Computational thinking in compulsory education: Towards an agenda for research and practice

Voogt, J.M.; Fisser, P.; Good, J.; Mishra, P.; Yadav, A.

Published in:
Education and Information Technologies

DOI:
10.1007/s10639-015-9412-6

Educ Inf Technol (2015) 20:715–728
DOI 10.1007/s10639-015-9412-6

Computational thinking in compulsory education: Towards an agenda for research and practice

Joke Voogt[1] · **Petra Fisser**[2] · **Jon Good**[3] · **Punya Mishra**[3] · **Aman Yadav**[3]

Published online: 17 June 2015

Abstract Computational Thinking is considered a universal competence, which should be added to every child's analytical ability as a vital ingredient of their school learning. In this article we further elaborate on what Computational Thinking is and present examples of what needs to be taught and how. First we position Computational Thinking in Papert's work with LOGO. We then discuss challenges in defining Computational Thinking and discuss the core and peripheral aspects of a definition. After that we offer examples of how Computational Thinking can be addressed in both formal and informal educational settings. In the conclusion and discussion section an agenda for research and practice is presented.

Keywords Computational thinking · Education · Curriculum · Computer science · Problem solving · Programming

✉ Joke Voogt
j.m.voogt@uva.nl

Petra Fisser
p.fisser@slo.nl

Jon Good
goodjona@msu.edu

Punya Mishra
punya@msu.edu

Aman Yadav
ayadav@msu.edu

[1] University of Amsterdam/Windesheim University of Applied Sciences, P.O.Box 15776, 1001NG Amsterdam, The Netherlands

[2] National Institute for Curriculum Development, PO Box 2041, 7500 CA Enschede, The Netherlands

[3] Michigan State University, 620 Farm Lane, #509 A Erickson Hall East, Lansing, MI 48824, USA

1 Setting the stage

In March, 2006, Jeanette Wing published an influential article, "Computational Thinking," in the *Journal of the Association for Computing Machinery*. Wing posited that "computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science… It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (p. 33). Overall, the article argued that this new competency should be added to every child's analytical ability as a vital ingredient of their school learning. Several professional bodies and think tanks in for instance the US, the UK, and the Netherlands have called for more attention to computational thinking in the curriculum. At the EDUsummIT 2013, we aimed to advance this discussion by focusing on the core components of computational thinking, its relation with and distinction from other 21st century competences (Voogt et al. 2013), and its place in the curriculum.

This focus on computational thinking is not new. Its roots go back, most significantly, to Papert's work on the LOGO programming language and the idea of children manipulating the computer, which would allow them to develop procedural thinking through programming (Papert 1980, 1991). The recent advancement and availability of better computing tools and mobile technologies has led to a resurgence in interest in computational thinking. Recent work in the field builds upon Papert's pioneering work but has a distinct 21st century flavor with its focus on internet, gaming, big data and creativity.

Despite this renewed interest in computational thinking there is a range of issues and challenges facing us as we attempt to integrate computational thinking in the curriculum. They range from foundational issues such as defining what we mean when we speak of computational thinking, to what the core concepts/attributes are and their relationship to programming knowledge; how computational thinking can be integrated into the curriculum; and the kind of research that needs to be done to further the computational thinking agenda in education.

2 Computational thinking: Research history and implications

The concepts of Computational Thinking (CT) and the practice of programming are difficult to delineate in the literature because many CT studies or discussions of theory use programming as their context (Fletcher and Lu 2009; Hambrusch et al. 2009; Lee et al. 2011). This can be confusing to the reader and often lead to the impression that CT is the same as programming or at the very least that CT requires the use of programming. To further add to this confusion, there is a history in both research and popular press of the use programming as a way to develop thinking skills. CT focuses on developing these thinking skills while within subjects beyond computer science. CT does not necessarily require the use of programming nor are CT scholars making the claim that programming has to be the context in which these skills are developed. To understand why CT has taken this alternate route to developing thinking skills, we must examine the history of research regarding programming and thinking skills.

As George Santayana warned us "Those who cannot remember the past are condemned to repeat it" (1905, p. 284).

In searching for a history of research to inform this new area, the research related to Seymour Papert's work with the LOGO programming language immediately comes to mind. While Papert referred to computational thinking in his book "Mindstorms: Children, Computers, and Powerful Ideas" (1980) and other writings (Papert 1991), it is unclear whether he was referring to the same construct that Wing described in 2006. The portion of his work we are interested in is on the relationship between programming and thinking skills. In "Mindstorms" he theorized that the learning and application of LOGO programming would have an effect on students' learning and concepts of knowledge across multiple disciplines. The work with LOGO focused more on the explicit learning of programming, while the current focus of computational thinking involves using the general concepts born out of Computer Science (CS). To some observers these separate avenues may be of only the slightest distinction. The study of CS sooner or later requires knowledge of programming, and it is through CS that the core concepts of CT were developed.

Papert (1980) posited that students, through the creation of microworlds using the programming language LOGO, would develop into apprentice epistemologists. Through thinking about programming, students would become adept at thinking about thinking; hence, Papert thought that this use of technology would develop skills in students that would transfer to non-programming contexts both within and outside of the classroom. However, results from the studies surrounding LOGO were decidedly inconclusive. Pea et al. (1985), created a task, planning classroom chores, that was carefully designed to provide an opportunity for "near" transfer from the tasks they had done with LOGO. The treatment group had received instruction in LOGO for over a year. Their team did not find evidence that students with LOGO experience had better planning skills for a non-programming task than their peers with no LOGO experience. Conversely, Klahr and Carver (1988) found that students instructed in debugging skills while using LOGO did experience transfer of these skills to a non-programming environment. In trying to rectify problems with setting a table or planning a travel route, students with experience in LOGO would search for 'bugs' in their process in a more effective way than non-LOGO students. These effects persisted beyond the instance of LOGO instruction, with results still showing up 4 months after instruction had ended. These two studies are only two examples among many with conflicting results.

Salomon and Perkins (1989) offered a possible explanation for these differences in findings through examining high-road and low-road transfer. Low-road transfer encompassed skills that are practiced repeatedly, with the amount of transfer being dependent on the amount of repetition and the number of contexts in which it is practiced. High-road transfer encompasses mindful abstraction of the concept or process being learned. High-road transfer is beyond the reflexive nature of low-road transfer and requires reflection on the knowledge and opportunities for transfer. The authors posited that the amount of experience and practice needed for low-road transfer of programming for non-programming contexts was beyond the restrictions of a young child's school environment. High-road transfer, though, could be accomplished through careful instruction on how to apply programming skills beyond their original context. Thus, the high-road transfer is heavily dependent on the instructional practices of the

educator, which leads to question whether it is the programming experience that makes the difference or the methodology of the instructors. In short, programming alone will not likely produce results in students' learning, and it is possible that programming is only one context for the application of skills taught in an effective manner.

These findings from Papert's work with LOGO are what connect to our conceptualization of CT as the teaching of thinking skills applied to multiple domains. We can see echoes of Papert's original theories in the push for programming instruction to teach mental skills. We see initiatives such as the Hour of Code (see http://hourofcode.com/) making claims that learning to program will lead to improved problem solving, critical thinking skills, and academic outcomes. In a sense, CT is taking this existing framework of "program to learn mental skills" and turning it on its head. If the goal is to learn general mental skills that apply to multiple domains, CT is asking: Why not learn those skills explicitly within multiple domains? This aligns with the critiques by Salomon and Perkins (1989) in that the instructional methods and their focus on the general skills is the more effective strategy for ensuring transfer, not a focus on computer programming itself. Computer programming may indeed still be one of the contexts we work within, yet it should not be the only one.

For the researcher and educator interested in CT this review of the past research offers some guidance. Programming, Computer Science, and Computational Thinking are not equivalent concepts, yet are intertwined. Programming is but one context for the practice of Computer Science and Computational Thinking. Computer Science is the field and practice from which Computational Thinking skills arose, however is not the only discipline in which these skills can be found or applied. To focus on programming as the primary instantiation of CT, and to expect programming alone to result in more refined thinking, similar to the early research on students' use of LOGO, would be a mistake both conceptually and pedagogically. The focus should be on the higher-level concepts being learned and the multiple domains in which they can be applied.

3 Challenges in defining computational thinking

3.1 The challenges of defining CT

A major challenge in trying to define Computational Thinking (or anything else for that matter) is how we think of the idea of definition. There is an inherent tension in attempting to define CT and that has to do with the thinking of the "core" qualities of CT versus certain more "peripheral" qualities. This approach seeks to identify or otherwise specify a set of necessary and sufficient conditions that need to be met for a set of practices or cognitive processes to be regarded as CT. This approach to defining in terms of necessary and sufficient conditions has a long history in philosophy, going back to the days of Aristotle. However, we believe that most constructs (CT included) cannot be defined in such a manner–mainly because it is difficult to implement in practice.

Consider, for instance the Computer Science Teacher Association (CSTA) who argues for a number of dispositions or attitudes as being essential dimensions of CT (ISTE & CSTA 2011). Some of these dimensions include attributes such as a confidence in dealing with complexity, persistence in working with difficult problems,

having a tolerance for ambiguity in dealing with open-ended problems, and the ability to work in collaborative groups towards a common goal. Though we agree that these are important dispositions to consider for CT, we also believe that including a broad list of this nature runs the risk of diluting the idea of CT, blurring and making it indistinct from other 21st century skills (e.g., media information literacy). Furthermore if we deem these additional attributes essential, we may be emphasizing the logical definition (in terms of necessary and sufficient conditions) rather than the more pragmatic approach that focuses on the manner in which the concepts are actually used.

In contrast, we suggest a flexible and more pragmatic approach towards defining concepts based on current cognitive psychology. According to this approach (also known as prototype theory), understanding a concept does not require developing a series of necessary and sufficient conditions that need to be met. In contrast we seek to develop a more graded notion of categories with an emphasis on the *possible* rather than the *necessary.* This is consistent with work by Rosch (1978) and Lakoff (1987) as it is with Wittgenstein's discussion of the category *game* in his 1953 work *Philosophical Investigations (p. 66),* where he wrote:

> Consider for example the proceedings that we call 'games'. I mean board games, card games, ball games, Olympic games, and so on. What is common to them all? Don't say, "There must be something common, or they would not be called 'games' "–but look and see whether there is anything common to all. For if you look at them you will not see something common to all, but similarities, relationships, and a whole series of them at that. To repeat: don't think, but look! Look for example at board games, with their multifarious relationships. Now pass to card games; here you find many correspondences with the first group, but many common features drop out, and others appear. When we pass next to ball games, much that is common is retained, but much is lost. Are they all 'amusing'? Compare chess with noughts and crosses. Or is there always winning and losing, or competition between players? Think of patience. In ball games there is winning and losing; but when a child throws his ball at the wall and catches it again, this feature has disappeared. Look at the parts played by skill and luck; and at the difference between skill in chess and skill in tennis. Think now of games like ring-a-ring-a-roses; here is the element of amusement, but how many other characteristic features have disappeared! And we can go through the many, many other groups of games in the same way; can see how similarities crop up and disappear. And the result of this examination is: we see a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail (Wittgenstein 1953, p. 66).

So in our work to better understand CT we seek "similarities and relationships" that are "overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail" along the line suggested by Wittgenstein above. What this means for our task is that we seek not to find "necessary and sufficient conditions" but rather take on a broader philosophical perspective that connects with current thinking in the cognitive science—and in doing so provides us with a way of including these "peripheral skills" in our thinking about CT, but not seeing them as being essential to CT. We will make

an attempt to do so in our next sections in which we give an overview of the integration of CT in education.

3.2 Definitions of CT

Despite the challenges mentioned above, a number of scholars, primarily computer science educators, have attempted to define CT. We therefore first examine the discussion about the definition and core concepts of CT in the Computer Science domain.

Wing (2008) argued that CT complements thinking in mathematics and engineering with a focus on designing systems that help to solve complex problems humans face (Wing 2008; Lu and Fletscher 2009). The core CT concepts include, abstractions (the mental tools of computing, necessary to solve the problem), layers (problems need to be solved on different levels) and relationships between layers and abstractions (Wing 2008). The idea of abstraction and students' ability to deal with different levels of abstractions, as well as to think algorithmically and to understand the consequences of scale (big data), are fundamental to CT (Denning 2009; Lu and Fletscher 2009). Aho (2012) further argued that CT involves "thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (p. 832). Denning (2009) argued CT has a long history in computer science dating back to 1950s when it was known as algorithmic thinking meaning "a mental orientation to formulating problems as conversions of some input to an output and looking for algorithms to perform the conversions" (p. 28). Some computer science educators have also argued that programming is not essential in the teaching of computational thinking (e.g. Yadav et al. 2011; Lu and Fletscher 2009). Lu and Fletscher (2009) even suggested that an emphasis on programming might deter students from becoming interested in computer science. In sum, computational thinking is a conceptual way to "systematically, correctly, and efficiently process information and tasks" to solve complex problems (Lu & Fletcher, p. 261).

Many in the field of education, in particular educational technology, agree with computer science education community that CT is an important 21st century skill. Based on the definitions and core concepts of CT as provided by computer scientists, several definitions have emerged for what CT is in the domain of compulsory schooling (K-12). Key in all these definitions is the focus on the skills, habits and dispositions needed to solve complex problems (e.g. Barr and Stephenson 2011; Grover and Pea 2013; Lee et al. 2011; Sengupta et al. 2013; Wolz et al. 2011) with the help of computing (Wolz et al. 2011) and computers (Grover and Pea 2013; Lee et al. 2011). CT encompasses being able to distinguish several levels of abstraction and apply mathematical reasoning and design-based thinking (Sengupta et al. 2013). Mishra and Yadav (2013) have argued that CT goes beyond typical human computer interactions; instead, they argued that human creativity can be augmented by computational thinking, in particular with the use of automation and algorithmic thinking. Specifically, Mishra and Yadav suggested that computational thinking could move students from being consumers of technology to create new forms of expression build tools and foster creativity.

4 CT in education

4.1 Rationales for CT in education

Wing's (2008) call for computational thinking as a universal attitude and skill set that should be part of the repertoire of every child comes from the field of computer science, but it is an important competency and influences nearly all disciplines. Along the same lines, Bundy (2007) posited that the ability to think computationally is essential to conceptual understanding in every field, through the processes of problem solving and algorithmic thinking. CT is seen as an important competency because today's students will not only get to work in fields influenced by computing, but also need to deal with computing in their everyday life and in today's global economy (Barr and Stephenson 2011; Grover and Pea 2013). At the same time, however, computer scientists discuss the need to increase students' interest in computer science through paying attention to CT (and not programming) in the compulsory curriculum to offer students' the option to continue their further studies or their career in fields related to Computer Science (Wing 2006; Lu and Fletscher 2009; Wolz et al. 2011)

4.2 What should be taught?

While we agree that CT should add to every child's repertoire of thinking abilities, the question remains how and when children should learn this ability and how it should be taught. Wing (2008) indicated that CT should not only be part of undergraduate curricula at universities, but it should also be integrated at the elementary through high school levels of education. She argued that 'if we want to ensure a common and solid basis of understanding and applying CT for all, then this learning should best be done in the early years of childhood' (p. 3720). The question therefore remains what CT concepts should be learned in the different stages of schooling across the spectrum of subjects. Wing (2008) suggests that CT could be used to reinforce concepts that are already taught using visualizations and animations that explain abstract concepts, even at early grades. Barr and Stephenson (2011) described core computational thinking concepts and capabilities that could be embedded in K-12 classrooms. They suggested nine core concepts for CT in K-12 education: data collection, data analysis, data representation, problem decomposition, abstraction, algorithm and procedures, automation, parallelization and simulation. For example, the CT skill to be able to process and analyze data to create artifacts, could be implemented in language arts as well as STEM classrooms (see Barr & Stephenson for specific examples). Similarly, Grover and Pea (2013) see CT as a problem solving process with the following characteristics: 1. formulating problems in a way that enables us to use a computer and other tools to help solve them. 2. logically organizing and analyzing data; 3. representing data through abstractions such as models and simulations; 4. automating solutions through algorithmic thinking (a series of ordered steps); 5. identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; and 6.generalizing and transferring this problem solving process to a wide variety of problems. These operationalizations of CT clearly separate the cognitive activity of CT from the action of merely working on a computer or other digital device, such as word processing and/or creating webpages.

Finally, any discussion of CT must factor in the issue of human knowledge, expertise, and intuition. As a leader in big data analysis recently suggested, success in using CT requires knowledge not just of computer science and mathematics, but also imaginative capacities like innovative thinking and "a deep, wide ranging curiosity" (quoted in Lohr 2012).

Currently there are some examples of curricular frameworks that incorporate CT. One of these is a computational thinking framework for a Computer Science Principles course for high schools in the USA, designed by the College Board, presented in Table 1. This framework was developed upon deliberations in two US National Research Council workshops (2010, 2011) and considerable discussion by various professional societies, the US College Board and the US National Science Foundation.

The CS Principles framework above highlights the tension about what defines computational thinking. For example, the skill of collaboration is hardly unique to CT, and examples such as "collaborating with another student to produce an artifact" could equally well apply to engineering. As another illustration, competencies such as "select appropriate techniques to develop a computational artifact" and "use appropriate algorithmic and information-management principles" beg the issue on specifying what thinking skills are involved (Dede et al. 2013). Furthermore, given that this curriculum includes creativity as one of the seven big ideas, it is important to consider how CT can help students develop their creativity. It has been argued that CT can foster creativity, as one important 21st century competency, by enabling students not only to be consumers of technology, but also to build tools that can have significant impact on society (Mishra and Yadav 2013). The core argument for including creativity in this mix is that computing not only extends traditional forms of human expression, but also allows the creation of new forms of expression (The College Board 2012). Hence, it becomes important to consider where CT belongs in the K-12 curriculum; should it be as a computing subject on its own or should we embed CT across other subject areas.

4.3 Positioning CT in the curriculum

4.3.1 Computing as a separate subject

An important issue in integrating computational thinking in the curriculum is delineating its boundaries with other disciplines and the other 21st century competences (Voogt et al. 2013). Some (e.g. Royal Society 2012) position CT in the Computing curriculum stating, "Every child should have the opportunity to learn concepts and principles from Computing (including Computer Science and Information Technology) from the beginning of primary education onwards, and by age 14 should be able to choose to study towards a recognized qualification in these areas" (p. 44). Based on the report of the Royal Society, the UK has commenced the implementation of a new curriculum, an important part of which concerns computational thinking. An elaboration of the UK curriculum on computing for 4–7 and 14–16 year olds is provided in Table 2.

4.3.2 CT in cross-curricular practices

Wing (2006) called to isolate the major skills and mental frameworks learned within Computer Science (CS) and to teach them explicitly within other contexts. The need to

Table 1 Example of a draft curriculum framework for CT

In the Advanced Placement Computer Science Principles Draft Curriculum Framework, six computational thinking practices are identified (The College Board, 2013, pp. 7–8):

Computational Thinking Practices

P1: Connecting computing

Developments in computing have far-reaching effects on society and have led to significant innovations. These developments have implications for individuals, society, commercial markets, and innovation. Students in this course study these effects and connections, and they learn to draw connections between different computing concepts. Students are expected to:

- Identify impacts of computing;
- Describe connections between people and computing; and
- Explain connections between computing concepts.

P2: Developing computational artifacts

Computing is a creative discipline in which the creation takes many forms, ranging from remixing digital music and generating animations to developing websites, writing programs, and more. Students in this course engage in the creative aspects of computing by designing and developing interesting computational artifacts, as well as by applying computing techniques to creatively solve problems. Students are expected to:

- Create an artifact with a practical, personal, or societal intent;
- Select appropriate techniques to develop a computational artifact; and
- Use appropriate algorithmic and information-management principles.

P3: Abstracting

Computational thinking requires understanding and applying abstraction at multiple levels ranging from privacy in social networking applications, to logic gates and bits, to the human genome project, and more. Students in this course use abstraction to develop models and simulations of natural and artificial phenomena, use them to make predictions about the world, and analyze their efficacy and validity. Students are expected to:

- Explain how data, information, or knowledge are represented for computational use;
- Explain how abstractions are used in computation or modeling;
- Identify abstractions; and
- Describe modeling in a computational context.

P4: Analyzing problems and artifacts

The results and artifacts of computation, and the computational techniques and strategies that generate them, can be understood both intrinsically for what they are as well as for what they produce. They can also be analyzed and evaluated by applying aesthetic, mathematical, pragmatic, and other criteria. Students in this course design and produce solutions, models, and artifacts, and they evaluate and analyze their own computational work as well as the computational work that others have produced. Students are expected to:

- Evaluate a proposed solution to a problem;
- Locate and correct errors;
- Explain how an artifact functions; and
- Justify appropriateness and correctness.

P5: Communicating

Students in this course describe computation and the impact of technology and computation, explain and justify the design and appropriateness of their computational choices, and analyze and describe both computational artifacts and the results or behaviors of such artifacts. Communication includes written and oral descriptions supported by graphs, visualizations, and computational analysis. Students are expected to:

- Explain the meaning of a result in context;

Table 1 (continued)

- Describe computation with accurate and precise language, notation, or visualizations; and
- Summarize the purpose of a computational artifact.

P6: Collaborating

Innovation can occur when people work together or independently. People working collaboratively can often achieve more than individuals working alone. Students in this course collaborate in a number of activities, including investigation of questions using data sets and in the production of computational artifacts. Students are expected to:

- Collaborate with another student in solving a computational problem;
- Collaborate with another student in producing an artifact; and
- Collaborate at a large scale.

link CT to other subjects than computer science alone has also been argued by others (e.g., Hemmendinger 2010). He argued that the goal of teaching CT is not for everyone to think like a computer scientist, but instead, "it is to teach them how to think like an economist, a physicist, an artist, and to understand how to use computation to solve

Table 2 Aims of the National curriculum for computing in the UK; attainment targets for key stage 1 and 4 (Department for Education 2013)

The National Curriculum for computing at the UK targets students in the age of 5–16.

Its overall aims are:

1. Students can understand and apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation;
2. Students can analyse problems in computational terms, and have repeated practical experience of writing computer programs in order to solve such problems;
3. Students can evaluate and apply information technology, including new or unfamiliar technologies, analytically to solve problems;
4. Students are responsible, competent, confident and creative users of information and communication technology.

At key stage 1 (5–7 year olds) students should be taught to:

- Understand what algorithms are; how they are implemented as programs on digital devices; and that programs execute by following precise and unambiguous instructions
- Create and debug simple programs
- Use logical reasoning to predict the behaviour of simple programs
- Use technology purposefully to create, organise, store, manipulate and retrieve digital content
- Recognise common uses of information technology beyond school
- Use technology safely and respectfully, keeping personal information private; identify where to go for help and support when they have concerns about content or contact on the internet or other online technologies.

At the end of the age range, in key stage 4 (14–16 year olds) all pupils must have the opportunity to study aspects of information technology and computer science at sufficient depth to allow them to progress to higher levels of study or to a professional career.

All pupils should be taught to:

- Develop their capability, creativity and knowledge in computer science, digital media and information technology;
- Develop and apply their analytic, problem-solving, design, and computational thinking skills
- Understand how changes in technology affect safety, including new ways to protect their online privacy and identity, and how to identify and report a range of concerns.

their problems, to create, and to discover new questions that can fruitfully be explored" (p. 6). It involves developing ways of thinking that allow learners to use computational tools in creative ways within the disciplines. Thus, delineation of CT's boundaries with other fields is clearly an area that needs greater attention.

A number of researchers and educators have provided frameworks and examples for incorporating computational thinking across different subject areas. Barr and Stephenson (2011) described how concepts and skills in computers science, math, science, social studies and language arts can be referred to the core concepts of CT as described earlier in this paper. According to Barr and Stephenson (2011), the integration of CT across disciplines would lead to increased use of computational vocabulary and the acceptance of failed solution attempts. In addition, they advocated the use of teamwork to practice important aspects of CT: decomposition, abstraction, negotiation and consensus building. Lu and Fletcher (2009) illustrated how CT language and notation can be used in math (e.g. learning multiplication, charting information, finding square roots), social studies (understanding the assembly line), language arts (learning grammar), group projects in science and interdisciplinary subjects. Sengupta et al. (2013) developed two educational units for middle school students in which scientific inquiry, algorithm design and engineering are applied in an agent-based programming environment for students to design, evaluate and refine scientific models on kinematics and ecology. In another middle school example, Wolz et al. (2011) demonstrated the integration of CT in a journalism course for students. Computational thinking has been implemented with students in elementary schools too. An example on how CT can be fostered in young children is provided by Bers et al. (2014), who developed a robotics curriculum for kindergarten. In this curriculum CT concepts were linked to mathematics and literacy concepts.

4.4 CT in informal learning

Fishman and Dede (in preparation) even question the need to situate CT within school subjects. They relate CT to the larger context of learners informally engage in as makers and creators (including Scratch programming, Do It Yourself digital textiles, and robotics competitions). Wing (2008) indicated that most children today are not afraid to explore and play with new concepts and tools and that we should explore not only formal learning, but also informal learning settings, as learning takes also place outside the classroom: 'children teach each other; learn from parents and family; learn at home, in museums and in libraries; and learn through hobbies, surfing the Web and life experiences' (p. 3721). A couple of examples of such settings for learning CT come from Lee et al. (2011), who developed CT by providing CT rich environments in the context of modeling and simulation, robotics and game design using a use–modify-create pattern of student engagement.

5 Conclusion and discussion

The idea of Computational Thinking as an ability, a skillset, that every child should possess has emerged since 2006 and has been gaining attention and importance ever since. Computational Thinking draws on concepts of Computer Science, but the two

are not identical. While Computer Science is an academic discipline in his own right that studies computers and computational systems, Computational Thinking refers to thought processes that are involved when solving complex problems and generalizing and transferring this problem solving process to a wide variety of problems. The relationship between computational thinking and computers science is well expressed in the report of the Royal Society (2012, p.29), Computational Thinking is "the process of recognising aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes". Our review of the literature showed that there is consensus that Computational Thinking is much more than programming. However, programming is an important tool to help develop Computational Thinking skills, as expressed by Mitch Resnick, "computational thinking is more than programming, but only in the same way that language literacy is more than writing. They are both very important. Yes, it's more, but don't minimize programming just because it's more… programming, like writing, is a means of expression and an entry point for developing new ways of thinking" (NRC 2010, p. 13).

There is no clear-cut definition for CT and the main tension in the attempt to define CT has to do with defining the "core" competencies of CT versus the more "peripheral" competencies . We argue that for the purpose of conceptualizing CT and integrating it in education, we should not try to give an ultimate definition of CT, but rather try to find similarities and relationships in the discussions about CT. Finding these similarities and relationships will lead to a more concise description of "what matters" in CT and how to integrate it within K-12.

Some work on the integration of Computational Thinking in the curriculum has been done, resulting in curriculum frameworks, mainly the CS Principles in the United States and the National Curriculum in England. However, much more needs to be done. A major issue for the implementation of CT in educational practice is careful preparation of teachers to implement CT in their teaching practices. For Computer Science teachers this implies learning how to link computer science concepts core to computational thinking, as discussed above, to other curriculum domains. Teachers from other domains need to become acquainted with the core concepts of computational thinking. Yadav, et al. (2011, 2014) incorporated computational thinking for teacher education students with no prior experience in computer science highlighting how computational thinking ideas can be used in daily life. Computational Thinking concepts (such as problem identification and decomposition, abstraction, logical thinking, algorithms, and debugging) were illustrated with concrete examples from day-to-day life to relate the terminologies to the pre-service teachers' personal experiences. These daily life scenarios helped students find the concepts to be personally meaningful and helped them discover the ubiquitous nature of CT in everyday life.

Research on the integration of Computational Thinking in education is still scarce. In particular there is a need to study how CT can be developed in students in disciplines other than Computer Science. Further, the claim that developing CT also increases students' ability to be able to deal with complexity and open-ended problems needs to be studied in-depth. And finally, more in-depth knowledge is needed to study the role of programming as a tool in developing further CT.

References

Aho, A. A. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Barr, V., & 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*(1), 48–54.

Bers, M. U., Flanner, E. L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education, 72*, 145–157.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67–69.

Dede, C., Mishra, P., & Voogt, J. (2013). Working group 6: Advancing computational thinking in 21st century learning. http://www.curtin.edu.au/edusummit/local/docs/Advancing_computational_thinking_in_21st_century_learning.pdf. Accessed 28 Feb 2015.

Denning, P. J. (2009). The profession of IT. Beyond computational thinking. Communications of the ACM,52, 8,28–30.

Department for Education (2013). Statutory Guidance. National curriculum in England: Computing programmes of study. Retrieved from: https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study.

Fishman, B., & Dede, C. (in preparation). Teaching and technology: New tools for new times. In D. Gitomer & C. Bell (Eds.), *Handbook of Research on Teaching, 5th Edition* (American Educational Research Association). New York, NY: Springer.

Fletcher, G. H., & Lu, J. J. (2009). Education: Human Computing Skills: Rethinking the K-12 Experience. *Association for Computing Machinery. Communications of the ACM*, *52*(2). Retrieved from http://search.proquest.com.proxy1.cl.msu.edu/docview/237069669/14092797E84FBDF89ED/6?accountid=12598#.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin, 41*(1), 183–187.

Hemmendinger, D. (2010). A please for modesty. *ACM Inroads, 1*(2), 4–7.

ISTE & CSTA (2011). Computational thinking. Teacher resources. http://csta.acm.org/Curriculum/sub/CurrFiles/472.11CTTeacherResources_2ed-SP-vF.pdf.

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology, 20*(3), 362–404. doi:10.1016/0010-0285(88)90004-7.

Lakoff, G. (1987). *Women, fire, and dangerous things: What categories reveal about the mind*. Chicago: University of Chicago Press.

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), 33–37.

Lohr, S. (2012). Big Data is great, but don't forget intuition. *The New York Times*. Retrieved from http://www.nytimes.com/2012/12/30/technology/big-data-is-great-but-dont-forget-intuition.html.

Lu, J.J., & Fletscher, G.H.L. (2009). *Thinking About Computational Thinking*. SIGSE'09, March 3–7. Chattanooga, Tenessee, USA

Mishra, P., & Yadav, A. (2013). Of art and algorithms: Rethinking technology & creativity in the 21st century. *TechTrends, 57*(3), 11.

National Research Council. (2010). Committee for the Workshops on Computational Thinking: *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: National Academies Press. http://www.nap.edu/catalog.php?record_id=12840.

National Research Council. (2011). Committee for the Workshops on Computational Thinking: *Report of a workshop of pedagogical aspects of computational thinking*. Washington, DC: National Academies Press. http://www.nap.edu/catalog.php?record_id=13170.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Papert, S. (1991). Situating constructionism. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 1–11). Norwood: Ablex.

Pea, R. D., Kurland, D. M., & Hawkins, J. (1985). Logo and the development of thinking skills. *Children and Microcomputers: Research on the Newest Medium*, 193–317.

Rosch, E. (1978). Principles of Categorization. In E. Rosch & B. B. Lloyd (Eds.), *Cognition and Categorization* (pp. 27–48). Hillsdale, N.J. : New York: L. Erlbaum Associates ; distributed by Halsted Press.

Salomon, G., & Perkins, D. N. (1989). Rocky roads to transfer: Rethinking mechanisms of a neglected phenomenon. *Educational Psychologist, 24*(2), 113–142. doi:10.1207/s15326985ep2402_1.

Santayana, G. (1905). *The Life of Reason: Reason In Common Sense*. Retrieved from http://www.gutenberg.org/files/15000/15000-h/vol1.html#endofv1.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*, 351–380.

The College Board (2012). Computational thinking practices and big ideas, key concepts, and supporting concepts. Retrieved from http://www.csprinciples.org/home/about-the-project.

The College Board. (2013). AP computer science principles draft curriculum framework. New York, NY: College Board. Retrieved from http://www.csprinciples.org/home/about-the-project/docs/csp-cf-2013.pdf?attredirects=0&d=1.

The Royal Society (2012). *Shut down or restart? The way forward for computing in UK schools*. London: The Royal Society. http://royalsociety.org/uploadedFiles/Royal_Society_Content/education/policy/computing-in-schools/2012-01-12-Computing-in-Schools.pdf.

Voogt, J., Erstad, E., Dede, C., & Mishra, P. (2013). Challenges to Learning and Schooling in the Digital Networked World of the 21st Century. *Journal of Computer Assisted Learning, 29*(5), 403–413.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. doi:10.1145/1118178.1118215.

Wing, J. (2008). Computational Thinking and thinking about computing, Philosophical Transactions of the Royal Societ, 366. doi: 10.1098/rsta.2008.0118, published 28 October 2008.

Wittgenstein, L. (1953). *Philosophical investigations*. London: Wiley John & Sons Ltd.

Wolz, U., Stone, M., Pearson, K., Pulimood, S., & Switzer, M. (2011). Computational thinking and expository writing in the middle school, *ACM Transactions on Computing Education, 11*, 2, article 9.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). *Introducing computational thinking in education courses*. In Proceedings of ACM Special Interest Group on Computer Science Education, Dallas, TX.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education, 14*(1), 1–16. doi:10.1145/2576872.