Teaching How to Teach Computational Thinking

Anna Lamprou
School of Education
Northwestern Switzerland (PH FHNW)
Windisch, Switzerland
anna.lamprou@fhnw.ch

Alexander Repenning
School of Education
Northwestern Switzerland (PH FHNW)
Windisch, Switzerland
alexander.repenning@fhnw.ch

ABSTRACT

Computational Thinking is argued to be an essential skill for the workforce of the 21st century. As a skill, Computational Thinking should be taught in all schools, employing computational ideas integrated into other disciplines. Up until now, questions about how Computational Thinking can be effectively taught have been underexplored preventing efforts to cross the large gap between early adopters and the early majority, conceptualized as the Computer Science Education chasm. A promising strategy to cross the chasm is underway in Switzerland. Switzerland recently introduced a national curriculum, called Lehrplan 21, mandating Computer Science Education. This mandate requires the Computer Science education of elementary and middle school students. In 2017, the School of Education of Northwestern Switzerland (PH FHNW), introduced a mandatory pre-service teacher Computer Science Education course, to satisfy this mandate. All the PH FHNW students who study to become elementary school teachers must pass this two-semester course. The first part of this course was taught for the first time in fall of 2017. This paper presents the philosophy of this course and an initial analysis of both qualitative data capturing the students' perceptions of Computational Thinking and quantitative data describing shifts in students' skills and attitudes as effect sizes. The data suggest that it is possible to teach a basic understanding of programming to non-self-selected pre-service elementary school teachers.

CCS CONCEPTS

• Social and professional topics → Computational thinking • Social and professional topics → Computer science education • Social and professional topics → K-12 education

ACM ISBN 978-1-4503-5707-4/18/07…$15.00
https://doi.org/10.1145/3197091.3197120

KEYWORDS

Computer Science Education, Computational Thinking, Pre-service teacher, K-12 education, Primary school

ACM Reference format:

Anna Lamprou and Alexander Repenning. 2018. Teaching How to Teach Computational Thinking. In Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18). ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197120

1 INTRODUCTION

Computational Thinking (CT) is described as a key skill for the workforce of the 21st century [1]. Wing also argues, that CT is a basic skill for all humans, not just computer scientists [1]. Following from that, CT should be taught everywhere, especially at school level. Since its conception by Papert [22] and its broader introduction by Wing [1], CT has been the subject of intense and long discussions within the Computer Science Education (CSEd) community. In the center of attention are questions about CT's importance and definition but also about how this important skill can be effectively taught [2, 3]. Questions surrounding CT's definition and importance even though far from resolved, have been intensively discussed to offer a consensus about its importance [28] and a basic framework for a common definition. However, questions about how CT can effectively be taught remain largely underexplored. This is mainly because CT cannot be taught in a traditional manner and has to overcome both a pedagogical and a systemic challenge.

With regards to the pedagogical challenge, previous research has shown that even though CT has close connections to programming the first cannot be automatically learned through teaching the latter. Duncan summarized a large pilot study with primary school students in New Zealand with "We had hoped that Computational Thinking skills would be taught indirectly by teaching programming and other topics in computing, but from our initial observations this may not be the case" [4] Indeed, CT is more than just programming and teaching such a skill may require the use of specific tools, so called Computational Thinking (CT) Tools [16], that can assist the CT process, without the introduction of complicated and difficult programming. With regards to the systemic challenge, the question is about how, with Computer Science (CS) being up

until recently mainly an extracurricular activity, taken up by interested in the subject (self-selected) teachers and students, we can cross the chasm [5] of CSEd: shifting from self-selected teachers and students to ALL teachers and students.

Switzerland, a highly affluent, but in terms of K-12 CSEd somewhat conservative country, has made a bold movement which may lead to the successful crossing of the CSEd chasm (Fig. 1). With the introduction of Lehrplan 21, the new common curriculum for compulsory education in the 21 German-speaking states, CS will be introduced into the Swiss elementary schools starting from the first grade. In anticipation of the changes imposed by the new educational framework, Switzerland is radically shifting its strategy by introducing mandatory pre-service teacher CSEd starting at the elementary school level. The term pre-service teacher refers to the undergraduate students who study to become primary level teachers. Following from that, since September of 2017, the School of Education of Northwestern Switzerland (PH FHNW), requires its students (pre-service teachers), to take a mandatory CSEd course in order to be able to graduate and teach. In order to find out how effective the course was we conducted a study collecting both qualitative and quantitative data from more than 600 students that took the course. This paper presents the philosophy of the course and discusses initial findings from the study. Our results show that even though pre-service teachers can easily learn basic programming, the question about learning CT still remains open.

2 RELATED WORK

There has been a long definition debate about CT. Jeannette Wing defines CT as the thought process involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out [6] (Fig. 2). Despite the fact that many agree with this definition, to this day there is no unanimous agreement on what CT exactly is [7]. In 2012, Aho defined CT as the thinking process of developing solutions for problems using algorithms and computational steps [3]. The same year, the Royal Society defined CT as the process of recognition of "aspects of computation in the world that surrounds us" and the application of "tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes" (p.29) [8]. In the context of K-12 CT education, a similar definition was provided by Barr and Stepheson summarizing that "CT is an approach to solving problems in a way that can be implemented with a computer… . It is a problem-solving methodology that can be automated and transferred and applied across subjects" (p. 115) [9]. Denning, made a distinction between traditional and new CT and argued that to this day the CSEd community is struggling to answer three questions: What is CT? How can it be assessed? And is it good for everyone [2]?

Despite the ongoing debates surrounding CT's meaning, importance and way of teaching, commonly used definitions are gradually emerging and there is broad acknowledgement that our global economy is based and shaped by computing. There is enough literature about teaching and learning programming and CS but mostly for the undergraduate college level. In terms of teaching CT in K-12, programming does not necessarily translate in teaching CT [4]. Despite many countries introducing CSEd in their K-12 curricula, such as: UK's Computing at School movement [24], the partnership of the University of Adelaide with Google to successfully implement Australia's Digital Technologies Curriculum [25], and the CS for All mandate in the US [26], and other initiatives aiming at introducing CS into schools, such as the CS4HS and Computing in the Core, CSEd has not been systematically introduced to K-12 education [7] [27]. The crossing of the CSEd Chasm requires not only finding a place for CSEd into K-12 curricula, but also finding a place for CS in pre-service teachers' educational curricula. Currently most efforts are focused in providing professional development of in-service teachers [10]. With the exception of pilot projects involving selected pre-service teachers' CS training [11, 31], there is not much literature about CS pre-service teacher CSEd [19]. Even though the need to educate pre-service teachers has been previously addressed in literature [29, 30], most research involving pre-service teachers and CS is focusing in questions about attitudes towards the use of computers [12]. A momentum crucial for the future of CSEd is building up in Switzerland, which addresses both requirements. The new Lehrplan 21 introduces CSEd to the primary school level by way of the module "Medien und Informatik." The important next step is to find ways to successfully implement the module descriptions of Lehrplan 21 in practice and bring CSEd to primary schools in Switzerland.

To meet the Lehrplan 21 requirements, the school of education of northwestern Switzerland (PH FHNW) offers since September 2017 an obligatory course in CSEd for its students: pre-service teachers. We conducted research to investigate this underexplored but very important area in order to fill in the literature gap, contribute to the development of CSEd and inform attempts of crossing the CSEd chasm.

3 APPROACH

To illustrate the fundamental challenges for a strategy to cross the CSEd chasm, a brief history of about 30 years of research may help.

3.1 Crossing the CSEd Chasm

"Hard and boring" two words describing CS, especially used by young girls [13], became a strategic research map exploring the challenges of CSEd. Over the many years, this work revealed three distinct research stages relevant to crossing the CSEd chasm (Fig. 1).

Stage I: The "Friday Afternoon Computer Club" Stage (Self-Selected Students / Self-Selected Teachers): The first stage focused on creating more accessible programming support tools (AgentSheets) and more engaging creativity support tools for a Friday afternoon computer club [14].

Stage II: The "Professional Development Movement" Stage (All Students / Self-Selected Teachers): The transition from

stage I to stage II required a shift in focus from tools to curricula. Scalable Game Design (SGD), became a curriculum for teaching CT [15], while the 3D browser-based AgentCubes online CT tool was developed [16].

Stage III: The "Mandatory Pre-Service Professional Development" Stage (All Students / All Teachers): Through networks of excited CS teachers, compelling curricula such as SGD, can spread surprisingly quickly, but one cannot rely to such networks to persuade more conservative teachers to become part of the CSEd revolution, and cross the CSEd chasm. If CT is considered to be an essential 21st century skill, then the in-service teacher professional development must transform to mandatory pre-service teacher education. This shift is now, since September 2017, a reality for the PH FHNW, with the introduction of the obligatory SGD Course.

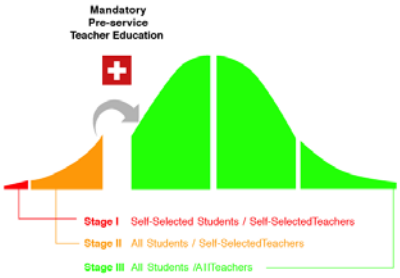

Figure 1: Crossing the Computer Science Education Chasm with mandatory pre-service teacher education

3.2 Scalable Game Design Course Philosophy

The Scalable Game Design (SGD) course is new and unique in Switzerland. It addresses all undergraduate students (pre-service primary teachers) of the PH FHNW, resulting in numerous challenges but also great opportunities. Most challenging for this course is the extraordinary heterogeneity regarding the background knowledge and interest of students. This course has to cope with a wide range of interests, ranging from intensive CS enthusiasm to complete CS apathy. In contrast to other subject areas such as mathematics, the course cannot assume that students have even basic CS knowledge from their previous education.

The goal of the SGD course is for pre-service teachers to become computational thinkers themselves and to be able to help their students become computational thinkers. The course does not require any prior knowledge in CS. Through the course the students are acquiring basic programming skills and CS knowledge, but also, they understand what constitutes creative processes and how these can be stimulated sustainably.

The definition of CT used for building this course was heavily influenced by the particularities and needs of the primary school teachers and their future students. In Switzerland, primary level teachers teach all subjects to their students who can be as young as 6 years old. An appropriate CT definition should address and respond to this reality. To that end, and for the purposes of this course, CT is a combination of mathematical -analytical thinking with natural sciences, engineering, and other disciplines. CT is conceptualized as thinking with the computer (Denken mit dem Computer in German) joining human abilities with computer affordances (Fig. 2). It is regarded and employed as a way of thinking that uses the computer as an instrument to support the human thought process. Based on Wing's [1] definition the CT process can be segmented into three stages:

1. Abstraction: problem formulation; ***2. Automation:*** representation of a solution; ***3. Analysis:*** execution and evaluation of the solution representation (Fig.2).

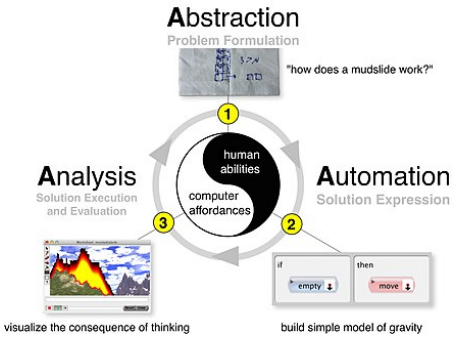

Figure 2: The AAA model: a three-stage process describing Computational Thinking

The course was conceptualized upon three pillars to combine CT with the Swiss Lehrplan 21.

1) Motivation and Learning Strategy: Scalable Game Design. Motivation is at the center of this course. The SGD pedagogy is based on the Zones of Proximal Flow theory and describes the creation of games and simulations and their continuous adaptation to new ideas [15, 23]. SGD starts with a project first approach and has a uniquely low threshold, so students can begin with easy activities (as described by Papert [17]), and a high ceiling so students can advance towards the creation of highly sophisticated games and simulations that model social and scientific phenomena in STEM. Through a set of increasingly sophisticated game and simulation projects the students' programming and computational thinking skills scale up gradually.

2) Tools designed specifically to teach and support CT at the elementary school level: Computational Thinking Tools. CT is not the same thing as programming. Unlike CT, each programming language is characterized by its own syntax, which is not as important as the conceptual understanding of general programming concepts. CT Tools are educational programming environments that make the teaching of CT practical on every school level. Good CT Tools address the two fundamental challenges of CS being perceived as difficult and boring [13] by having two key aspects. The first key aspect of CT Tools is that they minimize avoidable complexity and thus address syntactic, semantic and pragmatic challenges. The second aspect of CT Tools is that they support creativity at a high level [14]. By supporting all three stages of the CT process, minimizing

accidental complexity, and including elements that support the creative process, CT Tools render CT and programming accessible and exciting [18].

3) The 7 big ideas of Computer Science. The mapping of the SGD strategy onto the computer science part of the Lehrplan 21 was straightforward. The three main Lehrplan 21 CS topics (data, algorithms, and systems) were identified as a subset of the 7 big ideas found in the AP Computer Science Principles (CSP) framework: creativity, abstraction, data, algorithms, programming, the Internet and global impact [20].

The SGD course covered 14 weeks of classes (two-hour sessions per week). Each week's session was divided into two parts: a theory part and an activity part. The theory part provided knowledge and competencies regarding one of the seven CSP ideas. Each CSP idea was mapped onto a two-week block so that different aspects of each of the seven CSP ideas could be covered and its relevance to the professional function of primary school teachers could be demonstrated. For the activity part the students focused on creating their own projects using CT Tools. Students learned to program through a project first approach, supported by scaffolding methods and using mainly the CT Tool AgentCubes [18]. After being introduced to AgentCubes in their first session, the students started using it to program immediately. Initially they created simple Frogger like games using step-by-step instructions, but as the course progressed they gradually moved on to program more advanced games and STEM simulations while experimenting without the use of instructions. Other tools, such as Scratch and Processing were also introduced. In the second half of the course the activity part became even more self-guided since the students were working on their own final projects, which they had to develop in groups using CT Tools or object-oriented programming languages. Through this hands-on work using CT Tools, the students were able to learn how to write easy programs and understand how to think computationally. The instructors of the course had a combined background in computing and education. programs and understand how to think computationally.

4 METHODS

Since the SGD course is the first of its kind, the teaching was accompanied by a research project, which documented and collected data. The course was taken by approximately 650 students (pre-service teachers) mostly from the first and third semester. On September 17, 2017, in 4 states, 7 instructors began to teach 26 CS courses with approximately 25 pre-service teachers each. During this time a variety of methods from questionnaires to in-depth interviews were used to collect data. During the first semester of the course, we ran three questionnaires: one at the beginning, one in the middle and one at the end of the course. The questionnaires were consisted by background, self-efficacy and CS knowledge/skills questions and used a five-point Likert scale (1 = strongly disagree --- 5 = strongly agree). The middle questionnaire was focused more on the assessment of the tools used in the class, while the end questionnaire included a course evaluation part. In order to assess the skills and knowledge gained by the students during the course, we repeated a number of questions from the initial questionnaire in the final and calculated the effect sizes for these questions.

Additionally, for the purposes of teaching more than 650 students and coordinate seven instructors, the teaching team used a teaching/learning platform where each instructor could communicate with the students and the other instructors online. The platform contained the teaching material and homework assignments while it gave the possibility to develop wikis, groups, discussion forums etc. This gave us access to a plethora of qualitative data from projects to homework answers etc., and the ability to ask specific open-ended questions.

5 RESULTS

In the following sections we present in detail some of the findings from both the qualitative and quantitative data collected.

5.1 Quantitative Data

We present here the effect size, calculated using the Cohen's d formula [21], of one question with a particular interest to the teaching of CSEd. We asked the students to rate their ability to program in both the beginning and the end of the course. We received 539 and 471 answers respectively. In the beginning of the course 77.9% of the students rated their ability to program as practically non-existent (1 in Likert scale). This number drops to 5.3% at the end of the course. Cohen's d for this particular question was calculated to 2.05 which indicates a large effect size. The figure below displays the percentages of the answers before and after the course (Fig. 3).

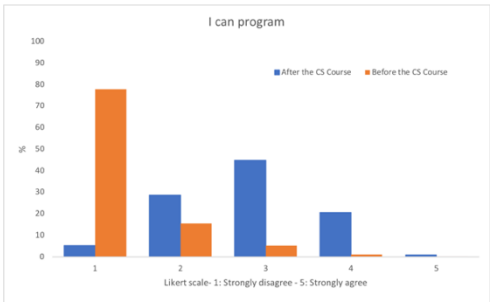

Figure 3 Percentages of the answer to the question: I know how to program, before and after the SGD course.

5.2 Qualitative Data

Given the central role of CT and the challenges connected to its definition and teaching, in October, after about one and a half months into the course we asked the students to answer the open-ended question: What is CT for you? We received 447 replies which we grouped according to seven major categories/themes. The themes were developed based on the answers given, while the responses were categorized as correct

Table 1: Results from the analysis of the open-ended question: What is CT for you? (N= 447)

Category & Description	Examples of Responses	%
a) Thinking like a computer: Answers suggesting that the computer is a thinking entity	"For me, computational thinking means understanding how a computer thinks, how to program it, how to abstract and simplify a problem, how to design solution strategies." "For me, "Computational Thinking" is how you can think like a computer. That you can think abstractly (way of thinking like a PC)"	21%
b) Programming CS/ Computer work: Answers strongly connecting CT to the ability to program or to apply computer science principles.	"Find solutions to a problem. Especially in the field of programming." "I think Computational Thinking has something to do with how the computer works." "Computational thinking is for me to deal with the world of computer science and learn more about the methods and models. Basically, it should help us to cope with our everyday problems easier."	16.1%
c) Problem division: Answers identifying CT as a process that involves breaking a problem into smaller less complicated parts.	"Computational thinking is for me a "system" for solving problems. It is divided into smaller steps to find an efficient solution. The problems are abstracted and suitable algorithms are searched for, so that the problem can be solved as quickly as possible." "Identify a problem, break it into sub-steps and formulate a solution strategy."	14.8%
d) Thinking with the Computer: Answers suggesting synergistic integration of human abilities with computer affordances	"Problem solving with the help of the computer. The keywords abstraction, automation and analysis come right to my mind" "Solution-oriented thinking and acting with computer support"	14.3%
e) Problem Solving: Answers identifying CT as being a problem-solving process, without much specification	"Computational thinking is for me the individual mindset of a person to answer or solve different, more or less complex, questions, problems and tasks." "For me, computational thinking means simply presenting a problem (abstractly) and thus finding a solution."	13.4%
f) CT Process/ AAA: Answers describing the CT process along the lines of the CT definition that was taught in the class	"To formulate a problem in that way so it can be solved by both a human and a computer." "Interaction between human and computer 1. problem formulation (abstraction) 2. Representation of a solution (automation) 3. Execution & Evaluation of the Solution (Analysis)"	11.4%
g) Other: Answers not matching any of the other categories	"Logical systematic thinking that only results in wrong or correct" "I do not remember exactly what the term "computational thinking" covers. When I hear it like this, it comes to my mind spontaneously that it is a process that is evolving."	9%

or wrong based on the definition of CT on which the course was based on. Table 1 presents the results of the analysis while it displays examples from answers from each theme.

25.7% (115 answers) of the students gave an answer that is along the lines of the CT definition that was taught in the class. From this, 14.3% (64 answers) were answers that identified CT as "thinking with the computer" (denken mit dem Computer) while 11.4% (51 answers) described a process along the lines of Wing's definition or the AAA. process (Fig.2). 14.8% (66 answers) described CT as a process of dividing a problem to smaller easier parts in order to solve it, while 13.4% (60 answers) referred to CT as a problem-solving process. Finally, 9% (40 answers) were classified as "other" clustering the remaining responses (Fig. 4).

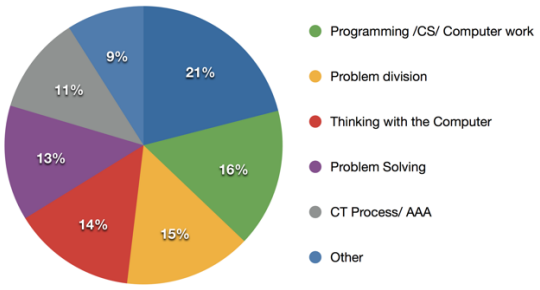

Figure 4 Categorized students' answers to the question "What is CT for you?"

6 DISCUSSION

The large effect size for the perception of programming skills suggests that the students (pre-service teachers) think that they successfully learned how to program but it is less clear what they actually learnt with respect to CT. The disparity between teaching programming and testing CT is consistent with Duncan's findings [4]. Even though the correctness of some of these answers is debatable we are only considering categories c, d, and f to be "right" in the sense that they were the intended learning outcomes of the course. Looking through this lens, only 40.5 % of the answers correct. However, the large percentage of false answers (59.5 %) does appear to suggest either instructional difficulties or tenacious misconceptions that are hard to change. The following discussion interprets some of the findings.

a) ***Thinking like a Computer (21%):*** We found this response surprising as the course tried to explicitly stress the notion of computational thinking as a synergistic combination of human abilities with computer affordances (Fig. 2 center). Do computers really think? Should people actually think the way computers operate, and, if so, at what level, e.g., machine language, would this kind of thinking take place?

b) ***Programming/CS/Computer work (16.1%):*** This category was perhaps too broad. It suggests that CT is an activity connected to programming, or to the application of CS principles. While this is true, it is not clear how inclusive these kinds of statements are. Is any form of programming or application of CS principles always necessarily a strong manifestation of CT? For the most part, these kinds of answers reflect our own blurred comprehensions of what CT is or is not.

c) ***Problem Division (14.8%):*** Divide and conquer is a well-known general problem-solving strategy. Most CT will include some kind of problem division but not every problem division is necessarily CT.

d) ***Thinking with a Computer (14.3%):*** This was the answer that we hoped to get but did not entirely succeed to convey. One very small part of the course explored the creation of STEM simulations. We can only speculate that the idea of thinking with a computer would perhaps have made more sense with simulation building than with game design.

e) ***Problem Solving (13.4%):*** While CT does include some problem-solving components, it is more than just problem solving. For instance, not every kind of problem solving includes the expression of programs.

f) ***The AAA/ CT Process (11.4%):*** Fig. 2 was our main learning prompt to capture the Abstraction /Automation /Analysis process described by Wing [1]. The relatively low frequency of this answer may suggest that the connection between theory and praxis was not sufficiently explicit.

g) ***Other (9%):*** This category was very wide ranging from actual admissions that pre-service primary level teachers actually did not know what CT is, to partial credit statements, e.g., mentioning the CT process model but with an incorrect sequence.

The same group of 600+ pre-service primary level teachers will have to take the mandatory Computer Science Didactics course next semester. Our findings suggest being more explicit in using and connecting CT with the practice of programming. One shift from the Introduction to Programming to Computer Science Didactics course will be the integration of CS with other disciplines such as STEM, art, music and languages. Seeing CT applied to other fields may help develop a better sense of purpose of CT. It really is not the case that we want teachers, or students, to think like computer scientists or programmers but to conceptualize CT as general-purpose thinking tool relevant not only to computer science but to most disciplines taught in public schools.

7 CONCLUSIONS

Schools of education in Switzerland are starting to make CSEd mandatory at the elementary school level. The data suggests that using Computational Thinking Tools, to create games and STEM simulations, it is possible to teach a basic understanding of programming to non-self-selected pre-service elementary school teachers. However, it is less clear, how much or what kind of CT is conveyed. There appear to be preconceived notions of computing that are difficult to overcome.

ACKNOWLEDGMENTS

This research was supported by the U.S. National Science Foundation, the Swiss National Science Foundation, and the Hasler Foundation.

REFERENCES

[1] Wing, J.M., *Computational Thinking*. Communications of the ACM, 2006. **49**(3): p. 33-35.
[2] Denning, P.J., *Remaining trouble spots with computational thinking*. Communications of the ACM, 2017. **60**(6): p. 33-39.
[3] Aho, A.V., *Computation and computational thinking*. The Computer Journal, 2012. **55**(7): p. 832-835.
[4] Duncan, C. and T. Bell, *A Pilot Computer Science and Programming Course for Primary School Students*, in *Proceedings of the Workshop in Primary and Secondary Computing Education*. 2015, ACM: London, United Kingdom. p. 39-48.
[5] Moore, G.A., *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*. 1999, New York, NY: HarperBusiness.
[6] Wing, J.M., *Computational Thinking Benefits Society*, in *40th Anniversary Blog of Social Issues in Computing*, J. DiMarco, Editor. 2014, University of Toronto: http://socialissues.cs.toronto.edu/2013/01/40th-anniversary/.
[7] Grover, S. and R. Pea, *Computational Thinking in K–12 A Review of the State of the Field*. Educational Researcher, 2013. **42**(1): p. 38-43.
[8] Furber, S., *Shut down or restart? The way forward for computing in UK schools*. The Royal Society, London, 2012.
[9] Barr, V. and C. Stephenson, *Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?* Acm Inroads, 2011. **2**(1): p. 48-54.
[10] Lamprou, A., A. Repenning, and N.A. Escherle, *The Solothurn Project — Bringing Computer Science Education to Primary Schools in Switzerland*, in *22nd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2017)*. 2017: Bologna, Italy.
[11] Huett, K.C. and M.A. Varga. *Building Pre-Service Teacher Interest in Computer Science Education through Mentoring Experiences*. in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016. ACM.
[12] Teo, T., *Pre-service teachers' attitudes towards computer use: A Singapore survey*. Australasian Journal of Educational Technology, 2008. **24**(4).
[13] *Women Who Choose Computer Science – What Really Matters, The Critical Role of Encouragement and Exposure*. 2014, Google: Google Report. p. 9.
[14] Repenning, A., *Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets*. Journal of Visual Languages and Sentient Systems, 2017. **3**(July): p. 68-89.
[15] Repenning, A., et al., *Scalable Game Design: A Strategy to Bring Systemic Computer Science Education to Schools through Game Design and Simulation Creation*. Transactions on Computing Education (TOCE), 2015. **15**(2): p.1-31.
[16] Repenning, A., A. Basawapatna, and N. Escherle, *Computational Thinking Tools*, in *IEEE Symposium on Visual Languages and Human-Centric Computing*. 2016, IEEE Press: Cambridge, UK.
[17] Papert, S., *Instructionism versus Constructionism*, in *The Children's Machine*. 1993, BasicBooks. p. 137-156.
[18] Repenning, A., et al., *AgentCubes: Enabling 3D Creativity by Addressing Cognitive and Affective Programming Challenges*, in *World Conference on Educational Media and Technology, EdMedia 2012*. 2012: Denver, Colorado, USA. p. 2762-2771.
[19] Döbeli Honegger, B. and M. Hielscher, *Vom Lehrplan zur LehrerInnenbildung - Erste Erfahrungen mit obligatorischer Informatikdidaktik für angehende Schweizer PrimarlehrerInnen*. Informatische Bildung zum Verstehen und Gestalten der digitalen Welt, 2017.
[20] CollegeBoard, *AP Computer Science Principles*. 2017, College Board.
[21] Cohen, J., *Statistical power analysis for behavioral sciences* 2nd ed. 1988, Hillsdale, NJ L. Erlbaum Associates.
[22] Papert, S., *An Exploration in the Space of Mathematics Educations*. International Journal of Computers for Mathematical Learning, 1996. **1**(1): p. 95-123.
[23] Basawapatna, A., et al., *The Zones of Proximal Flow: Guiding Students Through A Space Of Computational Thinking Skills and Challenges*, in *International Computing Education Research (ICER 2013)*. 2013, ACM Press.: San Diego, CA, USA.
[24] Crick, T. and S. Sentance, *Computing at school: stimulating computing education in the UK, in Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. 2011, ACM: Koli, Finland. p. 122-123.
[25] The University of Adelaide,. CSER Digital Technologies Education [accessed 20.03.2018]; Avalable from https://csermoocs.adelaide.edu.au/.
[26] Smith, M., *Computer Science For All*. 2016, The White House President Barack Obama: USA.
[27] Franklin, D., *Putting the computer science in computing education research*. Communications of the ACM, 2015. **58**(2): p. 34-36.
[28] Shuchi, G. and P. Roy, *Computational Thinking: A competency whose time has come.*, in *Computer Science Education: Perspectives on teaching and learning*, Sentance, S., Editors. 2018, Bloomsbury
[29] Darling-Hammond, L. and J. Bransford, *Preparing teachers for a changing world: What teachers should learn and be able to do*. 2007: John Wiley & Sons.
[30] Yadav, A., C. Stephenson, and H. Hong, *Computational thinking for teacher education*. Commun. ACM, 2017. **60**(4): p. 55-62.
[31] Yadav, A., et al., *Computational thinking in elementary and secondary teacher education*. ACM Transactions on Computing Education (TOCE), 2014. **14**(1): p. 5.