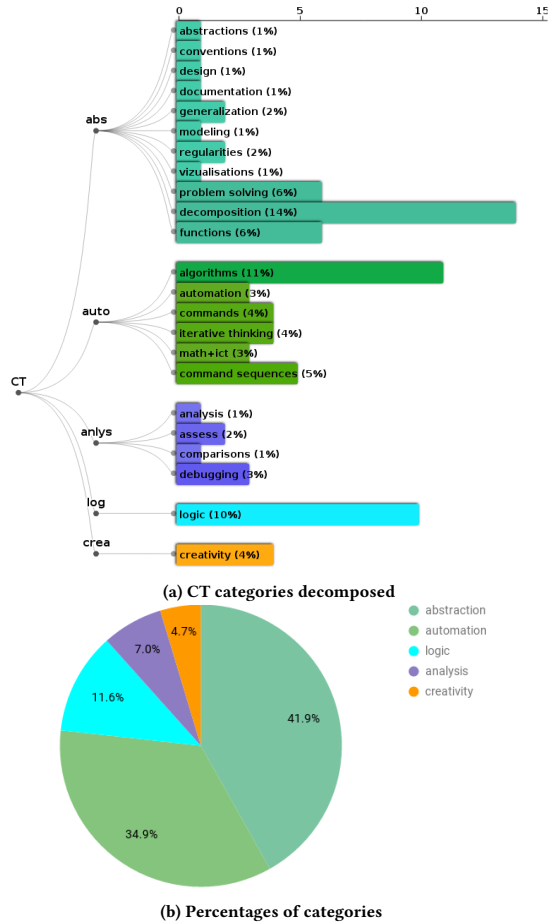


Figure 1: CT key areas by math teachers as a decomposed dendrogram (a), and a pie chart (b). Percentages illustrate the relative frequencies of the concepts in the essays; however, values less than 1% are omitted.

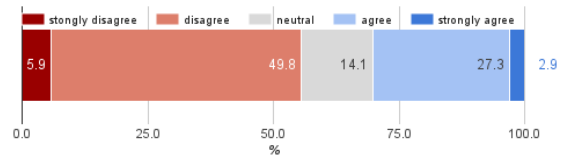


Figure 2: Computing should be taught by concentrating more on theory, concepts and design than creative hands-on experiments (N=206)

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 sub-problems 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.** In Wing's words [39], computing is automating abstractions, i.e., an implementation. Within the automation category, the teachers regarded algorithms as the most important 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-orientation, 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.

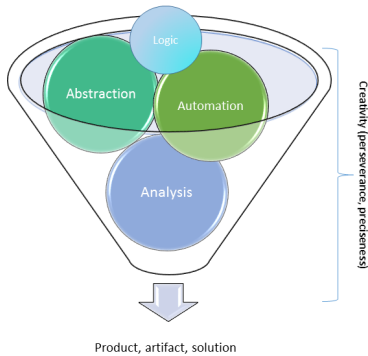


Figure 3: CT model enhanced with logic and creativity

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 enthruse 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*.

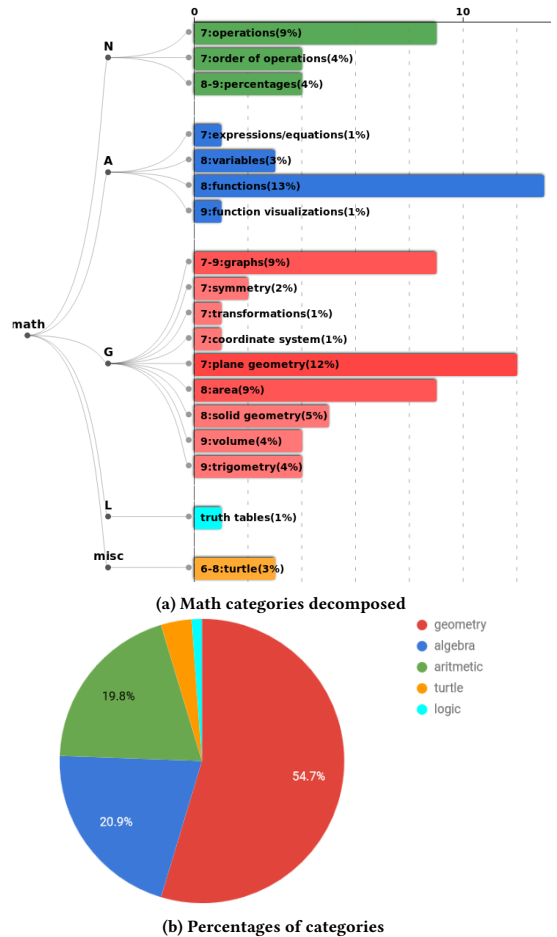


Figure 4: Syllabus areas fit for computing (N=206). The percentages are based on the relative frequency of exercise proposals of the teachers. Percentages illustrate the relative frequencies of the math syllabus areas connected to the exercises. Values less than 1 % were omitted.

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 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 categorization 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 LTs into the area of CT, thus implying the timing if the corresponding concepts were 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 focus 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 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 compared with its simplicity in math, being an entity of at least

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
Y6-9	C	Turtle, creative exercises related mainly to geometry, computer graphics, animations

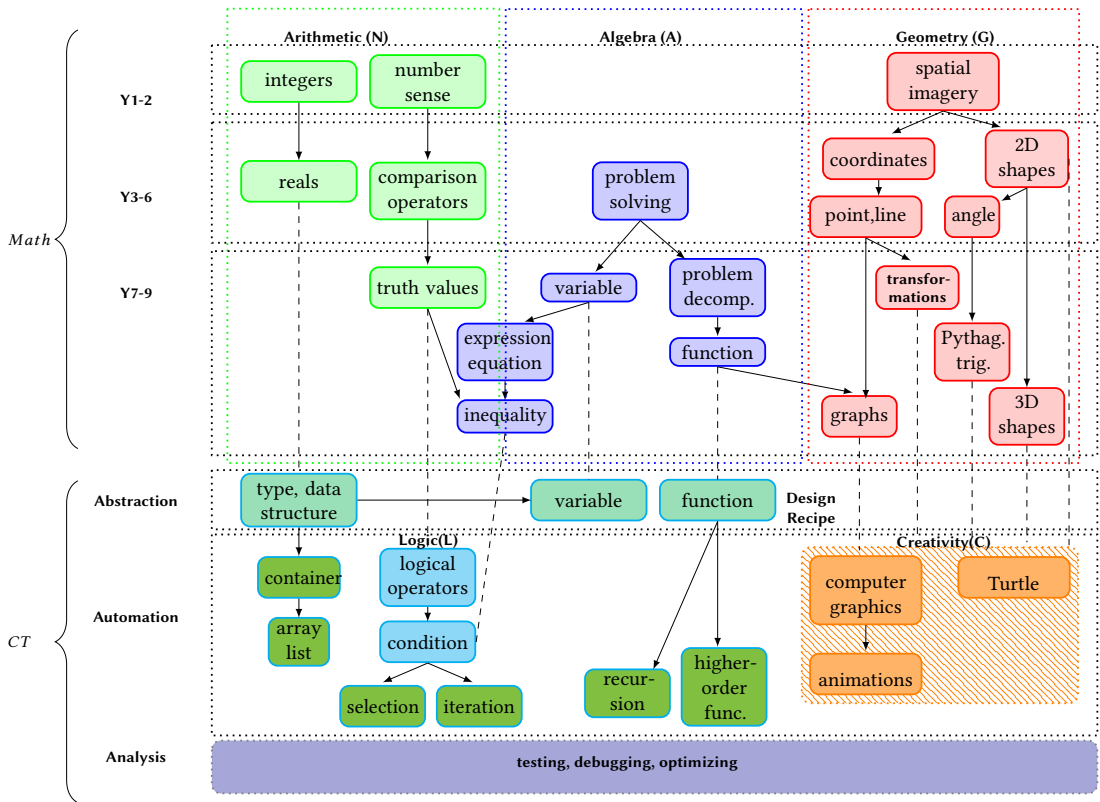


Figure 5: Hypothetical learning trajectories of CT

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; at present it seems that 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 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.

## 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](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 Shiram 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\\_opetusuunnitelman\\_perusteet\\_2014.pdf](http://www.oph.fi/download/163777_perusopetuksen_opetusuunnitelman_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/Ohjelmointikieliet-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, Shiram 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. 2008. Computational thinking and thinking about computing. *Philosophical transactions of the Royal Society of London A: mathematical, physical and engineering sciences* 366, 1881 (2008), 3717–3725.
- [40] Jeannette M Wing. 2010. Computational Thinking: What and Why? Link Magazine. (2010).





# Publication VII

Niemelä P. and Valmari A. “Elementary math to close the digital skills gap”, *Proceedings of the 10th International Conference of Computer Supported Education, 2018*

DOI: [10.5220/0006800201540165](https://doi.org/10.5220/0006800201540165)

Niemelä and Valmari (2018)

# Elementary math to close the digital skills gap

Pia Niemelä<sup>1</sup> and Antti Valmari<sup>2</sup>

<sup>1</sup>*Tampere University of Technology, PO Box 527, FI-33101, Tampere, FINLAND  
pia.niemela@tut.fi*

<sup>2</sup>*University of Jyväskylä, Jyväskylä, FINLAND  
antti.valmari@jyu.fi*

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

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

## 1 INTRODUCTION

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

The skills gap concerns not only the number of SW professionals but also the quality of their skills. The STEM shortage paradox highlights the peculiarity of having hard-to-fill open positions and at the

same time an excess of graduates who cannot find a job (Harris, 2014; Smith and White, 2017). One explanation is the skills mismatch, and in compliance with this, employers point out the candidates' incapability of breaking down problems into manageable chunks and solving them, and the gaps in technical, data modeling, and analytical skills. Accordingly, data base, data management, data analysis and statistics skills outnumber other requested digital skills of job advertisements in the US (Beblavy et al., 2016).

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

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

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

## 2 CS&SWE VS. ICT

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

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

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

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

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

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

## 3 RELATED WORK

### 3.1 ACM recommendations

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

#### 3.1.1 CS Knowledge Areas of ACM

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

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

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

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

KUs of Analysis, Strategies, Fundamental Data Structures, Automata, Computability, and Complexity.

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

#### 3.1.2 The most relevant math to support CS

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

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

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

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

## 3.2 SWEBOK recommendations

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

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

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

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

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

One obvious observation is a notably smaller portion of continuous math compared to traditional engineering education. In particular, calculus, differential equations, and linear algebra are missing. Instead,

several topics target a better position of underlying logic (2,3); and primers for data types, data structures and algorithms (1,4,5,9,11). In addition, subtopics of Basics of Counting (4), and Discrete Probability (6) and Number Theory (10) scaffold a deeper understanding of probability and cryptography. Numerical Precision, Accuracy, and Errors (9) section reveals underlying HW and memory specifics that have an effect on, e.g., the resolution of measurements and impossibility of expressing most real numbers precisely.

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

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

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

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

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

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

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

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

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

Matrices are extensively exploited, e.g., in machine learning, data analysis, pattern recognition, and game engines for 2D/3D-transformations. All suggested math syllabus areas remain at the preliminary level in UKNC and USCC and we propose the same: in logic truth tables and Boolean logic in order to simplify several simultaneous conditions; in sets, Venn diagrams and basic operations of union, intersection and cut with at most three sets; and in matrices, trans-

formations of translation, reflection, rotation and enlargement and finding an inverse matrix. This new math knowledge should be carefully bridged with the prior knowledge with lots of visual exercises and by starting early enough. Table 1 illustrates in which order these topics are handled in the UKNC and USCC.

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

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

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

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

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

## 4 Method

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

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

## 5 Results and Discussion

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

### 5.1 Feedback from SW engineers

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

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

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

The latter sub-figure is visually telling. Only al-

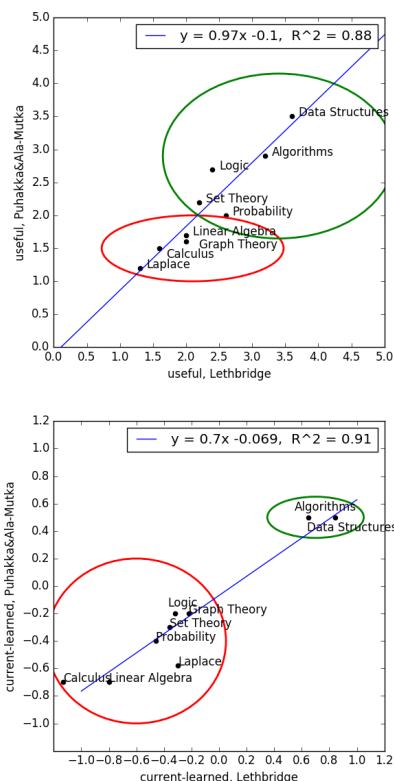


Figure 1: The comparison of usefulness and adequacy of math education evaluated by SW professionals (Lethbridge, 2000; Puhakka and Ala-Mutka, 2009,  $N = 181$ ;  $N = 212$ )

gorithms 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 math compared with continuous math is unchanged, yet the bias has an academic flavour. Discrete math scores highest among professors and lecturers (3.1).

### 5.2 CS-supportive math for elementary

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



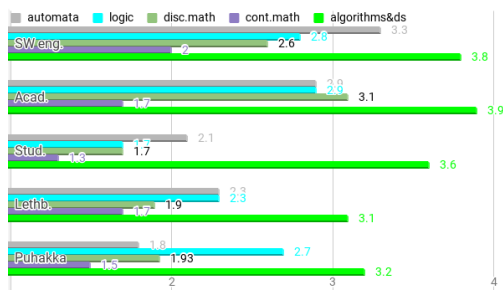


Figure 2: The math areas perceptions [1(not important), 4(very important)] of Surakka’s engineers, academics, and students contrasted with Lethbridge and Puhakka et al.;  $N = 11, 19, 24, 181, 212$ ; respectively

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

### Algorithmic thinking

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

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

The gradual division between human-completed calculation and computer-based computing has been the watershed between the disciplines of math and CS. In pondering the difference between the mindsets of mathematicians and computer scientists, Knuth points out that computer scientists need to

be concerned about algorithms and their computing specifics, such as the notion of complexity or economy of operations. In most programming languages, the computing process comprises a series of sequential state changes executed assignment-by-assignment, which is an operation absent in math. Moreover, data structures in CS are inhomogeneous, which spreads the spectrum of concerns compared with more convergent mathematical structures (Knuth, 1985), excluding the data structures of advanced set theory and logic.

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

### Data represented and modeled in multiple ways

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

Fig. 3 illustrates the use of the MathCheck learning tool (Valmari and Kaarakka, 2016) in studying the relationship between textual and tree representations. Such exercises aim at training the precedence and left- and right-associativity rules in particular. The exercises help students to grasp the distinction between semantics and syntax by differentiating between associativity as a semantic notion and left- and right-associativity as syntactic notions. Furthermore, the example in Fig. 3 reveals that the relation operators ( $=$  and  $\geq$ , 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,

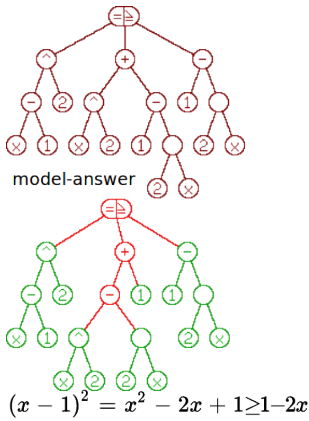


Figure 3: A tree representation of a model relation chain, and a failed student attempt to yield a similar tree

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

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

### Logic

In CS formalization, Dijkstra described its distinctiveness with a formula (Dijkstra et al., 1989):  $CS = math + logic$ . In accordance, he called students to learn formal math and logic to construct a well-grounded basis for CS. UKNC points out that already

a novice programmer at the elementary level needs simple Boolean logic, for example, the operators of AND, OR and NOT, and combinations, see Fig. 4. In the same context, logic gates in circuits are introduced. This connection between Boolean logic and logic gates can be used to create a link with electrical engineering and physics.

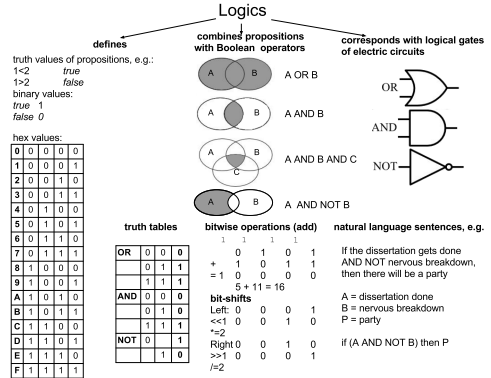


Figure 4: Logic in UKNC

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

### Sets, statistics, probability

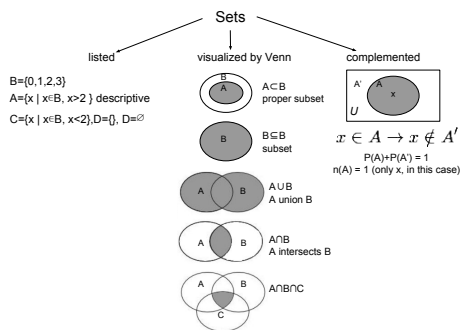


Figure 5: Sets in UKNC

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

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

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

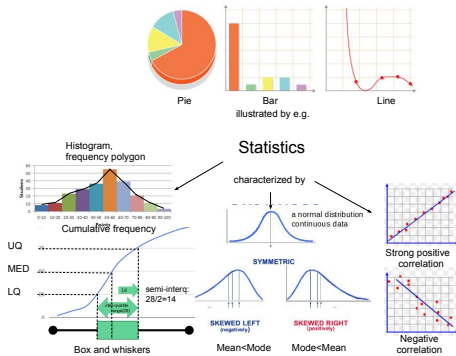


Figure 6: Statistics in UKNC

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

Venn diagrams and relative frequency charts prompt probability issues. The relative frequency of an event, e.g., which percentage of students are 140–150 cm tall, provides an obvious scaffold to investigate the probability of a randomly-selected student being 140–150 cm tall. In Venn, the bin of 140–150 cm students can represent the set  $A$ , where the complement set of  $\bar{A}$  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  $\bar{A}$  with 100% probability, i.e.,  $P(A) + P(\bar{A}) = 1$ . In Finnish elementary math, probability links closely with statistics in the described manner. In contrast, UKNC progresses further by in-

cluding the multiplication and addition rules (Fig. 7).

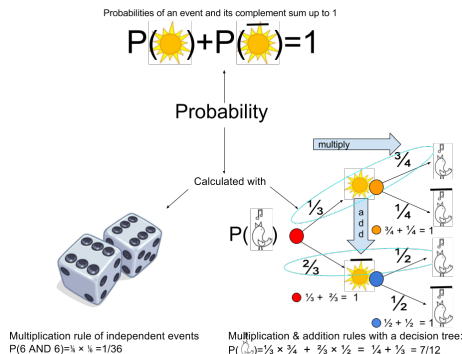


Figure 7: Probability in UKNC

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

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

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

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

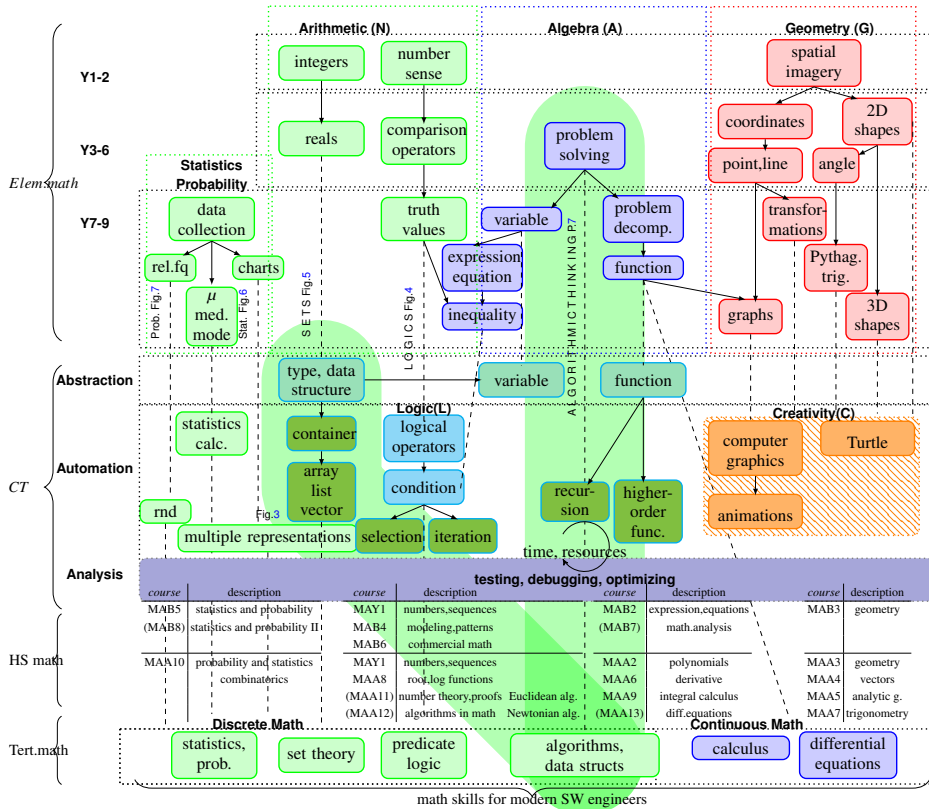


Figure 8: Hypothetical learning trajectories bridged from the FNC-2014 elementary to higher education math

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

The next layer of high school (HS) math is elective. It divides into A and B math: A is the ma-

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

## 6 CONCLUSIONS

### *RQ1: Math syllabus areas to be strengthened?*

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

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

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

### *RQ2: The overemphasized math syllabus areas?*

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

### Further studies

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

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

## ACKNOWLEDGEMENTS

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

## REFERENCES

- ACM&IEEE (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, December 20, 2013. Technical report.
- Ardis, M., Budgen, D., Hislop, G., Offutt, J., Sebern, M., and Visser, W. (2014). Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *Joint effort of the ACM and the IEEE-Computer Society*.
- Beblavý, M., Fabo, B., and Lenearts, K. (2016). Demand for Digital Skills in the US Labour Market: The IT Skills Pyramid. CEPS Special Report No. 154/December 2016.
- Bocconi, S., Chiocciariello, A., Dettori, G., Ferrari, A., Engelhardt, K., Kampylis, P., and Punie, Y. (2016). Developing Computational Thinking: Approaches and Orientations in K-12 Education. In *EdMedia: World Conference on Educational Media and Technology*, pages 13–18. Association for the Advancement of Computing in Education (AACE).
- Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.
- Charette, R. N. (2005). Why software fails. *IEEE Spectrum*, 42:42–49.
- Core Standards Organization (2015). Mathematics Standards — Common Core State Standards Initiative. [http://www.corestandards.org/wp-content/uploads/Math\\_Standards1.pdf](http://www.corestandards.org/wp-content/uploads/Math_Standards1.pdf).
- Core Standards Organization (2017). High School: Modeling. <http://www.corestandards.org/Math/Content/HSM/>.
- CSTA (2016). Computer science standards. [https://www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM\\_StandardsFINAL\\_07222.pdf](https://www.csteachers.org/resource/resmgr/Docs/Standards/2016StandardsRevision/INTERIM_StandardsFINAL_07222.pdf).
- Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, 52(6):28–30.
- Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, 60(6):33–39.
- Department of Education (2014). National Curriculum in England. Key stages 3 and 4 framework document.

- Dijkstra, E. W. et al. (1989). On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1398–1404.
- English Department for Education (2013). National Curriculum in England Computing programmes of study.
- Finnish National Board of Education (2014). Finnish National Curriculum 2014.
- Finnish National Board of Education (2015). National core curriculum for general upper secondary education. [http://www.oph.fi/download/172124\\_lukion\\_opetussuunnitelman\\_perusteet\\_2015.pdf](http://www.oph.fi/download/172124_lukion_opetussuunnitelman_perusteet_2015.pdf).
- Futschek, G. and Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. *Proceedings of the 2010 Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century (Constructionism 2010)*, pages 1–10.
- GCSE (2015). GCSE subject content for computer science. [https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/397550/GCSE\\_subject\\_content\\_for\\_computer\\_science.pdf](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/397550/GCSE_subject_content_for_computer_science.pdf).
- Harris, M. (2014). The STEM shortage paradox. *Physics World*, 27(10):56.
- Kitchenham, B., Budgen, D., Brereton, P., and Woodall, P. (2005). An investigation of software engineering curricula. *Journal of Systems and Software*, 74(3):325–335.
- Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, 92(3):170–181.
- Lamagna, E. A. (2015). Algorithmic thinking unplugged. *Journal of Computing Sciences in Colleges*, 30(6):45–52.
- Lethbridge, T. C. (2000). What knowledge is important to a software professional? *IEEE Computer*, 33(5):44–50.
- Liukas, L. (2015). *Hello Ruby*. A childrens’ book available in 22 languages.
- McGowen, M., DeMarois, P., and Tall, D. (2000). Using the function machine as a cognitive root.
- Meziane, F. and Vadera, S. (2004). A comparison of computer science and software engineering programmes in English universities. In *17th Conference on Software Engineering Education and Training (CSEE&T 2004), 1-3 March 2004, Norfolk, VA, USA*, pages 65–70. IEEE Computer Society.
- Parnas, D. L. (1985). Software aspects of strategic defense systems. *Commun. ACM*, 28(12):1326–1335.
- Parnas, D. L. (1999). Software engineering programs are not computer science programs. *IEEE Software*, 16(6):19–30.
- Peng, G. (2017). Do computer skills affect worker employment? An empirical study from CPS surveys. *Computers in Human Behavior*, 74:26–34.
- Pinar, W. F. (2012). *What is curriculum theory?* Routledge.
- Puhakka, A. and Ala-Mutka, K. (2009). Survey on the knowledge and education needs of Finnish software professionals. *Tampere University of Technology, Department of Software Systems*.
- Redecker, C. and Punie, Y. (2017). European Framework for the Digital Competence of Educators: DigCompEdu. EUR - Scientific and Technical Research Reports, The European Commission’s science and knowledge service.
- Smith, E. and White, P. (2017). A ‘great way to get on’? The early career destinations of science, technology, engineering and mathematics graduates. *Research Papers in Education*, 32(2):231–253.
- Surakka, S. (2007). What subjects and skills are important for software developers? *Communications of the ACM*, 50(1):73–78.
- Taub, R., Armoni, M., and Ben-Ari, M. (2012). CS unplugged and middle-school students’ views, attitudes, and intentions regarding CS. *ACM Transactions on Computing Education (TOCE)*, 12(2):8.
- Valmari, A. and Kaarakka, T. (2016). MathCheck: a tool for checking math solutions in detail. In *SEFI 2016 Annual Conference Proceedings*, pages VK.1–VK.9. European Society for Engineering Education SEFI.
- Wilkie, K. J. and Clarke, D. M. (2016; 2015). Developing students’ functional thinking in algebra through different visualisations of a growing pattern’s structure. *Mathematics Education Research Journal*, 28(2):223–243.
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences*, 366(1881):3717–3725.



# Publication VIII

Niemelä P., Partanen T., Mannila L., Poranen T. and Järvinen H.-M., “Code ABC MOOC for Math Teachers” *Revised Selected Papers. Vol. 865. Springer, 2018*

DOI: [10.1007/978-3-319-94640-5\\_4](https://doi.org/10.1007/978-3-319-94640-5_4)

Niemelä et al. (2017b)



# Code ABC MOOC for Math Teachers

Pia Niemelä<sup>1</sup>, Tiina Partanen<sup>2</sup>, Linda Mannila<sup>3</sup>, Timo Poranen<sup>4</sup>, and Hannu-Matti Järvinen<sup>1</sup>

<sup>1</sup> Pervasive Computing, Tampere University of Technology, Tampere, Finland,

<sup>2</sup> City of Tampere, Tampere,

<sup>3</sup> Aalto University, Espoo, Finland and Linköping University, Linköping, Sweden,

<sup>4</sup> Computer Science, University of Tampere

**Abstract.** Computing is the latest add-on to enhance the K-12 curricula of many countries, with the purpose of closing the digital skills gap. The revised Finnish Curriculum 2014 integrates computing mainly into math. Consequently, Finland needs to train math teachers to teach computing at elementary level. This study describes the Python and Racket tracks of the Code ABC MOOC that introduce programming basics for math teachers. Their suitability for math is compared based on the course content and feedback. The results show that conceptually the functional paradigm of Racket approaches math more closely, in particular algebra. In addition, Racket is generally regarded as more challenging in terms of syntax and e.g. for utilizing recursion as an iteration mechanism. Math teachers also rank its suitability higher because the content and exercises of the track are specifically tailored for their subject.

**Keywords:** Curriculum Research, Computer Science Education, K-12 Education, In-Service Teacher Training, MOOC, Computational Thinking, Math-integrated Computer Science, Python, Racket, Programming Paradigms, Imperative, Functional.

## 1 INTRODUCTION

Our society is becoming increasingly digitalized, which has therefore given rise to a global discussion on the role of computer science (CS) in K-12 education. As a consequence, a number of countries all over the world have introduced computational thinking, computing or CS (or aspects thereof) into their K-12 curricula. Since 2014, for instance, students in England have had Computing on their schedule from the age of five. In Finland, aspects of CS were included in the national curriculum in fall 2016, when the 2014 national curriculum came into force. Programming was introduced as part of the cross-curricular theme of digital competence, and also specifically integrated into the syllabi of crafts (Y3-9) and mathematics (Y1-9). In Y1-2, math teachers now need to help students learn how to create and test simple programs (unplugged, step-by-step instructions), while students in Y3-6 should learn how to program in a visual programming language. The new learning objectives for mathematics in Y7-9 intend to develop students' algorithmic thinking skills and applying programming in problem solving. The target is reached when "*a student can apply the principles of algorithmic thinking and is capable of implementing simple programs*" [15].

Integrating programming into comprehensive education is a remarkable change: both pre- and in-service teachers need to learn to program and to understand the core elements of computational thinking. Such curriculum enhancements change the job description of a teacher retrospectively. Consequently, employers are responsible for the need to train teachers and for providing time for sufficient professional development. Although this training need is recognized by the government, in-service training resources are insufficient as Finnish teacher training departments have not yet fully responded to the new requirements and the growing need.

Against this background, all voluntary training initiatives have been warmly welcomed by schools, principals and teachers. In this paper, we present the Code ABC MOOC, a project initiated informally by a group of volunteer educators to respond to the gap in teachers' formal training and preparedness for teaching programming. The initiative was later sponsored by the Technology Industries of Finland Centennial Foundation and the Finnish National Board of Education. The Code ABC MOOC offers four tracks targeted at teachers working at different school levels: ScratchJr (Y1-2), Scratch (Y3-6), Racket (Y7-9) and Python (Y7-9). In addition to supporting elementary school teachers in learning the basics of programming, the Code ABC MOOC also serves as a tool for highlighting best practices for teaching programming. This paper concentrates on the two tracks targeted at teachers of Y7-9, namely the Python and Racket tracks.

In this study we extract the key concepts and aspects of programming and computational thinking from the examined tracks of the Code ABC MOOC in an effort to strengthen the conceptual basis of the programming syllabus. The teacher feedback illustrates how these concepts have been adopted and evaluates the suitability of the material for supporting teachers in adopting the new curriculum. In addition, our study attempts to link these CS concepts to appropriate mathematics topics. In this regard, the differences between the programming languages used in the two tracks are noted and explained based on the underlying programming paradigms. Our goal is to answer three main questions:

- What CS concepts and computational thinking skills do these Code ABC tracks introduce?
- What topics did the teachers find challenging, inspiring, or suitable for math?
- Which programming paradigms do the tracks align with and how do they support math?

The paper is organized as follows. First we discuss previous work in fields related to our study, after which we describe our research context. Next we present and discuss the results, before concluding the paper with some final remarks.

## 2 RELATED WORK

### 2.1 CS in K-9 education

As noted in the introduction, introducing aspects of CS in K-9 education is an international trend. This is, however, not a new trend. As long ago as 1967, Papert developed the LOGO language [35], specifically aimed at teaching children how to program. His goal was to use programming as a tool to let children be creative with technology and to support their learning in other domains such as mathematics, the arts, languages and science. As computers became increasingly popular, accessible and easy to use, the focus in the school debate shifted from CS and learning how to program to IT and learning how to use computers and software. In the last five years, the trend has once again shifted, as the digital transformation has shown the need for understanding the digital world. Consequently, there has been an active debate on the need to shift the focus away from our future citizens being mere passive consumers of technology, and towards them becoming active producers.

Internationally, we now see an increasing trend for intergrating aspects of CS into K-9 education. For instance, in Europe the majority (17 out of 21) of countries participating in a survey conducted by the European Schoolnet in 2015 reported doing so [4]. The way in which this is accomplished varies. Some countries focus on K-12 as a whole, whereas others primarily address

either K-9 or grades 10-12. Some countries have introduced CS as a subject on its own (e.g. Computing in England) while others have decided to integrate it with other subjects, by, for instance, making programming a cross-curricular element (such as in Finland). Instead of computing and programming, some countries also use the term computational thinking.

Computational thinking (CT) has gained prominence, particularly in conjunction with the discussion on integrating aspects of CS as part of K-9 education. However, Papert had already set the course for CT much earlier, in 1996 [34]:

*“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”.* Not simply using computers but also creating digital artifacts is a valid stance – helping pupils to identify themselves as potential creators fosters their sense of empowerment.

In her seminal article, Wing [58] renewed this emphasis by establishing the term ‘computational thinking’ as an essential part of the recent CS education discourse, yet its comprehensive definition was omitted. To date, several definitions and operational descriptions of CT have been suggested. Even if no consensus on the definition has been reached, many of the suggestions build on the core of Wing’s later description (2010, [59]): *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.*

When operationalizing the term, it is commonly presented as a set of concepts and approaches. For instance, the support organization Computing at School (CAS) in England, describes CT as “the processes and approaches we draw on when thinking about problems or systems in such a way that a computer can help us with these.” (CAS, p. 1) They define CT in terms of six concepts (logic, algorithms, decomposition, patterns, abstraction and evaluation) and five approaches (tinkering, creating, debugging, persevering and collaborating).

**Relevance for our study** In the Finnish curriculum programming is to be taught as part of math [15]. Compared to programming, math already has a well-established syllabus that has evolved into its current state since the dawn of the educational system. Apart from a few minor syllabus areas being dropped from or reintroduced into the curriculum, the core content of the math syllabus has remained more or less the same for decades. In order to facilitate a smooth transition, there is a need to build on the well-established math core in order to introduce the analogous and logically progressive steps for CS. It is tentatively assumed that integrating CS into math will move the center of gravity of the syllabus towards CT.

## 2.2 Bidirectional transfer: *math* ↔ *CS*

Math is at the very core of programming, which requires algebraic, logic and problem-solving skills. Synergy implies mutual benefit between two entities, and although the benefits that a good understanding of math and perceived self-efficacy confer on the learning of computational skills are clear [62,44], the transfer in the other direction, from programming to math, may not be that obvious. In a successful transfer, however, a student should be capable of finding the common underlying conceptual bases of different topics [23]. Finding such analogies not only requires a certain level of intellectual maturity, but also that a student has elaborated on the learning material conceptually in order to reach a deeper understanding of the topic.

Transfer may happen either laterally or vertically [17], near or far, or by the low road or the high road [40] implying a certain hierarchy of learning. In addition, one of the two complementary subjects tends to be interpreted in the learners' minds in a more abstract manner, while the other focuses on its application [43]. In the case of math and CS, math is more abstract, while CS can be understood as applied math [8]. In math, educators have long talked about conceptual and procedural knowledge [18]: conceptual knowledge comprises a full possession of the appropriate concepts and the ability to link them together, i.e., the high road to knowledge transfer, while procedural knowledge (the low road) consists of well-internalized mathematical routines.

**Relevance for our study** Based on previous research, one potential assumption is that practicing math routines will provide an appropriate resource for programming exercises. To achieve this, the current math syllabus needs to be bridged with the corresponding programming topics. It seems reasonable to assume that this could be valuable not only to students, but also for in-service teachers, who might find similarities between math and programming motivating when learning to program themselves.

### 2.3 Programming metaphors, languages and paradigms

When attempting to determine the role of CS in education, various metaphors are used, e.g. thinking of programming as literacy, as a driver for the maker culture and a maker mind-set, or grounded math [7]. If the literacy metaphor is used, then programming as digital literacy emphasizes the same logical skills as are applied when constructing linguistically correct sentences, such as using and/or/not in order to express the internal logic of a sentence. From a 'maker mindset', the programming language should be as productive as possible, easy to learn ("low floor"), usable in a wide range of potential application areas and types ("wide walls") and facilitate the creation of both basic and advanced programs ("high ceiling"). In such contexts, scripting languages and visual programming languages such as Scratch ([scratch.mit.edu](http://scratch.mit.edu)) can be particularly useful.

The question of what programming language to use has been heavily debated throughout the years, in particular when discussing novice programming. At university level, most discussion has surrounded textual languages, such as Java, C, Python and Scala. Languages have been compared based on, for instance, how easy they are to learn, how many errors students make when using them, how verbal the languages are and potential syntactical overload. However, at the K-9 level this question has not been as actively discussed, for a number of reasons. First, programming at school level is still rather new, and thus still evolving into its final form. The goal of learning the basics of programming at school is not to educate future programmers, but rather to give them a basic understanding of the digital world. In addition, at the moment, educators seemed to reach a consensus about the programming progression at school, starting with unplugged activities, followed by visual programming languages with a textual language being introduced in the later grades (7–9). Nevertheless, there are some studies on programming languages at K-9 level as well. Despite the popularity of Scratch and other block-based languages, some studies have questioned the benefits of these in enhancing problem-solving skills and good programming practices [22,33] as these are claimed to lead to bottom-up development and fine-grained programming without coherence [33]. In sum, the problems relate to abstraction skills, i.e., "not seeing the forest for the trees" and designing the program in advance. On the other hand, other studies have found that visual programming languages such as Scratch make it easier for novices to learn some concepts, for instance, construct of conditions [27,31] and to switch to textual programming later on. Another consequence of using

block-based languages is the tight integration with the development environment (IDE), whether on-line or stand-alone. Such powerful IDEs become a new norm along with visual programming.

In addition to metaphors and languages, programming paradigms are essential for defining the angle of approach to teaching programming. Each paradigm provides its own basic concepts with paradigm-specific restrictions, which leads to different kinds of implementations and programming styles. Some tasks are more easily implemented in one paradigm, whereas other paradigms are more appropriate for other applications (e.g. due to their efficiency or flexibility). Consequently, there is not only a discussion around what programming language to use, but there are also recurring arguments about “the right paradigm for the job”. To be able to make well-informed language and paradigm choices, decision-makers and educators should have an adequate understanding of the alternatives available.

The division of programming languages into different paradigms is not easy, and is further blurred by multi-paradigm languages. Wegner (1989) simply divided languages into two fundamental categories: imperative and declarative [55]. In this division, the imperative paradigm comprises procedural, object-oriented, and distributed (parallel) languages, whereas the declarative one consists of functional, logic, and database languages. However, a parallel paradigm is not commensurate with, for instance, an object-oriented paradigm, which provides for parallel implementations as well. This problem can be circumvented by separating the programming model (sequential/parallel) from paradigms altogether, thus enabling new combinations. This change was proposed by Bal and Grune [3], who also raised the previous sub-paradigms of procedural, object-oriented, functional and logic to main categories.

Constantly increasing in number, multi-paradigm languages challenge this categorization. For instance, Scratch puts paradigm categorization to the test. Some of the features of Scratch comply with an object-oriented paradigm. According to Van Roy’s taxonomy, having a cell and a thread, i.e. assignment and concurrency, categorizes Scratch as an object-oriented paradigm [54]. However, the lack of data abstraction (inheritance) and functions makes the data encapsulation model of Scratch oxymoronic in regard to object-orientedness. Since Scratch targets only GUI applications reactive to user-generated events and inter-sprite messages, an event-loop [54] or event-driven paradigm [14] would seem to be a more accurate categorization. A few sources refer to Scratch as agent-based programming, where each sprite acts as an independent agent according to the defined rules. The interplay of such agents, for instance, facilitates easy STEM simulations [51].

**Relevance for our study** Since our study compares two tracks of the Code ABC MOOC, one using a functional language and the other an imperative one, this discussion of paradigms is important. In this section we look more closely at the paradigms most relevant to our study: imperative and functional programming, exemplified by Python and Racket respectively.

**Imperative paradigm and Python** Some argue that the imperative paradigm is appropriate for introducing programming as it makes it quite straightforward to translate algorithms into code, for instance. There are a wide range of imperative languages which can be used as general purpose languages and for particular application areas. Python (python.org) was originally designed with education in mind. The developer, Guido van Rossum, even suggested that everybody could master programming using Python back in 1999 [45]. The language had already aroused interest as a programming language for novices in the early 2000s, due to, for instance, its clear and readable syntax, strong introspection capabilities, natural expression of procedural code, high level dynamic

data types and its extensive standard library and third party modules providing functionality for a wide range of tasks. Python is one of the most commonly used languages in general use today (number 5 on the TIOBE programming index list in August 2017) and is widely used in education [21,10,9, etc.].

Guzdial [21] has a clear vision of the importance of web programming and sees Python as a viable tool for this, describing it as "one of the best web languages". However, he does not object to mixing paradigms and languages, exemplified by the Jython environment for students (JES) project, which combines Python with Java [5]. Nevertheless, Python is object-oriented even without Java by virtue of its own class model that provides e.g. polymorphism and multiple inheritance. Python easily interweaves multiple paradigms, although at the basic level it does fall into the imperative paradigm. Recently, Python was endowed with functional features as well, such as maps, comprehensions, and generators [28].

The Python material of the Code ABC track has been translated and modified based on the project outcomes of AP Computer Science Principles [19]. Already in 2003, Guzdial [20] championed the 'CS for all' ideology and the potential of CS over specific syllabus areas of math, such as calculus. He argues that teaching CS for all – not just for mathematically oriented students – should involve "sampling" instead of sorting. It is essential to allow space for students' authentic interests, such as arts, crafts, and music in the hunt for intrinsic motivation. Incorporating the maker mindset with tinkering, and creative and socially meaningful activities is especially beneficial for reaching less motivated student groups, including girls [6].

**Functional paradigm and Racket** In contrast to the multi-faceted nature of Python, the subset of Racket used in the Code ABC MOOC is more constrained and the closest metaphor would be "grounded math", where the pure functional programming language may be regarded as a realization of lambda calculus. Transfer between math and CS is claimed to be closest to the functional programming paradigm [26,49]. For example, functions in algebra can be practiced using functional programming languages. Combining functional programming with math is not new. Historically, attempts range from the early use of LOGO [16,25] to recent experiments employing Racket and Haskell [1]. While the results from the LOGO initiatives have varied [25], the Racket evaluations have been positive and stable [13,12,49,47].

The Racket programming language (<http://racket-lang.org>) is a multi-paradigm language, and thus also supports functional programming. Racket includes a programming IDE, DrRacket, designed especially for teaching purposes [13]. In contexts where DrRacket cannot be installed, the web-based environment WeScheme [61] steps into the breach, also enabling online sharing and remixes. DrRacket has built-in support for so-called 'Student Languages' starting with Beginning Student and ending with Advanced Student Language. Each of these Student Languages gradually introduces new programming primitives and concepts. Simplified syntax and semantics aim at helping beginners grasp the core concepts of function design, such as composition and function calls. Tool creators have also defined more precise error messages in order to assist novices in debugging and analyzing their code [30].

For the sake of the purity of the functional paradigm, the imperative features of Racket are held back. For instance, the assignment operation (set!) and functions causing side effects (display, read) are not introduced until the Advanced Student Language level. Felleisen et al. wrote the guide 'How to Design Programs' for high school and college level students [12], and in its most recent version the imperative features are done away with altogether to introduce appropriate coding practices.

The guide systematizes problem solving with Design Recipe, which teaches how to divide a problem into smaller solvable steps in the process of designing functions with a test-driven approach [12]. The use of Design Recipe has been shown to foster the right order of operations and the composition of nested functions. Thus, Felleisen and Krishnamurthi suggest that functional programming provides the strongest evidence for the favorable effects of programming on math skills [13].

A number of articles promote Racket's Beginning Student Language as a prominent way of learning algebra [26,49], especially with well-designed instructions. These should include purposefully planned exercises and pedagogical models, such as the Cycle of Evaluation [49], which visualizes expressions and the use of parentheses. The algebraic approach of functional programming has been shown to improve the understanding of math concepts such as variables and functions [60,49,48].

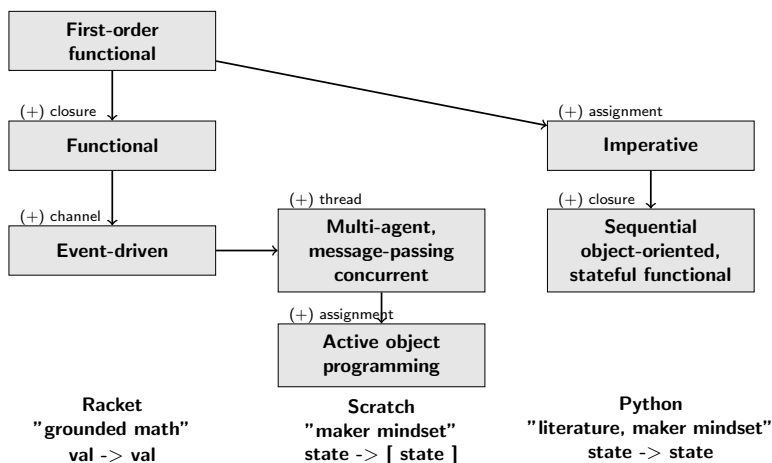


Fig. 1. The Code ABC paradigm taxonomy complying with [54]

**Comparison and Summary of the Paradigms of the Code ABC** In summary, Figure 1 illustrates the Code ABC, which is comprised of functional, event-driven, and imperative paradigms. In line with the paradigms, the figure also illustrates the corresponding language used in the MOOC and the closest metaphors. Obviously, these metaphors are speculative. Different "paradigm camps" tend to adhere to their own discourse: Scratch promoters, led by Resnick, highlight sharing and the unimpeded creation of one's own artifacts with the analogy of virtual LEGO construction, i.e. block snapping [42,32]. The founder of the Python language, Van Rossum, emphasizes the readability and efficiency of code [46], whereas the Racket camp regards functional computing as being rooted in lambda calculus, thus inherently connected with math, in particular algebra [49,48,13].

Van Roy categorizes languages based on their declarativeness and expressiveness [54,53]. In his fine-grained paradigm taxonomy, Van Roy defines a horizontal axis of declarativeness based on whether a state is unnamed or named, and adds expressiveness step-by-step (for instance, assignment, closure, channel, and thread) in order to evolve the paradigm taxonomy from simple to more

complex concepts. In distinguishing between functional and imperative paradigms, the diagnostic question is: can you assign a variable, i.e. have a named state? The answer divides paradigms into either imperative or functional. An imperative paradigm is statement-centric, the assignment being a statement as well. Each statement changes the state of the system, hence, imperative computation may be understood as state transformations in a sequence, i.e.,  $state \rightarrow state$ . This is in contrast to the functional paradigm, which may be described as sequential value transformations,  $val \rightarrow val$  without states. A named state enables modularity and the storage and management of updatable memory, which moves the paradigm in a less declarative direction.

A closure establishes a new variable scope in the context of a function. It 'closes' both a pointer to code and an environment for free variables. In the functional paradigm, closures are a central concept because they enable nested and higher-order functions that can access data from the outer scope, i.e. variables of the previous frames in the stack. Higher-order functions are a powerful substitute for e.g. for/while iterations without incrementable counters. Otherwise, control structures would be cumbersome to implement.

The event-driven paradigm leans on events that trigger execution, e.g. at the user's initiative. An event may trigger a message to be broadcast through a channel (or a port). In Scratch, several loosely-coupled receivers may listen to the same message [29]. Because the order of receivers is not determined, the concurrency model comprises explicit sending and implicit receiving, which implies a non-deterministic final state, i.e.,  $state \rightarrow [state]$ . Infinite forever loops are implemented as threads that enable concurrency.

### 3 The Code ABC

The initial idea for the Code ABC MOOC was introduced in spring 2015 by Tarmo Toikkanen and Tero Toivanen. The original concept was to help teachers learn programming using material that has been prepared especially for them by their peers, for instance, more experienced teachers. The first course, the so called Code ABC beta, was held in Autumn 2015 with three tracks: ScratchJr, Scratch and Racket. The Python track was added for the spring 2016 version of the MOOC. So far (fall 2017) 3649 participants have studied programming in the Code ABC MOOCs.

The initial three tracks of the Code ABC beta (ScratchJr, Scratch, and Racket) were developed by a group of Finnish

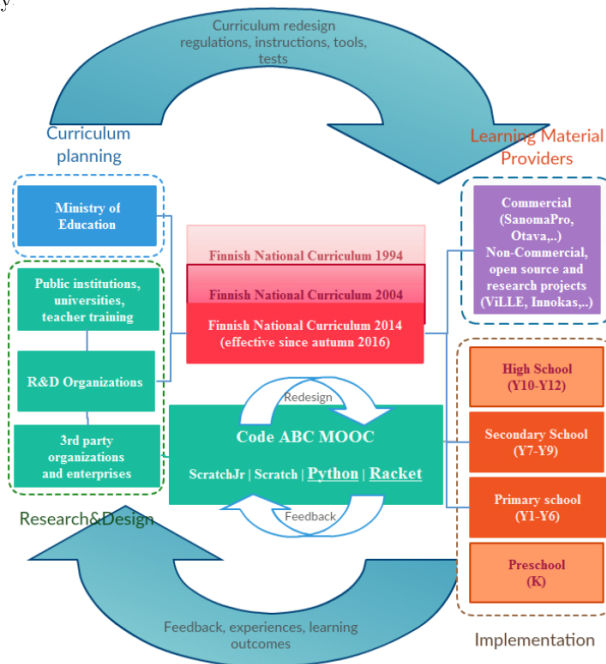


Fig. 2. Nested DBR cycles of curriculum updates (update/10yrs) and Code ABC tracks (2 updates/yr)



teachers and were improved incrementally based on the feedback from several iterations. The continuous development followed the principles of design based research (DBR), aiming at linking theory and practice in the discipline of education [41]. DBR stipulates the use of several iterations and redesigns of an educational artifact based on feedback and experience. Figure 2 illustrates the process of two nested design cycles.

The outer cycle is the curriculum planning cycle that takes place once a decade, while the inner is the iterative process of developing the Code ABC tracks twice a year. Development proceeds in cycles, taking into account the feedback given by different stakeholders, especially the customers, which in this context are in-service teachers. The artifact is then redesigned together with course instructors and researchers, whose research interests lie in integrating CT into elementary education.

The original material for the Python track was developed in the USA in similar cycles by Guzdial and Ericson [11]. This material was translated and adapted for the Code ABC MOOC in a project funded by the Finnish National Board of Education, coordinated by the Innokas Network at the Faculty of Educational Sciences at the University of Helsinki and implemented jointly by the Faculty of Educational Sciences and the Department of Computer Science at the University of Helsinki and the Department of Computer Science at Aalto University.

### 3.1 Design goals of the Code ABC tracks

The first three tracks of the Code ABC (ScratchJr, Scratch, and Racket) had a number of general goals: promoting creativity; introducing CS as a tool for creating something new and inspiring; sharing pedagogical ideas and artifacts during the course; using exercises directly applicable in a classroom context in order to make it easier for teachers to get started; offering course participants sufficient content knowledge so that they would not limit themselves to applying ready-made programming materials but also be able to create their own exercises; and enabling peer-support by encouraging participants to help each other on discussion forums. The Racket track had an additional design goal of integrating mathematics into programming exercises in order to motivate math teachers to adopt programming in their teaching, and to prove that programming lessons are not time wasted.

The main goals of the Python track were, according to the goals of the original material [10], to increase teachers' knowledge of computer science concepts as well as to improve teachers' confidence in their ability to teach CS principles. The course was designed as a lightweight learning experience [11], allowing busy teachers to participate when they had 20-60 minutes to spare. The exercises were designed to be small and feature low cognitive loads, which was achieved by placing relevant examples just before the exercises. The course did not focus on any individual subject such as mathematics – on the contrary, the material was aimed at anyone interested in teaching programming and CS.

### 3.2 Course implementation

The Code ABC MOOC was implemented using the A+ learning platform developed by Aalto University (<https://plus.cs.hut.fi/>). Piazza was utilized as the discussion platform and Rubyrlic for showcasing and peer reviewing returned artifacts [2]. Both the Python and the Racket tracks grouped learning objects into entities; termed modules in Python, and topics in Racket. For the sake of consistency, we will use 'topic' for both in this paper. At the end of each topic, feedback was collected with Grader, an on-line survey tool developed at Aalto University. Grader was also

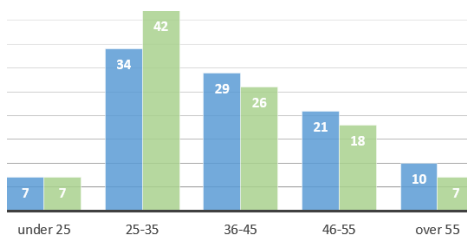
used to collect pre- and post-course surveys. The feedback was collected in order to further develop the courses. Teachers attended the Code ABC MOOC tracks free of charge. Both tracks spanned several months from February to May, 2016. No compensation was granted for course participants except for 2-3 credit points from the Open University of Helsinki after course completion (2 cp for Python (P), 3 cp for Racket (R)).

## 4 Study design and data collection

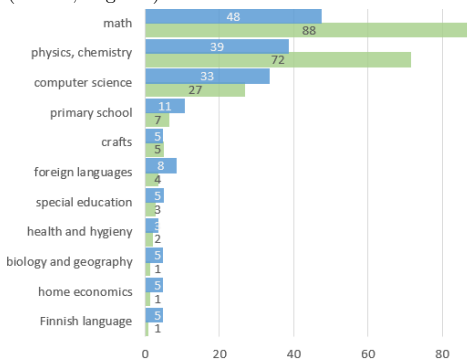
Our study is based on the Spring 2016 course implementation consisting of the second iteration of the Racket track and the Finnish translation of version 2 of the Python material. We conducted a pre-course survey to get background information about the participants (Python N=320, Racket N=137).

The largest participant group represented 25 to 35 year-olds (Fig. 3) the majority of whom were female (Python track 65%, Racket track 78%). Math teachers formed the largest groups (Fig. 4) in both tracks (Python track 48%, Racket track 88%). The next largest groups were science and CS teachers. The majority of the participants represented the original target group, i.e., K-9 teachers (Python track 72%, Racket track 91%).

Based on the survey, most participants had some previous experience in programming (Python track 78%, Racket track 74%). Quite a few had already used programming in their teaching (Python track 48%, Racket track 38%). In order of popularity, the languages previously used by the Python track participants were Scratch 34%, Java 32%, Basic 28%, Pascal 25%, C++ 23%, JavaScript 21%, C 18%, Python 17%, Visual Basic 17%, FORTRAN 12%, LOGO 11%, C# 6% and Perl 5%. In Racket, the order of popularity was Scratch, 34%, Java 26%, C++ 23%, Pascal 22%, Basic 20%, Python 15%, Visual Basic 14%, JavaScript 10%, FORTRAN 9%, LOGO 8%, and C 7%. Other languages were only mentioned by 5% of the participants or less.



**Fig. 3.** Age distribution of course participants in % (P blue, R green)



**Fig. 4.** Subjects taught by the teachers (P blue, R green; a subject omitted if P and R less than 5%)

#### 4.1 Python and Racket tracks in Spring 2016

The Python track (content described in Table 1) was implemented as a localized translation of the ebook Computer Science Principles: Big Ideas in Programming [10]. The course was organized in five topics in accordance with the original material: introduction to computing, naming, repetition, decision making, and data manipulation. Participants concluded the course by writing an essay on the pedagogical aspects of programming.

The original material is arranged in a book format, and should thus be read in a sequential manner. All the exercises are embedded in the ebook’s browser environment and they are automatically assessed and graded. The material has been designed to follow an example-exercise format to facilitate learning [10]. Multiple exercise types are used: multiple choice and fill-in-the-blanks test conceptual understanding, Parson’s problems are used for teaching basic programming constructs, and exercises consisting of modifying active code segments in the on-line programming environment provide wider opportunities to try out the concepts learned [36]. In addition, the material utilizes the code lens concept to demonstrate program execution [11].

As the original material did not specifically focus on how programming should be taught in Finnish schools, the Code ABC MOOC had an additional course project, during which the teachers designed a 2-hour lesson on any subject of their choice. The course projects were reviewed by both peers and course staff. The lessons that the teachers had designed typically reflected the subjects that they taught in schools, ranging from learning languages to applying CS in crafts. The participants had to complete 85% of the automated exercises and the final essay in order to pass the course.

The Racket track was designed so that different aspects of algorithmic thinking (abstraction, logic, repetition) were introduced side by side, starting from easier ideas and progressing to more advanced topics [39]. Altogether, the course content comprised seven topics: introduction to programming, control structures, functions and design recipe for functions, recursion, user interactions, lists, and higher-order functions with Turtle graphics (content described in Table 1). The very core of the Code ABC Racket track material is to reveal the nature of programming as a sort of applied mathematics, and to show how mathematics can be taught through programming. Hence most programming exercises are in a math context. The implementation of the Racket track was inspired by the Systematic Program Design online course offered by edx.org [24] and the material was constructed following the same procedure:

1. Short motivational video, in which the lecturer introduced the contents and the purpose of the exercises. A few videos also responded to questions from the previous week.
2. Tutorial videos introduced the core concepts as screen captures. The lecturer demonstrated the concepts to be learned with DrRacket. Its stepper tool was extensively employed in explaining the evaluation rules. Concise lecture notes were available on-line as well, but the majority of the course content was provided as videos. The course participants were expected to test the programming examples themselves while watching the videos.
3. The Design Recipe was used to demonstrate the principles of function design, see Figure 5. By using the recipe, a user can solve one detail at a time and proceed step-by-step until the whole

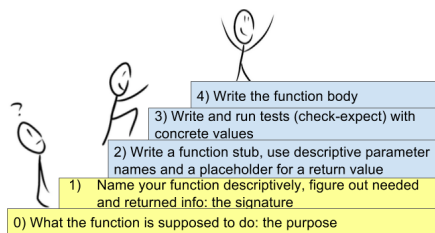


Fig. 5. The Design Recipe [12] presented as a staircase fostering a step-by-step design [37]

function is ready. The implicit intent is to solidify writing test cases before implementing the actual function body, which complies with the test-driven development.

4. The exercises and their solutions were delivered as both DrRacket and WeScheme files and used as self-tests of the course content presented in the video tutorials.
5. Hands-on exercises differed from the Systematic Program Design exercises as self-review for code and multiple choice quizzes were not used. Also, the final course project was an essay about the pedagogical aspects instead of a programming project as in Systematic Program Design.

The programming exercises and their solutions were taken from the Coding for schools student’s material [38] and the Coder’s handbook [37], which contains documentation for the graphics and animation libraries (2htdp/image and 2htdp/universe), Beginning Student Language primitives, and new libraries for turtle graphics (Racket Turtle) and user interactions (display-read).

### Participation and course completion levels

Fig. 6 shows the number of participants per topic. By "enrolled" we mean participants who registered to the MOOC, whereas "started" refers to participants who showed some activity in the first topic (completed one exercise or gave feedback). The numbers 1-7 refer in Racket case to participants who completed the corresponding topic. The numbers 1-5 in Python case refer to participants who completed the feedback form at the of each topic. The Racket track was started by fewer participants (N=171) than the newly introduced Python track (N=399). Both courses lost participants during the course period, but in the end more participants completed the Racket track (N=100, 58% of the started participants) than the Python track (N=66, 17% of the started participants). If the percentages are calculated per enrolled participants we get lower values for completion rates: Python 12% and Racket 31%. Only 80% of the course work was required to pass in Racket track, which allowed skipping one topic. This explains the lower numbers for topics 5, 6 or 7 in Fig. 6.

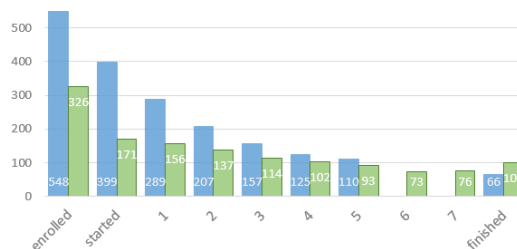


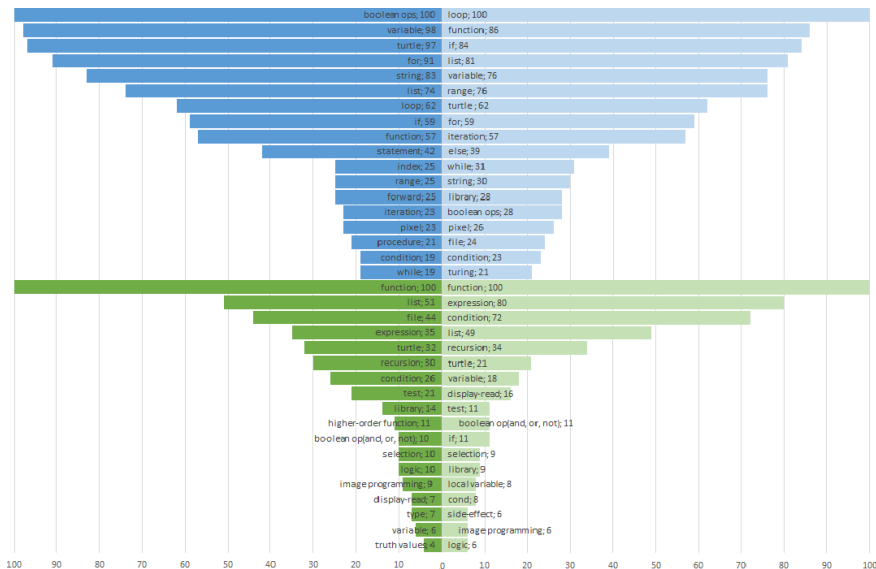
Fig. 6. Number of participants/topic (P blue, R green)

## 4.2 Research methods

The methods of the study are curriculum research and content analysis of the participants’ feedback. Curriculum research examines the most central concepts of the curriculum, which in our context correspond to CS concepts in the Code ABC MOOC. In deriving the main concepts, we counted their frequency in content, that is, how often they occurred. We also asked participants what they had learned after each topic, and their responses were similarly analyzed by counting the occurrence frequencies. After extracting the most central CS concepts, the paradigm-oriented differences were examined more thoroughly. The goal of the review was to check how good a match the respective paradigms were for teaching math.

## 5 Results

### 5.1 What CS concepts and CT skills do the Python and Racket tracks introduce?



**Fig. 7.** To the left the relative word frequencies of the MOOC material (P blue, R green); to the right teachers' feedback to the question "What did I learn?"

Figure 7 illustrates the topics taught vs. the topics learned for each course. The brighter-colored bars to the left correspond with the relative word frequencies in the course content. The most frequent word has a value of 100%, and other frequencies are compared to this maximum. The lighter-colored bars to the right represent the course content based on participants' responses to the question "What did I learn?"

In the figure, the blue Python tornado lies above the green Racket tornado. From the shape of the tornados we can conclude that the wider Python tornado covers a larger range of concepts with about equal intensity. In contrast, the Racket course focused on functions and a handful of other concepts. The relative similarity of the mirrored right side indicates that the participants in both tracks seemed to learn the intended concepts.

The Python topics in the upper blue tornado indicate that the main concepts of the Python track were control structures (selection and iteration). Boolean operators (and, or, not) were widely exploited in the program examples, including conditions for iterations, such as for and while loops. All selection and iteration-related topics appear in the list: boolean ops (1<sup>st</sup>), for (4<sup>th</sup>), loop (7<sup>th</sup>), if (8<sup>th</sup>), index (11<sup>th</sup>), range (12<sup>th</sup>), iteration (14<sup>th</sup>), condition (17<sup>th</sup>), while (18<sup>th</sup>).

Naming and variables (2<sup>nd</sup>) were the second most central concepts, reflecting the stateful and assignment-oriented nature of Python as an imperative language. We also group statements (10<sup>th</sup>) here, as a superclass, including functions (9<sup>th</sup>), procedures (16<sup>th</sup>) and assignments. By comparing

functions and procedures the material highlighted the difference of functions returning a value and procedures lacking return values.

The third topic group were applications, exemplified by turtle (3<sup>rd</sup>) and pixel-level image editing (15<sup>th</sup>). Concept-wise, these applications do not bring anything new, but rather give students the opportunity to put the pieces together while working on engaging problems. The last group consists of data types and structures of string (5<sup>th</sup>) and list (6<sup>th</sup>); in the course, strings are named (used as variables) and lists are iterated. So, list would fit in the iteration of the first group as well.

The participants echoed these emphasis areas, apart from the Turing machine and file areas, which received more emphasis than they had in the text. The Turing machine, completeness and halting problems, were used to explain the history and most prominent ideas of CS. Files were introduced in the last topic (Data handling) simultaneously with related functions such as `open` and `close`. Statements and index were not mentioned.

The analysis of the Racket course concepts is only partial for technical limitations; we were not able to analyze video material since it was not transcribed into textual format. The analysis is based on the lecture notes and textual material in the course platform, so the analysis might be missing some concepts that were taught but are not showing in Fig 7.

In the green Racket tornado functions, expressions, conditions, lists, recursion and Turtles form the top 6 concepts mentioned on both taught and learned sides. Testing, boolean operators and library usage are mentioned next. The high frequency of the 'file' concept on the lecture side is explained by the fact that the term appeared frequently throughout the course instructions ("Save your code in a file", "Send your file for peer-review", etc) even though file handling was not explicitly taught. The participants reflected, quite faithfully, the same concepts except higher-order functions and type, which were missing from the list of concepts mentioned by the participants.

## 5.2 What topics did the teachers find challenging, inspiring, or suitable for math?

The participants evaluated each topic after completion using 5-point Likert-scales. The evaluation was based on a set of criteria such as difficulty level, enthusiasm, and suitability for their teaching. The following subsections present the results of these evaluations.

**Challenging topics** The average level of difficulty experienced for each topic is given in Fig. 8 (1=not challenging enough, 5=far too challenging). For both tracks, the difficulty level increased during the first four topics. After the fourth topic in the Racket track, the difficulty level decreased, whereas it continued to increase in the Python track. Topic 4 in the Racket track (recursion) was considered most challenging. Animations (topic 3) and lists (topic 5) are the next most challenging. Starting from topic 5 the challenge level in Racket starts

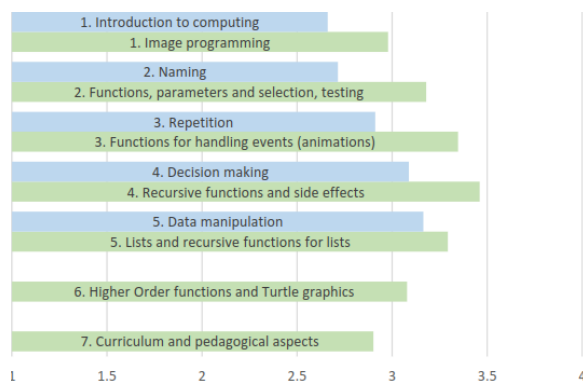
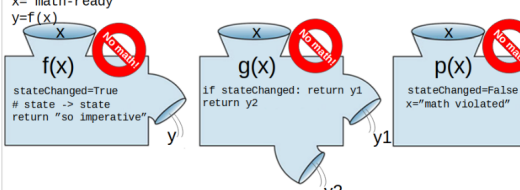
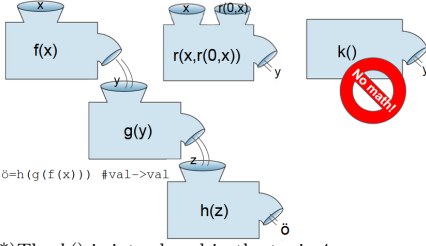
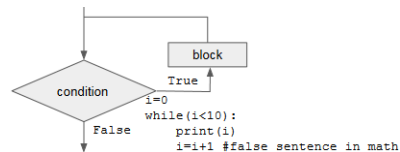


Fig. 8. Difficulty level/topic (P blue, R green)

**Table 1.** Fundamental concepts and CT related aspects of the Python and Racket tracks with regard to the underlying paradigm.

Python (imperative)	Racket (functional)
<p>1 Basic operations, computing.            What is a computer, program (Python vs. Java)?            Background, Turing machine (completeness).</p>	<p>Introduction to Racket programming,            global constants.            CT: problem decomposition using functions</p>
<p>2 Naming (including variables) applied to numbers, strings, objects such as turtles and images, as well as functions            string: substrings, indexing.            Functions (<math>f(x)</math>, <math>g(x)</math>) vs. procedures (<math>p(x)</math>).</p> <pre> stateChanged=False x="math-ready" y=f(x)           </pre> 	<p>Control structures: selection (<code>if</code>), truth values, comparisons.            Function definition: purpose, signature.            Unit tests using <code>check-expect</code></p>  <p><math>\delta = h(g(f(x)))</math> #val-&gt;val</p> <p>*)The <math>k()</math> is introduced in the topic 4.</p>
<p>3 Control structures: iteration (<code>for</code> and <code>while</code>).            Iteration based on list or counter condition, e.g.,            a list [1,2,3] iterated based on range:  <code>for item in list: block</code>  <code>while (condition): block</code>            Block: indentation-grouped commands            CT: becoming aware of iterative patterns.</p>	<p>The Design Recipe[12]: test-driven development.            Selection (<code>cond</code>): comparisons, predicates.            Logical ops for combinations: <code>and</code>, <code>or</code>, <code>not</code>.            Interactive applications: animations, mouse events            CT-abstraction: Design Recipe.</p>
<p>4 Blocks by indentation.            Control structures, also nested, selection: <code>if</code>, <code>elif</code>, and <code>else</code>.            Decision making:            – condition (logical expression) in iteration or selection            – logical operators for combinations.  <code>if condition: block</code>  <code>else: print("condition false")</code></p> <p>CT abstraction: flowchart illustrating the control flow.</p> 	<p>Reading user input with <code>display-read</code> (user interaction causes side-effects);            the user input stored into a local variable.            Recursion, here factorial <math>n!</math> as an example:</p> <pre> n=3, y=? (* 3 (factorial 2)) n=2, y=? (* 2 (factorial 1)) n=1, y=? (* 1 (factorial 0)) return 1*1 return 2*1 return 3*2           </pre> <pre> (define factorial   (lambda (n)     (if (= n 0) 1         (* n (factorial (- n 1)))))) (factorial 3)           </pre>
<p>5 Data handling: collections and files.            Collection operations:</p> <ol style="list-style-type: none"> <li>indexing</li> <li>string: <code>split</code>, <code>find</code>, <code>substring</code></li> <li>list: <code>len</code>, <code>range</code>, <code>for-each</code>.</li> </ol> <p>Reading files: <code>open</code>, <code>close</code>, <code>readlines</code>.            Conventions: <code>commenting</code>.</p>	<p>Iterating lists recursively, producing new lists or one accumulated value.            Image files.</p>
<p>6 Revision, extra material.</p>	<p>Iterations cont.            Higher-order functions: <code>map</code> and <code>apply</code>, lists.</p>
<p>7 Finnish curriculum reflections in regard to CT/CS, and how to integrate computing with own teaching subject.</p>	

to decrease. Explanation might be that there were less exercises in topic 5 than in the previous topics. Also topic 6 covered Turtle graphics programming, which is conceptually simpler than functions and recursion.

The list below provides a few free-text feedback snippets describing challenging topics:

### Python

- Repetition: a number of participants were not capable of constructing loops on their own
- Decision: the difficulty level rose sharply compared to the previous modules; too much information and challenge
- Image processing: exercises felt hard and difficult to understand for beginners, and did not foster learning repetition
- Data manipulation: A few exercises were too difficult, e.g. in searching data from a file, one really needs to know what each function returns

### Racket

- The most challenging topics included recursion, animation, lists, and loops
- Recursion: content of topic 4 was clearly too much. It should have been split into two separate topics
- Lists: topic 5 also challenged a number of participants

**Inspiring topics** The average enthusiasm score for each topic is shown in Fig. 9 (1 = not inspiring at all, 5 = extremely inspiring). The highest levels of enthusiasm were reported in Racket for Turtle graphics (topic 6) and Image Programming (topic 1). This is in line with Toikkanen’s [52] findings of the mesmerizing effect of Turtle throughout all the Code ABC MOOC tracks. Young students immediately start drawing Logo-like figures after discovering the `pen.down` function. In addition, animation (topic 3) inspired a good number of participants.

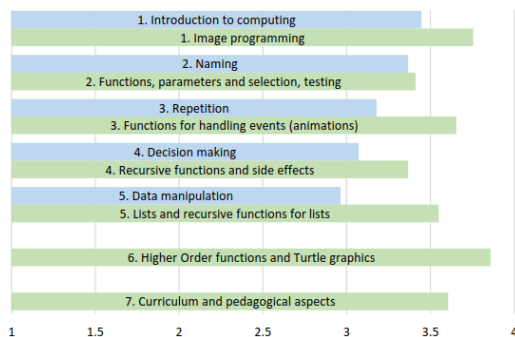


Fig. 9. Level of enthusiasm/topic (P blue, R green)

For the Python track, the difficulty level (Fig. 8) and the level of enthusiasm seem to go hand in hand, i.e. the more difficult the material, the less enthusiasm experienced. In Racket, this trend is less visible. For example, animations (topic 3) are considered to be both challenging and inspiring at the same time. The lower levels of enthusiasm for the Python material could also be due to the fact that the material was originally meant for a different audience, whereas the Racket material was specifically designed for in-service math teachers. For instance, the math teachers did not find Python’s image processing particularly fit for their purposes. After the second module, the clamour for hands-on programming in order to learn became louder; yet during the first module, the Parson’s problems were regarded as both easy and motivating. Overall, the participants questioned the usefulness of some exercises because they lacked ready-made student material that could be utilized in a school context, which is probably also reflected in the enthusiasm scores. Below we summarize the most inspiring aspects of the courses in order. We also provide some excerpts from teachers’ responses (translated from Finnish).



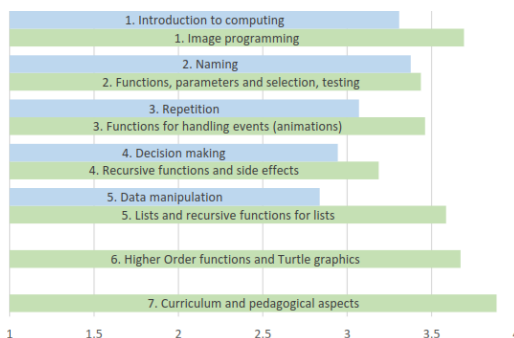
## Python

- Turtle images were fun and inspiring. *I was proud to be able to modify the Turtle code so that it formed a house*
- Image processing was inspiring as well. *As an art teacher it was easier to understand (than math)*
- Data Manipulation: Working with the Small Particles Data example (looking for values in the list, calculating averages), and the possibility to investigate real-life problems were beneficial

## Racket

- Image programming: drawing and designing own images. Possibility to see code, games and images drawn by other participants gave new ideas for teaching, i.e., sharing artifacts promoted creativity and increased enthusiasm

**Suitable topics** Fig. 10 represents the average suitability scores for each course topic (1 = not suitable at all, 5 = extremely suitable). The highest suitability score was given to Racket’s last topic, which includes material about CS as a new addition to the curriculum, CT, and pedagogical approaches to teaching the new content. In addition, the participants wrote an essay on ideas stimulated during the course and/or a lesson plan for integrating programming into their teaching. The next most suitable topics were Racket’s Image Programming (topic 1) and Turtle Graphics (topic 6). Topic 5 scored high as well, featuring a quiz that utilized recursive function and lists in implementation. The least suitable was topic 4, which employed recursion and user interaction.



**Fig. 10.** Suitability of the course topics (P blue, R green)

Python scored notably lower in suitability. As mentioned, the material was not originally designed for math teachers, although these made up half of the participants (48 %). Hence, the generic material did not meet the teachers’ needs. Surprisingly in this regard, the Data Manipulation topic (number 5) averaged as the least suitable topic, even though it included real-life applications from, for instance, the statistics domain. Nevertheless, some teachers noted its value:

## Python

- Turtle graphics could be used to teach geometry (focuses on angles and side lengths)
- Repetition, Decisions: using numbers, strings, turtles and images consistently within all topics was perceived positively
- Data manipulation (Statistics):
  - *In data analysis, real world problems made me understand what programming can be used for (environmental science and geography, analyzing air pollution in USA) and its usefulness in analyzing big amounts of data*
  - *Easy access to online data made me think I could use this in my teaching*
  - *Using statistics in mathematics could be useful (if I were a math teacher)*

## Racket

- Possibility to utilize exercises in the school context
- Functions (Algebra): content was designed specifically for math teachers (including the exercises and utilizing the functional paradigm), thus the material and tools were useful for teaching

### 5.3 Which paradigms do the tracks align with and how do they support math?

Neither of the Code ABC tracks pronounced explicit paradigm considerations, nor were they present when the participants evaluated the topics. More experienced programmers compared languages – not paradigms – focusing mainly on the learning threshold (the lower the better) and differences in syntax, e.g. confusion with the excessive amount of parentheses and the prefix notation of Racket. However, several topics inherently implied paradigm-related issues. In the Python track, the Naming topic introduced data mutability and immutability, variable assignments, and in accordance with variables the most common misconceptions as well. The Python material highlighted the difference between functions and procedures, and introduced code division in the form of reusable modules and libraries. Python contains a class structure and an option for object-orientation but, in this context, Python was classified as imperative even though a few objects, such as turtles, were extensively utilized throughout the MOOC (in effect requiring the introduction of dot notation).

Similarly, the Racket material does not explicitly emphasize the underlying paradigm and hardly mentions the term ‘functional’ at all. However, the built-in principles of Felleisen’s ‘How to design a program’ [12] recommend avoiding imperative features and re-assignment of global variables. These principles aim at a purely functional programming style. However, the enhancement of `display-read` indicates the need for pointing out side-effects contradicting this purity: `display-read` interacts with a user. With regard to the variables in Racket, it is possible to define constants and `let` allows local variables to be assigned. These can still not be re-assigned without the `set!` command, forbidden by functional paradigm purists.

Having no re-assignable variables – thus no loop counters – has its implications for iterations as well. Although looping lists with the command of `foreach` is still possible, missing a re-assignable counter leads towards recursion and higher-order functions, where recursion calls “fake” mutability with expressions as function arguments and by returning partial results. Higher-order functions, such as `map`, `apply` or `filter`, creates new lists or accumulates values, based on given lists and functions. Later, these functional list-handling mechanisms were introduced in the Python language as well, along with lambda calculus and list comprehensions. In Python, however, there are “imperative” ways of looping (such as `for` and `while`), leaving a minimal need for recursion compared to Racket. In both tracks, the paradigm-influenced alternatives for implementing iteration and selection structures were the core content of the whole course.

As the paradigms reside implicitly in the material, it is no surprise that the teachers do not pay much attention to them. As with learning to drive a car, all attention is first drawn to the main aspects of driving – not to comparing the features of various car models. Nevertheless, teachers with previous programming experience compared the features of the course language to their previously learned languages. For instance, teachers in the Python track stated that *I learned that Python is simpler than Java (lists and their handling), I learned for and foreach loops and list modifications, and In my experience, Python requires less lines of code. Another nice thing was that when defining variables you don't need to think as much as in Java.*

In Racket, on the other hand, the teachers’ first impressions in particular were as follows: *I learned Racket's syntax. Writing mathematical expressions is cumbersome compared to Python, Java, Pascal, Ruby, and Visual Basic and If you consider using Racket at school, it is relatively complex compared to, for instance, Java.* On a more positive note, Racket teachers also noted some benefits: *Racket is really engaging! The first exercise was well selected: it is nice to immediately achieve some colorful shapes instead of the traditional “Hello World”. I did not know that programming can be this much fun!*

**Algebra** lies at the core in terms of math focus. It includes fundamentals such as functions, variables, statements and expressions, which are fundamental not only in math, but also in CS. In CS, functions and variables all encompass implementation mechanisms. However, the differences between concepts in these two disciplines may cause misconceptions and programming errors that are difficult to detect. In Table 1, module 2 compares the differences between functions in both tracks. Compared with Racket, Python allows remarkable freedom of implementation, which becomes particularly visible with functions.

In math, the function definition dictates at least one function parameter as input, and one and only one as output. It is possible to write a function without parameters in both languages, such as  $k()$  (the rightmost figure in the 2<sup>nd</sup> row), which is not valid in math. In Python, a procedure  $p(x)$  may return no value explicitly (in which case Python implicitly returns *None*), which is not acceptable in math, if  $x$  is in domain. Neither may a function return multiple values with the same input, as does the function  $g(x)$ , which returns either  $y1$  or  $y2$  based on the state.

What is wrong with the  $f(x)$  on the Python side then? On the face of it, nothing: it gets at least one input as a parameter (in this case  $x$ ) and returns only one output ( $y$ ), as specified. However, the function body changes the global state of the outer program by re-assigning the global variable `stateChanged`, i.e., it causes a side-effect. Functions causing side-effects are not allowed in math. The idea of functional purity in the Racket track prevents side-effects, and in Python, no side-effects arise if the programmer is aware of the pitfalls and is capable of avoiding them. In accordance with a pure functional paradigm, immutable data and having no side-effects makes, for instance, parallel execution possible: different threads handling the same data can rely on the validity of the data.

In Racket, variables are essentially constants. In math, variables are also constants in the context of evaluating the value of an expression. The variables do not change during the evaluation, but between evaluations. For example, in the case of function  $y = f(x)$ ,  $x$  changes when the position moves on the x-axis. Thus, in math, the term *variable* can be deceptive, because variable does not actually vary. Instead, the terms *a symbol* or *a representative* might give a more authentic view of the true meaning of the concept.

In the imperative paradigm, the situation with variables becomes even more obscure with the counter-type behavior. As an example, see the fourth row in Table 1, where the `while` loop exploits the value of  $i$  to decide when to stop. In the loop body,  $i$  is incremented with the assignment of  $i = i + 1$ . In math, this statement makes no sense. In CS, the statement is split by the equal sign to enable the left and the right value to be referred to separately. The left value opens a gaping rabbit hole to the underlying world of hardware specifics and constraints that are normally carefully guarded. In essence, it represents the memory address to which the right side – still a normal expression – is assigned.

Re-assignment is a dangerous operation and provides an endless source of error; for instance, if types are mixed in assignment. Thus, typing relates closely to variables. In static typing, a type must be given when a variable is defined and it is checked during the compilation. In dynamic typing, variables need not be assigned a specific type, but it may change assignment-by-assignment in runtime: now an integer, after the next assignment maybe a string. The operations allowed for integers differ remarkably from those allowed for strings. Strong typing prevents operations on invalid types [50], while prevention leads to compile and runtime errors when these are detected. Both Racket and Python are dynamically and strongly typed languages.

**In Arithmetics**, types of integer and decimal numbers provide more in-depth affordances. Typing relates to number sets in math, such as integers, and rational and irrational numbers. In math,

gradual progression from simple to more complex operations results in changes in the respective number sets as well. With addition and multiplication, a student remains in the comfort zone of integers. With subtraction, the student may move into negative numbers. A substantial paradigm shift happens when division is introduced, which along with fractions transfers a student into the zone of rational numbers, represented as decimals in code.

The ultimate challenge at elementary level is irrational numbers, which are met when a ratio is never-ending and non-repeating. Irrational numbers result e.g. from square root operations, and surds, such as  $\sqrt{2}$ , are never-ending. In a computer context, the limits of the physical memory allocated to each variable complicate the handling of such irrationalities. Historically in CS, selecting the right type has been important due to the constraints of memory size: a number has to be cut when the allocated bytes are used up resulting in the cut part being lost. The type implies the number of bytes in use, which influences the preciseness of a number.

Preciseness is also a consideration in many scientific calculations, for instance when rounding and defining significant numbers. In math,  $\sqrt{2}$  or trigonometric expressions such as  $\sin(60)$  are exact. However, when represented in decimal format they are not, irrespective of the length of the type of `float`. All in all, in Arithmetics, when calculating basic and advanced operations, a computer can be compared to a calculator, with which students practice new arithmetic functions such as `abs`, `sqrt`, or `exp`, and drill the right order of calculations.

In the Python material, math equations, such as speed-distance-time calculations, exemplify the use of a newly introduced mathematical functions. The Racket material is more geometry-oriented, placing greater emphasis on calculating areas, angles, and perimeters. Even if extensively used in examples, Geometry is not central to understanding the concepts of CS. Our study, however, notes its value as an area providing visually appealing applications.

**Logic** and logical operations (`and`, `or`, `not`) combine conditions into more complex conditions. Conditions – or logical expressions – fall into the area of logic. However, in the current math syllabus, logic does not have a prominent position. As logical expressions prompt control structures, the natural progression would be to learn logic first. Consequently, control structures and the use of conditions might fit within both Logic and Arithmetics.

Selections, or decisions as the Python track calls them, correspond to inequalities in math: 1D inequalities, such as  $x > 0$ , are represented on a number line. Whether the line is open-ended or close-ended depends on the comparison operator: `<` and `>` result in open-ended lines, while `≤` and `≥` result in close-ended ones. In 2D inequalities, such as  $y > x$ , the line divides the coordinate plane into two halves, one of which is shaded. An open-ended condition is represented as a dashed line, a closed condition as a solid one. Consequently, inequalities can be expanded to 3D as well.

Conditions can also be combined. When multiple inequalities hold simultaneously, the number-line is cut into segments, and lines define geometric shapes in a coordinate plane, most often triangles. In 3D, conditions may result in solid geometry shapes, such as prisms. In addition, conditions apply to piece-wise defined functions, which, for instance, have discontinuity points or behave differently depending on the range of  $x$ .

Iterations or recursive structures are relatively rare in elementary math. The accumulator pattern introduced in both tracks can be used in Statistics, e.g., when calculating mean values, or when looking for *min* or *max* values. Later,  $\Sigma$  and  $\prod$  operations are applied iteratively to the defined number domain and these new notations abbreviate e.g. the previous calculation of the mean. In Pre-Algebra, recognizing growing patterns prepares for sequences, abstracted later as functions in Algebra [57,56]. It is axiomatic that sequences and their sums and series are iterative. In inductive

problem solving, a student determines the  $n^{\text{th}}$  term in a sequence. Instead of the iterative `for` and `while` loops favoured by Python, Racket favours recursions. In contrast to induction, which starts from the first and aims at finding the  $n^{\text{th}}$  term, recursion starts from the  $n^{\text{th}}$  term and aims at reaching the first. The recursive calculation of factorial ( $n!$ ) illustrates the product type iteration; see topic 4 in Table 1. Basic operations of Probability, combinations and permutations, make extensive use of factorials.

## 6 CONCLUSIONS

We have studied two approaches to teaching programming to in-service elementary school teachers in Finland. The majority of the participating teachers were mathematics teachers, which is understandable, given that programming has been added to the math syllabus in the new curriculum. Below we summarize our findings.

**What CS concepts and CT skills do these Code ABC tracks introduce?** Both tracks covered a substantial amount of basic programming concepts in their respective programming languages, such as subprograms (functions or procedures), conditional structures, boolean logic, data types, and lists. The Python track provided a generic synopsis of programming basics. As a primer, Python provided a history and some general knowledge about computers and CS. After this history review, the track focused on imperative programming fundamentals, such as assignment and `for/while` loops. Each new concept was demonstrated using numbers, strings, images, and Turtles. The Racket track, on the other hand, presented programming as yet another way of learning math rather than as a generic tool. Programming appeared as a new means of problem solving by exploiting functions. In addition to functions, control structures of selection and iteration were introduced. Iterations consisted of recursion and higher-order functions that manipulated lists.

CT links math and CS. When solving a problem, dividing the problem into smaller subproblems is essential (decomposition). In the context of programming, subproblems are subprograms, i.e. functions. Functions were discussed substantially more in Racket than in Python. As a recipe for a well-planned function, the Racket track introduced Design Recipe by Felleisen [12]. This Recipe promotes test-driven development: unit tests are implemented before a function body. Both courses emphasized the importance of using descriptive names for functions and variables, and the need for clear comments in order to improve readability; these coding conventions may be considered to be part of CT as well.

### What topics did the teachers find challenging, inspiring, or suitable for math?

- Challenging:
  - The most challenging topics in the Python track were data and image processing, evidently due to the extensive exploitation of Repetition and Decision structures. Furthermore, the difficulty level suddenly rose when moving from Repetition to Decision.
  - In Racket, the most challenging topics included recursion (loops), animation, and lists, by far the most challenging of which was recursion.
  - The Racket track required a significant amount of effort because of the hands-on exercises and complex topics. Frequently, the exercises took more time than expected. This was experienced as a challenge by the participants, who had to take care of their normal work duties during the course.

- Inspiring (enthusiasm in the survey):
  - Turtle graphics were considered inspiring in both tracks. In Python, Turtle moves exemplified both Repetition and Decision topics, while in Racket, Turtle was linked with higher-order functions, which also rank high in the list of most challenging CS topics
  - In Python, the participants were interested in learning more about the history of CS, and the prominent and influential persons behind it. The Image Processing exercises divided opinion, some considered it interesting while others found it difficult, tedious, and a rather useless topic for the target group, (math teachers). Data processing was also received with mixed feelings. Some appreciated the real-life applications but a number of participants regarded it as too difficult.
  - The Racket participants valued highly creative and playful open-ended exercises allowing them to create their own 'art', even though this was not considered to be "traditional math" or central to conceptual learning. Sharing artifacts with others was one significant factor in creating enthusiasm and promoting creativity.
- Suitable for teaching:
  - In Python, suitability ranked significantly lower than in Racket. Even when the Racket track was challenging, it scored higher. The difference in suitability scores indicates that the course content should be tailored to better suit the target group, in this case math (and science) teachers.
  - In the Racket track, the participants regarded the pedagogical essay, image programming, Turtle graphics, animation and quiz as the most suitable and interesting.
  - Suitability and enthusiasm seem to correlate.

**Which paradigms do the tracks align with and how do they support math?** Conceptually, the functional paradigm is closer to math, in particular in its representation of functions and variables. The imperative paradigm comprises more elements that are foreign to pure math. As imperative, Python might call for less effort in its approach, but it contains built-in hazards that may cause misconceptions and programming errors. For instance, re-assigning a global variable changes the state and function outcome, thus conflicting with the mathematical definition of a function. In these paradigms, the meaning and importance of a variable varies as well. A variable's visibility is defined by its scope (local/global). In functional Racket, global variables are constant and re-assigning local variables is not advisable either. In Python, global variables can be re-assigned anywhere and types will change accordingly, which is indefensible from the viewpoint of math. Without having become used to re-assignable variables and the possibility of comparing, a novice programmer will learn the functional programming smoother and regard it more suitable and inspiring. In contrast, an experienced imperative programmer lacks his normal means of exploiting variables, which causes frustration.

The Finnish curriculum integrates CS into math without allocating more time to teach it. This necessitates making the CS syllabus as close as possible to math: no time can be wasted on learning irrelevancies or concepts causing unnecessary confusion. The curriculum, however, should determine the targeted CS concepts more precisely. Languages should be categorized based on those concepts and their math-suitability in order to make justified tool selections. Systematic research and various learning experiments will enable determination of the concepts, computational thinking skills, and teaching practices best suited to closing the digital skills gap, as stipulated by the Finnish Curriculum 2014.

## 7 ACKNOWLEDGMENTS

We gratefully acknowledge the grant support of and the Academy of Finland (grant number 303694; *Skills, education and the future of work*), the Finnish National Board of Education and Technology Industries of the Finland Centennial Foundation that enabled the research and the development of the Code ABC MOOC. In addition to the funders, we thank the Aalto University A+ and Rubyric teams for their efforts to continuously improve the MOOC platform. Last but not least, thanks to Tarmo Toikkanen, Tiina Korhonen, Otto Seppälä, and Arto Hellas for providing Code ABC MOOC material and corrections for this paper.

## References

1. Alegre, F., Moreno, J.: Haskell in Middle and High School Mathematics. In: TFPiE. vol. 1. EPTCS, Sophia-Antipolis, France (2015)
2. Auvinen, T., Karavirta, V., Ahoniemi, T.: Rubyric: an online assessment tool for effortless authoring of personalized feedback. Workingpaper, ACM (2009)
3. Bal, H.E., Grune, D.: Programming language essentials. Addison-Wesley (1994)
4. Balanskat, A., Engelhart, K.: Computing our future: Computer programming and coding-Priorities, school curricula and initiatives across Europe (2014)
5. Barr, V., Guzdial, M.: Introducing CS to newcomers, and JES as a teaching tool. Communications of the ACM 59(11), 10–11 (2016)
6. Brady, Corey and Orton, Kai and Weintrop, David and Anton, Gabriella and Rodriguez, Sebastian and Wilensky, Uri: All Roads Lead to Computing: Making, Participatory Simulations, and Social Computing as Pathways to Computer Science. IEEE Transactions on Education 60(1), 59–66 (2017)
7. Burke, Q., Burke, Q.: Mind the metaphor: charting the rhetoric about introductory programming in K-12 schools. On the Horizon 24(3), 210–220 (2016)
8. Dijkstra, E.W.: How do we tell truths that might hurt? In: Selected Writings on Computing: A Personal Perspective, pp. 129–131. Springer (1982)
9. Ericson, B., Adrion, W.R., Fall, R., Guzdial, M.: State-Based Progress Towards Computer Science for All. ACM Inroads 7(4), 57–60 (2016)
10. Ericson, B., Guzdial, M., Morrison, B., Parker, M., Moldavan, M., Surasani, L.: An eBook for teachers learning CS principles. ACM Inroads 6(4), 84–86 (2015)
11. Ericson, B.J., Rogers, K., Parker, M., Morrison, B., Guzdial, M.: Identifying design principles for cs teacher ebooks through design-based research. In: Proceedings of the 2016 ACM Conference on International Computing Education Research. pp. 191–200. ICER '16, ACM, New York, NY, USA (2016), <http://doi.acm.org/10.1145/2960310.2960335>
12. Felleisen, M., Findler, R., Flatt, M., Krishnamurthi, S.: How to Design Programs, Second Edition. MIT-Press (2014), <http://www.ccs.neu.edu/home/matthias/HtDP2e/>
13. Felleisen, M., Krishnamurthi, S.: Viewpoint Why computer science doesn't matter. Communications of the ACM 52(7), 37–40 (2009)
14. Fesakis, G., Serafeim, K.: Influence of the familiarization with scratch on future teachers' opinions and attitudes about programming and ICT in education. In: ACM SIGCSE Bulletin. vol. 41, pp. 258–262. ACM (2009)
15. Finnish National Board of Education: National core curriculum for basic education 2014. In: National Core Curriculum for Basic Education 2014. Publications, Finnish National Board of Education (2016), <https://www.ellibs.com/fi/book/9789521362590/national-core-curriculum-for-basic-education-2014>
16. Futschek, G.: Algorithmic thinking: the key for understanding computer science. In: International Conference on Informatics in Secondary Schools-Evolution and Perspectives. pp. 159–168. Springer (2006)

17. Gagné, R.M.: *The Conditions of Learning*. New York: Holt, Rinehart and Winston (1965)
18. Gray, E.M., Tall, D.O.: Duality, ambiguity, and flexibility: A proceptual view of simple arithmetic. *Journal for research in Mathematics Education* pp. 116–140 (1994)
19. Guzdial, M.: Drumming up support for ap cs principles. *Communications of the ACM* 59(2), 12–13 (2016)
20. Guzdial, M., Soloway, E.: Computer science is more important than calculus: The challenge of living up to our potential. *SIGCSE Bulletin* 35(2), 5–8 (2003)
21. Guzdial, M.J., Ericson, B.: *Introduction to computing and programming in Python, a multimedia approach*. Prentice Hall Press (2009)
22. Gülbahar, Y., Kalelioglu, F.: The effects of teaching programming via Scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education - An International Journal* 13.1(Vol13.1), 33–50 (2014)
23. Jarvis, S., Pavlenko, A.: *Crosslinguistic influence in language and cognition*. Routledge (2008)
24. Kiczales, G.: UBCx: SPD1x Systematic Program Design - Part 1 (version 1, summer 2015) (2015)
25. Kulik, J.A.: *Meta-analytic studies of findings on computer-based instruction, Technology assessment in education and training, vol. 1*, pp. 9–34. Psychology Press (1994)
26. Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., Werner, L.: Computational thinking for youth in practice. *ACM Inroads* 2(1), 32–37 (2011)
27. Lewis, C.M.: How programming environment shapes perception, learning and goals: Logo vs. Scratch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*. pp. 346–350. ACM (2010)
28. Lutz, M.: *Learning Python: Powerful Object-Oriented Programming*. Safari Books Online, O'Reilly Media (2013), <https://books.google.fi/books?id=ePyeNz2Eoy8C>
29. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10(4), 16 (2010)
30. Marceau, G., Fislser, K., Krishnamurthi, S.: Measuring the effectiveness of error messages designed for novice programmers. In: *Proceedings of the 42nd ACM technical symposium on Computer science education*. pp. 499–504. ACM (2011)
31. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Learning computer science concepts with scratch. *Computer Science Education* 23(3), 239–264 (2013)
32. Monroy-Hernández, A., Resnick, M.: FEATURE empowering kids to create and share programmable media. *interactions* 15(2), 50–53 (2008)
33. Orni Meerbaum-Salant and Michal Armoni and Mordechai Ben-Ari: Habits of programming in scratch. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. pp. 168–172. ACM (2011)
34. Papert, S.: An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning* 1(1), 95–123 (1996)
35. Papert, S., et al.: *Logo philosophy and implementation*. Logo Computer Systems Inc (1999)
36. Parsons, D., Haden, P.: Parson's programming puzzles: a fun and effective learning tool for first programming courses. In: *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. pp. 157–163. Australian Computer Society, Inc. (2006)
37. Partanen, T.: *Coding for schools: Coder's Handbook (in Finnish)*. <http://racket.koodiaapinen.fi/manuaali/> (2016)
38. Partanen, T.: *Coding for schools: Student exercises (in Finnish)*. <http://racket.koodiaapinen.fi/tehtavat/> (2016)
39. Partanen, T., Mannila, L., Poranen, T.: Learning programming online: a racket-course for elementary school teachers in Finland. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. pp. 178–179. ACM (2016)
40. Perkins, D.N., Salomon, G.: Teaching for transfer. *Educational leadership* 46(1), 22–32 (1988)
41. Reimann, P.: Design-based research, pp. 37–50. *Methodological choice and design*, Springer (2011)



42. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B.: Scratch: programming for all. *Communications of the ACM* 52(11), 60–67 (2009)
43. Rich, P.J., Leatham, K.R., Wright, G.A.: Convergent cognition. *Instructional Science* 41(2), 431–453 (2013)
44. Robert W. Lent and Frederick G. Lopez and Kathleen J. Bieschke: Mathematics self-efficacy: Sources and relation to science-based career choice. *Journal of counseling psychology* 38(4), 424 (1991)
45. van Rossum, G.: *Computer programming for everybody* (1999)
46. Rossum, G.V.: Python Programming Language. In: *USENIX Annual Technical Conference*. vol. 41, p. 36 (2007)
47. Schanzer, E., Fisler, K., Krishnamurthi, S., Felleisen, M.: Transferring skills at solving word problems from computing to algebra through Bootstrap. In: *Proceedings of the 46th ACM Technical symposium on computer science education*. pp. 616–621. ACM (2015)
48. Schanzer, E., Fisler, K., Krishnamurthi, S., Felleisen, M.: Transferring skills at solving word problems from computing to algebra through Bootstrap. In: *Proceedings of the 46th ACM Technical symposium on computer science education*. pp. 616–621. ACM (2015)
49. Schanzer, E.T.: *Algebraic Functions, Computer Programming, and the Challenge of Transfer* (2015)
50. Scott, M.L.: *Programming language pragmatics*. Morgan Kaufmann (2000)
51. Sengupta, P., Kinnebrew, J.S., Basu, S., Biswas, G., Clark, D.: Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies* 18(2), 351–380 (2013)
52. Toikkanen, T., Leinonen, T.: The Code ABC MOOC: Experiences from a Coding and Computational Thinking MOOC for Finnish Primary School Teachers. In: *Emerging Research, Practice, and Policy on Computational Thinking*, pp. 239–248. Springer (2017)
53. Van-Roy, P.: Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music* 104 (2009)
54. Van-Roy, P., Haridi, S.: *Concepts, techniques, and models of computer programming*. MIT press (2004)
55. Wegner, P.: Guest editor’s introduction to special issue of computing surveys. *ACM Comput. Surv* 21, 253–258 (1989)
56. Wilkie, K.J.: Students’ use of variables and multiple representations in generalizing functional relationships prior to secondary school. *Educational Studies in Mathematics* pp. 1–29 (2016)
57. 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), 223–243 (2016; 2015)
58. Wing, J.M.: Computational thinking. *Communications of the ACM* 49(3), 33–35 (2006)
59. Wing, J.M.: *Computational Thinking: What and Why?* Link Magazine (2010)
60. Wright, G., Rich, P., Lee, R.: The influence of teaching programming on learning mathematics. In: *Society for Information Technology & Teacher Education International Conference*. vol. 2013, pp. 4612–4615. editlib.org (2013)
61. Yoo, D., Schanzer, E., Krishnamurthi, S., Fisler, K.: WeScheme: the browser is your programming environment. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. pp. 163–167. ACM (2011)
62. Zeldin, A.L., Pajares, F.: Against the odds: Self-efficacy beliefs of women in mathematical, scientific, and technological careers. *American Educational Research Journal* 37(1), 215–246 (2000)

Tampereen teknillinen yliopisto  
PL 527  
33101 Tampere

Tampere University of Technology  
P.O.B. 527  
FI-33101 Tampere, Finland

ISBN 978-952-15-4183-4

ISSN 1459-2045