An Overview of Computational Thinking

Sibel Kılıçarslan Cansu, PhD [1]
Fatih Kürşat Cansu[2]

[1] Abant İzzet Baysal University Faculty of Natural Sciences

[2] Bahçeşehir University Institute of Educational Sciences

DOI: 10.21585/ijcses.v3i1.53

Abstract

Computers and smart devices have become ubiquitous staples of our lives. Computers and computer-controlled devices are used in all industries from medicine to engineering, and textile production. One field where computers have inevitably spread into is education, and one pre-requisite of controlling computers, or increasing the level and efficiency of our control over them, is making human-computer interaction as efficient as possible. This process of efficient and effective computer use, known as "Computer-like Thinking" or "Computational Thinking", is seen as a field with the potential to support individual and societal development in our rapidly progressing world and to provide significant economic benefits. The fundamental concepts and scope of this field have been delineated in diverse manners by different researchers. Similarly, researchers have also advanced distinct critical viewpoints towards and potential benefits of computational thinking. This study aims to first define the concept of computational thinking by referencing source literature, then analyze the aims of certain criticisms of the field, and discuss the fundamental elements of computational thinking and contemporary research on these elements.

Keywords: computational thinking, computer-like thinking, computational-informatic thinking

1. Introduction

"Computer" as a word references a device that "computes", localized into Turkish as "bilgisayar" by Prof. Dr. Aydın Köksal (Keser,2011: p.88). Yet it is difficult to claim the same about "computational thinking", which is localized in a number of ways by researchers. Özden et al. (2015) use "*bilgisayarca düşünme",* whereas Yesan, Özçınar and Tanyeri (2017) prefer "*hesaplamalı düşünme".* Çınar and Tüzün (2017), meanwhile, used "*bilgi sayımsal düşünme"* and "*bilgi işlemsel düşünme"* in their paper. This study will primarily use "*bilgi işlemsel* düşünme" (Computational Thinking). The presence of such diverse localization attempts is natural. As Piaget has (Bringuier, 1980: p.57) specified, definition of terms comes after the creation of terms in scientific research. The novelty of this field, leading to a lack of uniformity in jargon and everyday divergence of terms in common usage, may be the explanation of this phenomenon. A similar differentiation is observed in the computer science / informatics divide separating researchers in the field. Whereas European sources prefer the term "informatics", putting information before the devices used to process it; American researchers seem to prefer "computer science" as their term for this field (Kalelioğlu, Gülbahar and Kukul, 2016). Nonetheless, despite differences in terminology, it is observed that the fundamental focus of this field is the basic principles of computer science and their interaction with mankind.

2. The History of Computational Thinking

While computational thinking is widely considered to have begun by Wing's (2006) article on the subject, it was first referenced by Papert (1996), as "procedural thinking". Papert, then in MIT's Department of Mathematics, in the course of his research on computer and software usage in solving geometric problems claimed that computational thinking could be employed in defining the relationship between a problem and its solution and the structuring of data. Papert and his colleagues had developed the LOGO programming language in the 1960's. The main aim of this language was aiding students in thinking mathematically and logically. LOGO was at its core a constructivist language, accepting learning to be a fundamentally individual activity and explaining it in Piagetian terms. Papert (1991: p.1)'s individualization of this concept resulted in the notion of learning-by-making. Papert's

adoption of this philosophy is not surprising, considering his experience working alongside Piaget in the Centre of Genetic Epistemology in Geneva between 1958 and 1963. LOGO was thus designed as an environment conducive to and supportive of Piagetian learning (Logo, 2015).

Figure 1. Seymour Papert and LOGO-based robot Turtle.

LOGO and the constructivist ethos behind it were considered to have the potential to transform education when the language was first introduced. This potential did not come to life however, as constructivism gradually lost traction in the education systems of the UK and the USA (Agalianos, Noss, and Whitty, 2001: p.497). This loss was not unprecedented, as other programming languages such as PLATO (Programmed Logic for Automatic Operations), CAI (Computer Assisted Instruction), CBT (Computer Based Training) and CAL (Computer Assisted Learning) also faced the same fate (Etherington, 2017).

3. Defining Computational Thinking

As computational thinking is a newborn field, its definition varies from researcher to researcher. Due to this variation between academics, this paper will consider practical definitions offered by organizations such as ISTE (International Society for Technology in Education) and CSTA (Computer Science Teacher Association) in addition to those determined by the academics themselves. Wing (2006, p.33) defines computational thinking as "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science.". However, after further revisions [as the original article was 4 pages long and many topics were not fully explored.] a different definition was accepted in 2011. According to Wing (2011), computational thinking is defined as "Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.". Table 1 showcases the various definitions of computational thinking employed by the contemporary academia.

Table 1. Contrasting Definitions of Computational Thinking.

Definition	Source
...the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.	(Cuny, Snyder, Wing, 2010 akt. Wing, 2011, p.20)
Computational thinking is the thought processes used to formulate a problem and express its solution or solutions in terms a computer can apply effectively.	Wing (2014)
The mental process for abstraction of problems and the creation of automatable solutions.	Yadav et al. (2014)
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.	Furber (2012)

Computational thinking has a long history within computer science. Known in the 1950s and 1960s as "algorithmic thinking," it means a mental orientation to formulating problems as conversions of some input to an output and looking for algorithms to perform the conversions. Today the term has been expanded to include thinking with many levels of abstractions, use of mathematics to develop algorithms, and examining how well a solution scales across different sizes of problems.	Denning (2009)
...[Computational Thinking] is to teach them how to think like an economist, a physicist, an artist, and to understand how to use computation to solve their problems, to create, and to discover new questions that can fruitfully be explored.	Hemmendinger (2010)

These definitions tend to focus on the cognitive performances and processes of individuals. Accordingly, we may conclude that activities based on computational thinking are essentially meant to improve cognitive skills and support the processes of teaching and learning in the affected individuals.
Researchers in the field have also held workshops with the aim of establishing the true nature of and a working definition for computational thinking. Some of these workshops have concluded that a rigorous and consistent definition would benefit the field (BİD Workshop Committee, 2011). On the other hand, certain researchers held that attempting to define computational thinking in clear-cut terms is unnecessary and that effort should be applied in establishing the internal relationships within the computational thinking corpus (Voogt et al., 2015: p.726):

> *"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 (Voget et al., 2015: p.726)."*

Whilst a general concept of computational thinking can be established based on these definitions, they offer little insight into how computational thinking should be applied in practice in the field of education. Practical definitions of computational thinking and its constituents are needed before achievement targets and educational programmes can be created in the classroom. CSTA and ISTE have provided activity rubrics for computational thinking in the years 2011, 2015 and 2016. Table 2 is a list of these activities, sorted according to keywords.

Table 2. Practical computational thinking activities, curated by ISTE.

Keywords	**Source**
Formulating, organizing, analyzing, modelling, abstractions, algorithmic thinking, automating, efficiency, generalizing, transferring	ISTE (2011)
Creativity, algorithmic thinking, critical thinking, problem solving, cooperation	ISTE (2015; Oden et al. 2015)
Data analysis, abstract thinking, algorithmic thinking, modelling, representing data, breaking problems into components, automation	ISTE (2016) (Computational Thinker Definition)

As these definitions show, the activity lists provide a framework for educators, delineating the educational achievements which they should aim for and outlining methods for assessment and evaluation of these achievements. For example, an educator using these rubrics would know that teaching visual programming tools such as Scratch or KODU in class, is not only meant to help students have fun while designing computer games; They would also use the experience as a medium for instilling some of the concepts and abilities outlined in Table 2.

4. Components of Computational Thinking

The fundamental components of computational thinking are also a source of divergence between researchers. In order to establish a baseline for further analysis, components used by various researchers have been provided in Table 3.

Table 3. Components of Computational Thinking

Components	Source
Abstraction, Algorithms, Automation, Problem Decomposition, Parallelization, Simulation	Barr & Stephenson (2011)
Abstraction, Automation, Analysis	Lee et al. (2011)
Abstraction, Algorithmic Thinking, Decomposition, Evaluation, Generalization	Selby & Woollard (2013)
Abstraction, Algorithms, Decomposition, Debugging, Generalization	Angeli et al. (2016)
Abstraction, Algorithms, Automation, Problem Decomposition, Generalization	Wing (2006, 2008, 2011)

While the exact components may differ, we believe the essential concepts they represent are largely uniform across the field. Computational thinking abilities are essentially the set of skills needed to convert complex, messy, partially defined, real-world problems into a form that a mindless computer can tackle without further assistance from a human (BCS, 2014, p.3). As such, this paper will use the definitions of abstraction, problem decomposition, algorithmic thinking, automation and generalization from amongst the components provided. These definitions can be listed as (Humphreys, 2015):

- *Abstraction* makes problems or systems easier to think about. Abstraction is the process of making an artefact more understandable through reducing the unnecessary detail and number of variables; therefore leading to more straightforward solutions. One of the best-known examples of this is the London Underground example, provided by Humphreys (2015). The London Underground map provides just enough information for the traveller to navigate the underground network without the unnecessary burden of information such as distance and exact geographic position. It is a representation that contains precisely the information necessary to plan a route from one station to another – and no more. Similar examples may be provided for other subjects, allowing the concept to be better understood (Wing, 2008):
    - Verbal and story-based problems in mathematics such as filling rates of pools, areas to be fenced off and accounting calculations are essentially an exercise in abstraction for the students where they are required to separate relevant and irrelevant data and state their solutions in the symbolic language of algebra, geometry, or arithmetic.
    - In geography, students make use of specialized maps (physical, topographic, political, touristic etc.), ignoring many aspects of real-world geography in favour of ease-of-access for data relevant to their current study.
    - History lessons are essentially abstractions of local histories and individual biographies taught as national or world history – abstract projections of real-world events.
- *Problem Decomposition* is a method for taking apart problems and breaking them into smaller and more understandable constituents. This method is also known as “Divide and Conquer”.
- *Algorithmic Thinking* is the process of constructing a scheme of ordered steps which may be followed to provide solutions to all constituent problems necessary to solve the original problem.
- *Automation* is the configuration of formed algorithms over computers and technological resources to be efficiently applicable to other problems.

- *Generalization* is the process of adapting formulated solutions or algorithms into different problem states, even if the variables involved are different.

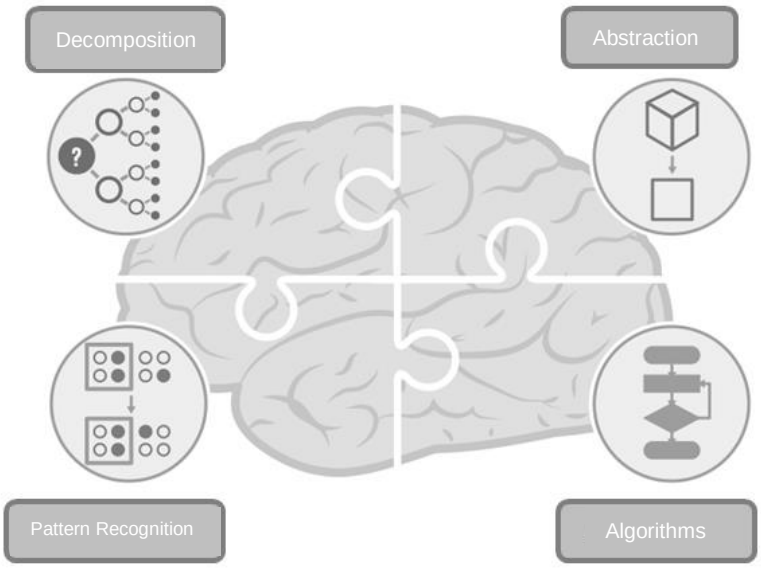

Figure 2. 4 basic strategies for computational thinking (McNicholl, 2018: p.37).

There are also a number of techniques used to exemplify and evaluate computational thinking. These comprise the equivalent of a scientific method for computer science. They are employed to put computational thinking to practice in the classroom, at home and at work (Humphreys, 2015):

- Reflection
    - Reflection is the skill of making judgements (evaluation) that are fair and honest in complex situations that are not value-free. Within computer science this evaluation is based on criteria used to specify the product, heuristics (or rules of thumb) and user needs to guide the judgements. A child's realization, when playing with pebbles, that 3 + 4 is the same as 4 + 3 is an example of reflection (or rather, reflective abstraction). The information created in this example is derived not from the pebbles themselves but from the actions taken on them.
- Coding
    - An essential element of the development of any computer system is translating the design into code form and evaluating it to ensure that it functions correctly under all anticipated conditions. Debugging is the systematic application of analysis and evaluation using skills such as testing, tracing, and logical thinking to predict and verify outcomes.
- Designing
    - Designing involves working out the structure, appearance and functionality of artefacts. It involves creating representations of the design, including human readable representations such as flowcharts, storyboards, pseudo-code, systems diagrams, etc. It involves further activities of decomposition, abstraction and algorithm design.
- Analysing
    - Analysing involves breaking down into component parts (decomposition), reducing the unnecessary complexity (abstraction), identifying the processes (algorithms) and seeking commonalities or patterns (generalisation). It involves using logical thinking both to better understand things and to evaluate them as fit for purpose.

- Applying
    - Applying is the adoption of pre-existing solutions to meet the requirements of another context. It is in generalization - the identification of patterns, similarities and connections - and exploiting those features of the structure or function of artefacts. An example includes the development of a subprogram or algorithm in one context that can be re-used in a different context.

5. Critique and Contemporary Research in Computational Thinking

Wing (2006), in the article "Computational Thinking", provided a definition of computational thinking, and held that computational thinking is a fundamental ability for the future which will become a necessity for all individuals and should be employed in the curriculums for students of all levels. However, the article itself in Wing (2006) totaled only 4 pages, was not based on independent research and lacked in-depth analysis of many topics covered in the article. While the article has been used as a foundation for research done by many academics, it has also been put under a heavy amount of critique. Hemmendinger (2010) especially claimed that the components of computational thinking as presented in Wing (2006) are not unique to computational thinking. According to Hemmendiger (2010):

- Reformulating hard problems is typical of all domains of problem solving,
- Philosophers have been thinking about thinking — recursively — for a long time,
- Mathematics surely uses abstraction, and so do all disciplines that build models,
- Separation of concerns and using heuristics also characterizes problem-solving in general.,

Furthermore, Hemmendinger (2010) advances that teaching individuals involved in other disciplines how to think like a computer scientist is unreasonable. Rather than employing a single discipline to dictate the thought processes for all disciplines, physicists should think like physicists and economists should think like economists *while* making use of computational thinking and computational processing technologies in order to solve questions in their field efficiently and determine new questions which would result in novel, efficient methods once solved. Another objection to Wing comes from Denning (2016). According to Denning (2016), the article ascribes an undeservedly significant weight to algorithms and algorithmic thinking. Rather than valuing algorithms above their contribution, Denning (2016) suggests that an algorithmically-controlled computational thinking model should not be ignored as an alternative. Additionally, they advance the notion that computational thinking is not a fundamental skill and cannot be regarded as an equal to fundamental abilities such as reading and writing. In short, the idea that every individual can use computational thinking and campaigns with claims such as "Coding for Everyone", "A Nation of Coders" and "A Coder at Every Home" are unrealistic. The question of whether every profession and every individual really needs to employ computational thinking and consequential coding abilities as a part of computational thinking, is an unresolved discussion in the field. One of the most striking comments on this conundrum is provided by Barr & Stephenson (2011: p.113):

> *The ultimate goal should not be to teach everyone to think like a computer scientist, but rather to teach them to apply these common elements to solve problems and discover new questions that can be explored within and across all disciplines (Barr and Stephenson, 2011: p.113).*

Learning computational thinking and computer science are not one and the same. Yet colloquially, these two expressions are used interchangeably. This supposed equivalency is erroneous as the latter is essentially meant to educate learners in the study and use of the principles of mathematical calculation. One reason why this belief is in wide circulation could possibly be Wing (2006)'s original claim that *"computational thinking is thinking like a computer scientist."*. Denning (2009) and Hemmendinger (2010) oppose this claim mainly because of their thesis that such a definition of computational thinking could give the impression that computational thinking is only relevant to the field of computer science and is largely inapplicable to everyday situations in would-be computational thinking learners.

Programming education is a sub-field of computer science and while primarily conducted to educate learners in the best practices of computer programming, one of its goals is being conducive to the creation of high-quality computer programs. Computational thinking, while it has considerable overlap with computer science on certain elements, focuses mainly on developing and disseminating approaches to problem solving, unlike computer science.

While the terms "coding" and "programming" are used interchangeably with each other, "coding" has been employed as a more exciting and less scary alternative, especially to entice and motivate beginners in scripting. Platforms such as Code Studio, Hour of Code, Code Monkey and MIT's Scratch and App Inventor 2 tend to use coding rather than programming. More advanced text-based and OOP languages (Python, Java etc.) edge towards the use of programming instead. One widely-held belief is that computational thinking, and as a result coding and programming education, has a positive effect on students' problem-solving abilities. Multiple different manifestations of this belief may be observed in contemporary research, and it can be connected to more solid scientific reasoning via analyzing the results of contemporary research:

- Palumbo (1990)'s meta-analysis study concluded that strong evidence to the existence of a meaningful correlation between programming education and problem-solving abilities could not be found. Palumbo (1990) came to this conclusion by evaluating different studies conducted on high school students by a

variety of researchers. These included studies based on CAI (Computer Aided Instruction), LOGO and BASIC languages being taught to different groups of students in various class hours and total course length in weeks configurations – none of which discovered a scientifically significant correlation. As previously stated in this article, one of the reasons for the near-extinction of these programming languages may be their inability to provide the expected contribution to the students' problem-solving abilities.

- Kalelioğlu & Gülbahar (2014) held a 5-week long study with 5th Grade Middle School students (22 girls and 27 boys) in the 2013-2014 educational year. Students conducted varying activities in the Scratch programming language as part of the study. Their results indicated that when quantitative data is analyzed, there was no statistically significant divergence between the pre-study and post-study problem-solving ability quotients. Analysis of qualitative data, on the other hand, showed increased student enthusiasm towards programming.
- Kukul & Gökçearslan (2014) worked with 304 5th and 6th grade students who had not taken any programming lessons previously. Similarly, to Kalelioğlu & Gülbahar (2014), they also used Scratch. Their conclusions indicated that no statistically significant change in the students' problem-solving abilities was observed.
- Morelli et al. (2011) analyzed the results under specific indicators. The "App Inventor" mobile programming application was taught to high school students as part of a summer programme. Neither the "*problem-driven learning*" nor "*support for learning"* indicators mention an increase in the problem-solving abilities of students, instead opting to focus on the increase in motivation observed.
- Wong et al. (2015) conducted an experimental study on 264 5th Grade students in Hong Kong between the years of 2012 and 2014. The first year of the study was used to teach KODU (A game engine developed by Microsoft) to the students, while in the second year Scratch and Small Basic were used in the curriculum. The students' mathematics grade average rose from 74.86 in 2012-2013 to 77.59 in 2013-2014. The students' creativity, critical thinking and problem-solving abilities were also evaluated. Based on t-Test results conducted on data retrieved from the ESDA student evaluation portal, the students' problem-solving abilities appeared to rise from 2.75 to 2.95. However, while the researchers did indicate that participation in coding developed certain abilities in the students, other fundamental abilities were not conclusively affected.

Various strong claims have been made regarding the positive influence of programming/coding education in the cognitive development of children. Papert (1980), believed that programming allowed children to shape their own learning environments. Papert's most important claim was that learning LOGO improved problem-solving abilities by providing concrete experiences which were conductive to conceptualizing pictures on an operational scale (As Papert himself was a mathematician, his examples were frequently based on mathematics and geometry. Concrete experiences were defined as the appearance of geometric shapes on the screen.). Formal operational thinking was defined by Piaget as the ability to construct relationships, make inferences and build hypotheses (Kıncal & Yazgan, 2010: p.724). An individual capable of formal operational thinking can make abstractions, understand mathematical constructs requiring high-level thinking, generalize by applying the acquisitions from these problems to other problems, is able to make plans, and employs a procedural method of thinking. At this point, the similarities between formal operational thinking as defined by Piaget and CT-based abilities become apparent. This is why Papert claimed that LOGO could aid in dismissing negative attitudes towards math in students, teaching mathematical concepts, and strengthening self-control and success-oriented attitudes in children (Liao & Bright, 1991: p.252).

Results from these studies show conflicting opinions in computational thinking literature when it comes to the question of whether programming education on its own has a meaningful effect in the problem-solving abilities of students. But studies where components of computational thinking are employed show an increase in the students' problem-solving, abstract-thinking, troubleshooting and cooperative learning abilities.

- Roman-Gonzales et al. (2017) studied 1251 Spanish students in 5th – 10th grades. CTt (Computational Thinking Test) and PMAt (Primary Mental Abilities Test) were applied to the students. The correlation between CT abilities and "spatial memory", "Reasoning" and "Problem-solving" was calculated experimentally, with spatial memory being k (r=0.44), reasoning (r=0.44) and problem-solving (r=0.67). Problem-solving appears to be more heavily-influenced than other abilities.
- Grover, Pea & Cooper (2015) worked with 54 students in Northern California who were between 11 and 14 years old. A 7-week course was designed for the students where they used the Scratch coding platform and were able to translate their code into text-based platforms based on their acquisitions from the platform. The researchers were able to correlate CT abilities with problem-solving abilities. When the results are analyzed, the students are shown to have advanced themselves especially in algorithmic thinking abilities. Another interesting point is that the students' previous CT experiences and

mathematical abilities (as determined by an introductory exam conducted by the researches) were strong indicators of learning outcomes. Pea & Kurland (1984, p.35) enumerated "mathematical ability", "memory capacity", "analogical reasoning ability", "situational reasoning ability" and "procedural thinking ability" as the mathematical skills necessary for acquisition of programming ability, while also specifying that students who are especially able to operate the LOGO language successfully were also successful in English and humanities classes, and not merely in mathematics.

- Webb (2010) assayed the contribution of programming education to students' troubleshooting abilities. A regimen of 2 hours per week for 5 weeks was planned; with CT skills being connected to problem-solving ability. While 19 boys and 21 girls were present at the beginning, due to personal reasons and exams, only 24 students (16 boys, 8 girls) completed the regimen. At the end of the study, the students were asked to "Fix the Frogger Program" in 40 minutes. Only 1 student failed this assignment, with the rest proceeding to the debugging phase.
- The study conducted by Bers et al. (2013) was based on 3 pre-school classes (2 public and 1 private) of 53 students in total, and had a length of 20 hours. During this study, learners were exposed to 6 main subjects including engineering design processes, robotics, instruction-based programming, loops, sensors, and control mechanisms. TangibleK robots and software were employed in the study. The contents of these subjects were tailored to suit the students' age. Songs, games, and rhythmic and repetitive moves were inserted to the applications. For example, "Simon Says" was used in lesson 3: algorithmic programming and CHERP (Creative Hybrid Environment for Robotic Programming), a drag-and-drop software was taught. The students' troubleshooting, understanding of the relationship between instructions and movement, and use of instruction order and flow-control instructions was studied. The results indicated that students' abilities to cooperate, create ideas, share via negotiation as well as motor skills improved. Furthermore, the students were described to have become more active in their creativity and problem-solving abilities, both in the mathematical and real world.
- The study conducted by Durak and Sarıtepeci (2018) was applied to 156 public school students in Ankara. Two different data collection tools were used in this study. The first one is the personal information form and the second one is the computational thinking ability form. In this study, the factors affecting the computational thinking skills of students were examined. These factors are gender, education level, IT usage experience, daily internet usage period, mathematics achievement, attitudes towards the mathematics course, attitudes towards science courses, achievements in science courses, achievements in information technology courses and attitudes towards information technology courses. Among these factors, it was determined that the most effective factors on computational skills were education level, mathematics achievement, attitude towards the mathematics course and attitude towards science courses.

Upon analysis of these studies, it becomes apparent that it is lessons in coding, mathematics, natural sciences, social sciences and language arts, taught according to computational thinking skills and not mere programming or coding education, which affect an increase in the problem-solving, abstract thinking, troubleshooting, procedural thinking and similar abilities in students. An appropriate and interdisciplinary application of the component of CT abilities needs to be advanced in order to raise students not only as coders but as individuals with a radical way of thought and perspective. Furthermore, it may be appropriate for Computational thinking and STEAM (Science, Technology, Engineering, Arts and Mathematics) to be considered together as these two fields share a great deal of subject material (Gülbahar, 2017: p.331). Interdisciplinary work on the part of the students and their ability to realize relationships between areas of study, determine the problems they are facing, investigate potential solutions, decide upon the correct solution, gather data, analyze data, troubleshoot, develop their models and generalize solutions (ISTE, 2016) will aid their problem-solving abilities.

6. Conclusion

Computer science-based technologies are developing rapidly in our era, influencing the problem-solving processes and social lives of both individuals and societies. From medical work to social media use, results of computer science studies are integrated to the daily lives of individuals in a multitude of fields. The effects of computer science on modern society is also an indicator of its effects on the scientific method and therefore, naturally, scientists. Natural scientists have long positioned computation as a "third" foundation of the scientific method alongside theory and experimentation, and that computational thinking is essential to their work (Denning, 2009).

Though the definitions of and framework for computational thinking as set out by Wing (2006) have long been critiqued by other researchers, the importance of computer science has been growing daily, finding applications in multiple fields from curing disease to preventing terrorist attacks. Nonetheless, the claim that computer science

and as a results computational thinking is a fundamental discipline on par with reading, writing and basic arithmetic, is still being debated.

Populist notions such as "Computer Science and Computational Thinking for All", aimed at bringing the field to the mainstream, will make it more difficult for the field to preserve its rightful rigour. As we have deducted from the works of Denning, Hemmendinger and Barr amongst others presented in this article, ascribing an undeserved importance to certain fields – whether they be deemed coding, computer science, or computational thinking – would be inappropriate. Still, researchers may benefit from holding computational thinking as a potential method of transforming education, as long as they also hold the criticisms applied to the field in equal regard. As Denning (2010, p.28) has also stated, holding computational thinking (and coding) in (undeservedly) excessive esteem may lead us back to the same pitfalls we are attempting to avoid.

As a final remark, we hold that the fundamental goal of computational thinking (and instilling this ability in students) and computer education should be aiding students in understanding and – through use of their creative impulses – changing the world they live in, for the better (Department for Education, 2014, p.217).

References

Agalianos, A., Noss, R., & Whitty, G. (2001). Logo in mainstream schools: the struggle over the soul of an educational innovation. *British Journal of Sociology of Education*, *22*(4), 479-500.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. *Journal of Educational Technology & Society*, *19*(3), 47.

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.

BCS, The Chartered Institute for IT. 2014. Call for evidence - UK Digital Skills Taskforce. http://bit.ly/ILi8mdn [Retrieved 17.01.2018].

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

Bringuier, J. C. (1980). Conversations with Jean Piaget. *Society*, *17*(3), 56-61.

Çınar, M. & Tüzün, H. (2017, February). Bilgisayımsal Düşünme Sürecinin Doğasına İlişkin Nitel Bir Analiz (A Qualitative Analysis on the Nature of the Computational Thinking Process). Presented to 19. Akademik Bilişim Konferası (Conference on Academic Informatics), Aksaray University, retrieved 24.12.2017 from http://ab.org.tr/ab17/ozet/233.html.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?. *Computers & Education*, *58*(1), 240-249.

Denning, P. J. (2009). The profession of IT Beyond computational thinking. *Communications of the ACM*, *52*(6), 28-30.

Department for Education. 2014. The National Curriculum in England, Framework Document. Reference: DFE-00177-2013. Retrieved 26.12.2017 from: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/335116/Master_final_national_curriculum_220714.pdf.

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. Computers & Education, 116, 191-202.

Etherington,C. (2017), Retrieved 24.12.2017 from: https://news.elearninginside.com/how-plato-changed-the-world-in-1960/.

Furber S (2012) Shut down or restart? The way forward for computing in UK schools. Technical report, The Royal Society, London.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199-237.

Hemmendinger, D. (2010). A plea for modesty. *Acm Inroads*, *1*(2), 4-7.

Humpreys, S. (2015). Computational Thinking, a guide for teacher. Computing at School. Charlote BCS. The Chartered Institue for IT

ISTE (2011), Operational definitions of computational thinking, retrieved 24.12.2017 from: https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf.

ISTE (2016), ISTE Standarts for Students, retrieved 24.12.2017 from: http://www.iste.org/docs/Standards-Resources/iste-standards_students-2016_one-sheet_final.pdf?sfvrsn=0.23432948779836327.

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via Scratch on problem solving skills: a discussion from learners' perspective. *Informatics in Education*, *13*(1), 33.

Kalelioglu, F., Gülbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*(3), 583.

Keser, H. (2011). Türkiye'de Bilgisayar Eğitiminde İlk Adım: Orta Öğretimde Bilgisayar Eğitimi İhtisas Komisyonu Raporu (Turkey's First Steps in Computer Education: Specialized Commission on Computer Education in Secondary Education Report). *Eğitim Teknolojisi Kuram ve Uygulama (Theoretical and Practical Educational Technologies)*, *1*(2), 83-94.

Kıncal, R. Y., & Yazgan, A. D. (2010). Investigating the formal operational thinking skills of 7th and 8th grade primary school students according to some variables. *Elementary Education Online*, *9*(2), 723-733.

Korkmaz, Ö., Çakır, R., Özden, M. Y., Oluk, A., & Sarıoğlu, S. (2015). Bireylerin Bilgisayarca Düşünme Becerilerinin Farklı Değişkenler Açısından İncelenmesi (A Multi-Variable Investigation of the Computational Thinking Abilities of Individuals). *Ondokuz Mayıs Üniversitesi Eğitim Fakültesi Dergisi (19th May University Faculty of Education Journal)*, *34*(2), 68-87.

Korkmaz, Ö., Çakır, R., Özden, M. Y., Oluk, A., & Sarıoğlu, S. (2015). Bireylerin Bilgisayarca Düşünme Becerilerinin Farklı Değişkenler Açısından İncelenmesi (A Multi-Variable Investigation of the Computational Thinking Abilities of Individuals). *Ondokuz Mayıs Üniversitesi Eğitim Fakültesi Dergisi (19th May University Faculty of Education Journal)*, *34*(2), 68-87.

Kukul, V., & Gökçearslan, Ş. (2014). Scratch ile programlama eğitimi alan öğrencilerin problem çözme becerilerinin incelenmesi. (Investigation of the Problem-solving Skills of Students with Scratch-based Programming Education.)

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32-37.

Liao, Y. K. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research*, *7*(3), 251-268.

Logo Foundation (2015). Logo and Learning, retrieved 24.12.2017 from: http://el.media.mit.edu/logo-foundation/what_is_logo/logo_and_learning.html.

McNicholl, R.(2018). Computational thinking using code.org. Hello World, 4, 37.

Morelli, R., De Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., & Uche, C. (2011, March). Can android app inventor bring computational thinking to k-12. In *Proc. 42nd ACM technical symposium on Computer science education (SIGCSE'11)* (s. 1-6).

National Research Council. (2010). Committee for the Workshops on Computational Thinking. In *Report of a workshop on the scope and nature of computational thinking, Natl Academy Pr*.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of educational research*, *60*(1), 65-89.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, *36*(2), 1-11.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New ideas in psychology*, *2*(2), 137-168.

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2016). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. Computers in Human Behavior, 1-14

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.

Tekerek, M., & Altan, T. (2014). The effect of scratch environment on student's achievement in teaching algorithm. *World Journal on Educational Technology*, *6*(2), 132-138.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715-728.

Wing, J. (2014). Computational thinking benefits society. *40th Anniversary Blog of Social Issues in Computing*, *2014*.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

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.

Wing, J.M. (2011), Research Notebook: Computational thinking -what and why? The Link Magazine, 20-23. https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

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 (TOCE)*, *14*(1), 5.

Yecan, E., Özçınar, H., & Tanyeri, T. (2017). Bilişim Teknolojileri Öğretmenlerinin Görsel Programlama Öğretimi Deneyimleri (A Collection of Visual Programming Experiences by Information Technologies Educators). *İlköğretim Online (Elementary Education Online)*, *16*(1).