BİLGİSAYAR BİLİMLERİ VE TEKNOLOJİLERİ DERGİSİ
JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGIES
http://dergipark.org.tr/tr/pub/bibted

Araştırma Makalesi

Improving Computational Thinking Skills with the Collaboration of Computer Science and Mathematics

Rukiye ALTIN*[1],

[1]*University of Kiel, Department of Computer Science, Kiel, Germany*

Keywords:
Computational Thinking
Computer Science
K12
ICT
Programming

ABSTRACT

The importance of Computational Thinking (CT) skills has gained significant attention in K-12 education, with several research studies highlighting the key role of CT in today's education. To enhance students' CT skills, various approaches have been integrated into education, with the incorporation of computer science (CS) being the most popular. This approach not only exposes students to CT but also teaches problem-solving concepts that benefit both CS and CT. Furthermore, CT involves problem-solving processes that include specific dispositions and characteristics essential for developing basic computer applications, making it a necessity for students to both conceptualize and apply these skills. The current study aims to examine the influence of programming skills by teaching CT through the integration of mathematics in an interdisciplinary exercise. This experimental research involved four experimental groups and four control groups, with an overall sample size of N = 188. The groups were randomly assigned. The study results indicated that teaching programming by integrating mathematics as an interdisciplinary approach improves both students' programming and CT skills. This study is important as it provides lesson plans for a secondary school programming course that had a positive effect on students' programming learning.

Bilgi İşlemsel Düşünme Becerilerinin Bilgisayar Bilimi ve Matematik İş birliği ile Geliştirilmesi

Anahtar Kelimeler:
Bilgi İşlemsel Düşünme
Bilgisayar Bilimi
K12
Bilişim Teknolojileri
Programlama

ÖZ

Bilgi İşlemsel Düşünme (CT) becerilerinin önemi, K-12 eğitiminde önemli bir dikkat çekmiştir ve birçok araştırma çalışması, günümüz eğitiminde CT'nin oynadığı kilit rolü vurgulamaktadır. Öğrencilerin CT becerilerini geliştirmek için çeşitli yaklaşımlar eğitime entegre edilmiştir ve bilgisayar biliminin (CS) entegrasyonu en popüler yaklaşım haline gelmiştir. Bu yaklaşım, öğrencileri sadece CT ile tanıştırmakla kalmaz, aynı zamanda hem CS hem de CT için faydalı olan problem çözme kavramlarını öğretir. Ayrıca, CT, temel bilgisayar uygulamalarının geliştirilmesi için gerekli olan belirli eğilim ve özellikleri içeren problem çözme süreçlerini içerir ve bu nedenle öğrencilerin bu becerileri hem kavramsallaştırması hem de uygulaması için bir gerekliliktir. Bu çalışma, disiplinlerarası yaklaşım ile matematiğin bilgisayar bilimleri dersine entegre edilmesi ile öğrencilerin CT becerilerinin etkisini incelemeyi amaçlamaktadır. Bu deneysel araştırma, dört deney grubu ve dört kontrol grubu olmak üzere toplamda N = 188 kişilik bir örneklemle yürütülmüştür. Gruplar rastgele atanmıştır. Araştırma sonuçları, matematiğin disiplinler arası bir yaklaşımla entegrasyonu yoluyla programlama öğretiminin, öğrencilerin hem programlama hem de CT becerilerini geliştirdiğini göstermiştir. Bu çalışma, ortaokul düzeyinde bir programlama dersi için olumlu bir etki yaratan ders planları sunduğundan önemlidir.

***Sorumlu Yazar**

*(altin.rukiye@gmail.com) ORCID ID 0000-0001-7593-2775

e-ISSN: 2717-8579

Geliş Tarihi: 29/09/2024; Kabul Tarihi: 27/05/2025

1. INTRODUCTION

Today's living keeps us connected to computers and computer applications more than ever. Consequently, computer science, as a discipline, has become extremely significant and those with computer science skills such as programming are in high demand to fill positions in numerous areas of modern life where such skills are required. Also, according to Dagdilelis et al. (2004) students who are learning programming at a young age get better at problem solving skills because it helps their critical thinking. On the other hand, computation means solving a problem step by step and showing the solution in a way that a computer can understand and use. Scholars note that the concept of "computational thinking" (CT), often described as thinking in ways similar to a computer, has existed since the 1950s, alongside the broader notion of problem-solving (PS) (Haseski, Ilic, & Tuğtekin, 2018; International Society for Technology in Education & Computer Science Teachers Association, 2011; Kalelioğlu, Gülbahar, & Kukul, 2016; Tran, 2019).

It is stated that CT means finding ways to understand and solve problems using tools like computers to find, study, and show information. (CSTA, 2011). Consequently, CT can be defined as the ability to solve problems by applying fundamental concepts such as abstraction, developing step-by-step strategies through algorithms, and collaborating with others on tasks grounded in real-world contexts. CT is a way to solve problems that can have more than one right answer. Researchers highlight that CT can be used to solve many problems by making guesses, predictions, and using simple ideas, abstraction (Baytak and Land, 2011a; Grover and Pea, 2013; Schulte et. al., 2025; Wing, 2006). While programming is often considered a key component in developing CT skills, it is argued that it is not the sole pathway. and also it emphasized that CT should also be supported through activities that promote higher-order thinking skills (Grover and Pea, 2013). Selby and Woollard (2013) explain that CT involves two main skill areas: problem-solving and computer science. Important skills within CT include logical thinking, creating step-by-step instructions (algorithmic thinking), problem-solving, analyzing and generalizing information, designing systems, automating tasks, making models and simulations, visualizing data, and understanding basic computer science concepts.

According to scholars, CT skills can grow not just by programming and solving problems, but also by using pictures to understand ideas (visualization), linking problems to real life, fixing errors in code (debugging), and using simple ideas (abstraction). (Armoni, 2012; Wing, 2008; Baytak and Land, 2011a; Altin et. al., 2021; Maloney et al., 2010; Nowak et al., 2002). Following past research, this study wants to test how learning computer science and using math as a second subject can help students improve their CT skills.

1.1. Linking Computer Science Education to Computational Thinking Competencies

Computational thinking (CT) means understanding ideas that help people learn and solve problems. One way to build CT is by learning programming because programming gives a positive impact on achieving CT aspects (Selby and Woollard, 2013). It also helps people understand how others behave in the community. Therefore, learning to program is valuable not only because it equips students with the ability to solve real-world, everyday problems, but also because it enhances their understanding of various behaviors exhibited by individuals within their communities (Howland et al., 2009, Altin et. al., 2021). Programming is also considered vital within society, as it enables learners to address practical, real-life challenges. Those who acquire knowledge of computational languages develop the skills needed to engage effectively with computers and technology-based systems. According to Wing (2008) many fields in science and engineering are interacting with computers in their nature and this interaction creates a natural link that makes us understand the relation between systems and deeper CT. These benefits strongly support the learning and development of CT in education.

Learning how to program and improving computer skills are closely connected to each other. Computer skills help learners understand programming better, and learning programming helps students gain advanced computer skills. These skills can then be applied to solve problems in real life. Programming and computer skills work well together and help students learn and grow (Cansu and Cansu, 2019). Wing (2006) explains that even though CT and computer science are not the same, students can get better at CT while they learn to program.

Students also learn to think by using CT they gain during learning. Researchers agree that programming helps develop metacognitive skills, which are essential for CT. This is because programming helps students connect new information with what they already know (Depryck, 2016; Wing, 2006; Allsop, 2019; Romero et al., 2017). Moreover, programming aspects such as sequence, conditional operations, debugging, loops are involving metacognitive skills such as evaluating, modifying, reflecting and understanding the nature of the problem (All- sop, 2019). Nouri et al. (2020) indicate that even though programming has been part of education in K-12, it took a new meaning today by including CT in it and teachers started to look for new ways of teaching methods to develop

both CS and CT skills at the same time. Therefore, learning programming largely depends on students' computational skills knowledge, and therefore computational skills are a significant part of learning programming. However, to facilitate learning programming to young students, new teaching methods are required. Therefore, these two aspects depend on each other to achieve the desired educational goals.

Threekunprapa and Yasri (2020) say that the connection between CT and computer science is like the connection between shapes and functions in a flowchart, which helps show how a program works. In other words, CS and CT are connected naturally. Studies show that computer science helps people use skills like algorithmic thinking, critical thinking, and problem-solving because these skills are part of CT and can be used to solve real-life problems. (Anderson, 2016; Czerkawski and Lyman, 2015; Ioannidou et al., 2011; Israel et al., 2015; Wing, 2006). According to Grover and Pea (2013) CT focuses on breaking down a problem into manageable disparate parts and then generalizing it into patterns, which is largely the same as happens in computer science when correcting coding errors through debugging and the application of recursive thinking. Li et al. (2020) highlight that programming is considered as difficult to understand and in the same way improving CT skill is seemed as difficult, therefore, connecting these two fields is a way of making both easier to not just understand the depth of but to improve the skills in both computer science and CT. Computer science contributes to the development of CT skills, as both disciplines emphasize problem-solving through analytical reasoning and innovative approaches. To connect programming with computational skills, learners must be able to articulate clear, detailed instructions and translate them into sequential steps suitable for developing the intended program. The program developed can then be used to solve problems in professional settings. An additional aspect involves envisioning a system made up of various components that outline the scope the program is intended to address. These program components can be different, but they shouldn't have overlapping functions (Kurilovas and Dagiene, 2016). Systems designed this way should be easy for users to understand and operate. Learning to program also requires grasping complex systems and their behaviors, which becomes evident through the interplay between CT and the acquisition of programming languages. System designers are expected to build models that transparently demonstrate their functionality and offer insights relevant to the original problem. More broadly, CT encompasses the logical organization and examination of data. After a thorough understanding and analysis of a problem, potential solutions can be formulated to address it.

"According to Barr and Stephenson (2011), computer science plays a vital role in society by enhancing algorithmic thinking through problem-solving. This, in turn, supports the adoption of CT practices, facilitated by the field's inherently interdisciplinary nature. Kong et al. (2020) the three components; sequences, loops and conditions come from computer science serving CT components directly because it enables young learners to connect their perspective with the digital world. Hemmendinger (2010) also emphasized that computer science and CT overlap in several core competencies, including analysis, modeling, system development, and error detection and correction (debugging). Therefore, when educators design a computer science course, they often inherently incorporate elements of CT. The instructional approach to CT aligns with teaching fundamental algorithms, as both involve interpreting and constructing step-by-step solutions to problems in a logical and reflective manner (Lu & Fletcher, 2009). Research has mostly focused on how to improve different skills areas through interdisciplinary collaboration. The significance of programming in teaching CT is well established, as it contributes not only to the development of essential 21st-century competencies (Sirakaya & D, 2019) but also to the enhancement of algorithmic thinking skills. These skills include the ability to approach problems methodically, create simulations, visualize solutions, and identify and correct errors through debugging (Kazimoglu et al., 2012). By incorporating computer science into K–12 education, students are exposed to opportunities that extend beyond learning to code. They also engage in structured problem-solving processes that lie at the heart of CT. This integration supports the cultivation of logical reasoning, creativity, and persistence—key attributes that underpin both CT and broader academic and real-world success (Kong et al., 2020; Angeli & Giannakos, 2020).

2. METHOD

2.1. Research Design

This research investigates the impact of integrating programming instruction with mathematics on the development of CT skills among secondary school students. The central research question examines how different instructional strategies—specifically, a traditional teaching approach (TA) compared to an interdisciplinary approach (IA)—affect students' acquisition of these skills. Adopting a design-based research (DBR)

framework, the study employs a quasi-experimental design to structure its methodology. Statistical analyses were conducted using t-tests to compare the outcomes across the different instructional conditions.

2.2. Procedures

The idea of the study had applied to secondary school students for eight weeks. A total of eight 7th grade classes were randomly assigned to either an EG or a CoG. The instructional content, including lesson plans and teaching materials, was designed by the researcher. Two ICT teachers from the same school implemented the lessons, each delivering the content to their assigned groups according to the planned instructional approach. The researcher was mentoring the ICT teachers during the process and informing them frequently about the application of the lesson plans. In this study, the experimental group (EG) received instruction in the Small Basic programming language with integrated mathematical content. The problems used in the lessons were drawn from the official seventh-grade mathematics textbook utilized by the students. In contrast, the control group (CoG) was taught Small Basic using a traditional instructional method, emphasizing programming concepts such as syntax and structure without interdisciplinary integration. The intervention spanned eight weeks, totaling 16 hours of instruction. During the initial two weeks, both groups followed the same introductory lesson plan designed to familiarize students with the Small Basic environment. This phase also included the administration of a pretest. Over the next four weeks, the instructional paths diverged: the EG engaged with lessons that incorporated mathematical problem-solving, while the CoG continued with conventional programming instruction. In the final week, the pretest was re-administered as a posttest to assess changes in CT skills and compare learning outcomes between the two groups.

2.3. Participant

The study sample consisted of a total of 188 seventh-grade secondary school students attending a private school in Ankara, Turkey. The population of the secondary school is 980 and the total population of the school including kindergarten. primary and high school is around 4000. The participants were aged from 12 to 13 years old. Each group- both the experimental and control groups- had 94 students in total, and the gender division was done equally at 50%. The academic levels of the classes were considered similar because the school follows a policy of creating academically balanced classes by grouping students with similar GPA scores. Both groups were taught by two ICT teachers who had the same teaching approach and were working in the same school. A demographic summary of the participant students is provided in Table 1.

Table 1. Demographic Summary of the Students

	Group	*N*	%	Valid %	Cumulative %
Experimental Group (EG)	E1	24	12.8	12.8	12.8
	E2	24	12.8	12.8	25.5
	E3	24	12.8	12.8	38.3
	E4	22	11.7	11.7	50.0
Control Group (CoG)	C1	24	12.8	12.8	62.8
	C2	23	12.2	12.2	75.0
	C3	24	12.8	12.8	87.8
	C4	23	12.2	12.2	100.0

2.4. Data Collection

Quantitative data were gathered using the Computational Thinking Skills Self-Efficacy Scale (CTSSES), originally developed by Gülbahar et al. (2019). To evaluate students' progress in CT, the same scale was administered again as a posttest following the instructional period. The original development of the scale involved a sample of 952 secondary school students from various regions across Turkey. Through their analysis, the authors identified five underlying dimensions, resulting in a 39-item instrument formatted on a 5-point Likert scale.

It is reported that a Kaiser-Meyer-Olkin (KMO) coefficient of .966 and a Bartlett's Test of Sphericity with a significance value below .05 (Gülbahar et al., 2019). These results confirmed the suitability of the data for factor analysis, as the KMO exceeded the minimum threshold of .50 and the Bartlett test was statistically significant. The exploratory factor analysis revealed a five-factor structure for the original 39-item scale but in this study three items were excluded, resulting in a finalized scale with 36 items. The corrected item-total correlations ranged between .386 and .632, while Cronbach's Alpha values ranged from .762 to .930, demonstrating strong internal consistency and reliability.

3. RESULTS

To address the research question, the results from the CTSSES were analyzed for both the experimental and CoGs. To assess whether significant differences existed between pretest and posttest scores, a paired-samples t-test was performed independently for each group. This test helped identify any changes in the average scores within each group. Furthermore, independent samples t-tests were conducted to compare the pretest and posttest scores of the experimental and CoGs, allowing for the evaluation of mean differences between the two teaching approaches.

The results of both group students' CTSSES pretest and posttest are presented in Table 2.

Table 2. Results of paired t-test

Group	Scale	*M*	*Sd*	*t*	*df*	p
Exp	Pre-test	40.29	13.81	-9.775	93	.000
	Post Test	58.74	11.87			
Cont	Pre-test	44.57	16.11	0.595	93	.553
	Post Test	43.31	13.51			

*p<0.05

As shown in Table 2, the EG had a mean pretest score of X = 40.29 (SD = 13.81) on the CTSSES, while the CoG's mean pretest score was slightly higher at X = 44.57 (SD = 16.11). Following the instructional intervention, the EG's average score increased substantially to X = 58.74 (SD = 11.87), whereas the CoG's average decreased slightly to X = 43.31 (SD = 13.51). These findings suggest that the intervention had a notable positive impact on the CT skills of students in the EG. In contrast, students in the CoG showed a modest decline in their posttest scores compared to the pretest.

The t-test analysis showed that the EG's average score on the CTSSES increased by 18.45 points from pretest to posttest. In contrast, the CoG's average score decreased by 1.26 points. This suggests a significant improvement in the EG's CT skills compared to the CoG.

The results of the paired-samples t-test indicated a statistically significant difference between the EG's pretest and posttest scores ($t(93) = -9.775$, $p < .01$), suggesting a substantial enhancement in their CT self-efficacy as a result of the intervention. In contrast, no statistically significant difference was observed between the CoG's pretest and posttest mean scores. These findings suggest that the traditional instructional approach did not lead to measurable gains in the CoG's information processing and CT skills.

An independent samples t-test was subsequently performed to examine whether there were significant differences in average scores between the EGs and CoGs. Table 3 presents the results of this analysis, comparing participants' self-efficacy in CT skills according to their group assignment (experimental vs. control).

In this study, an independent samples t-test was performed to find out if the differences in mean scores between the EG and CoG is significant or not. As shown in Table 3, the analysis revealed a significant difference in both pretest ($t(186) = -1.96$, $p < .05$) and posttest ($t(186) = 8.32$, $p < .05$) scores. Prior to the intervention, the EG reported lower self-efficacy in CT compared to the CoG. However, following the instructional intervention, the EG's scores increased significantly, surpassing those of the CoG.

Table 3. Results of the Independent Samples t-Test Between Groups

Scale	Group	*M*	*Sd*	*X2-X1*	p
Pre	Exp	40.29	13.81	-4.28	.05
	Cont	58.74	16.11		
Post	Exp	44.57	11.87	15.44	.00
	Cont	43.31	13.51		

*p<0.05

The results of the independent samples t-test indicate a statistically significant difference in the pretest and posttest mean scores on the CTSSES. These findings suggest that integrating coding instruction with an interdisciplinary approach, specifically through mathematics, had a positive effect on students' self-efficacy related to CT skills.

4. DISCUSSION

This study explored the impact of programming instruction on enhancing CT skills through an interdisciplinary approach that integrated mathematics. Analysis of the results from both the independent samples t-test and paired samples t-tests revealed that students in the EG demonstrated a significant improvement in their CT self-efficacy. Specifically, their average score on the posttest was notably higher than on the pretest. In contrast, the CoG exhibited a slight decline in average scores from pretest to posttest. Quantitatively, the EG showed an average gain of +18.45 points, while the CoG showed a decrease of -1.26 points. These results indicate a substantial improvement in the EG's CT skills following the intervention. Furthermore, when posttest scores were compared across groups, students in the EG clearly outperformed those in the CoG. These findings suggest that teaching coding through an interdisciplinary framework, particularly by integrating mathematics, can effectively enhance students' self-efficacy in CT.

In a study conducted by Fields, Liu and Kafai (2017), it was observed that teachers improved students' CT skills through computer-assisted instruction and programming. Also, a study applied by Jun, Han and Kim (2017), as a result of the experimental process with students using coding programs, has been shown that design-based learning improves CT in some of the elementary school students. In conclusion, the findings of this study provide strong evidence that integrating a second discipline—such as mathematics—into

programming instruction can significantly enhance students' CT skills. This research contributes meaningfully to the field by demonstrating an effective approach for making computer science more accessible and engaging for younger learners. It also underscores the value of interdisciplinary strategies in supporting the development of core competencies such as problem-solving and algorithmic thinking. Consistent with existing literature, this study highlights a promising method for bridging CT and computer science education. By contextualizing programming within a familiar academic subject, the learning process becomes more meaningful, and the connection between the real world and the computational domain becomes easier for students to grasp. As a result, interdisciplinary instruction proves to be more effective than traditional methods in fostering CT skills. By working with mathematics teachers during the preparation of lesson plans, it has been noticed that students will be challenged with problem solving more than the traditional methods because first they need to understand the nature of mathematical problems then they have to involve programming into it to solve the problem. Scholars widely agree that studying computer science and learning a programming language can enhance a range of individual skills, particularly those related to algorithmic thinking, critical thinking, and problem-solving, which are central components of CT (Anderson, 2016; Ioannidou et al., 2011; Israel et al., 2015; Wing, 2006). The overall results of this study align with that consensus, providing evidence that programming plays a significant role in fostering the development of CT skills.

5. LIMITATION

Several limitations took the part in the study. First, working with young groups made the duration of the study longer, approximately one year. Second, the number of participants in the study became limited because some parents did not give permission for their children to be part of the study. Third, as this study took place in a private school, due to the hierarchical regulations, to make changes in the curriculum took longer than it could take in a state school. That made the duration of study longer than expected. Another limitation of the study is that as the study was applied in a private school, it only reflects the results of students in private school context.

6. CONCLUSION

Thanks to this significant research, the results explicitly show that including mathematics into programming course in K-12 level, improves students' CT skills. CT is defined as the process of understanding a problem and devising innovative solutions using computer science principles and techniques (Wing, 2008). In an increasingly technology-driven world, both computer science and CT have become essential societal competencies. Kale et al. (2018) noted that the World Economic Forum projected a significant shift in the job market by 2020, with many traditional roles becoming obsolete due to the rise of robotic technologies. However, this transformation is also expected to generate over two million new employment opportunities in emerging fields. Given the growing importance of computer science and CT, driven by rapid technological advancements in industry and the transformative effects of the COVID-19 pandemic, there is an urgent need to cultivate CT skills in the younger generation.

To address this growing need, computer science must be integrated into K–12 education—not only to enhance students' problem-solving and algorithmic thinking but also to foster the development of CT skills. Several scholars have emphasized that current opportunities within K–12 settings remain insufficient to adequately support students in acquiring these essential skills (Gülbahar & Kukul, 2016; Guzdial, 2016; Kale et al., 2018; Wing, 2008). As computer science knowledge and CT skills become increasingly essential for future generations, it is crucial for schools, educators, and policymakers to update national curricula and incorporate interdisciplinary teaching strategies. Furthermore, to make computer science more accessible to younger learners, its complexity should be reduced to create more effective pathways for developing CT. The current study has highlighted an important strategy in establishing a link between CT skills acquisition and computer science, and that teaching programming along with a second discipline such as mathematics can increase the level of learners' CT skills more than by teaching purely by traditional methods. It is suggested that this increased learning due to the connection between the comprehension of today's students with the real world and the computing world. This study proved the need for including mathematics into programming curriculum as well as creating lesson plans and designing materials in line with the currently planned Curriculum to improve students' CT. After incorporating the necessary changes into the lesson plans and instructional materials—specifically by integrating relevant mathematical problems with input from mathematics teachers—it was observed during implementation that students in the interdisciplinary group faced greater challenges with mathematics-related problem-solving compared to those taught using traditional methods. This difference can be attributed to the added cognitive demand: students first needed to comprehend the nature of the mathematical problem before attempting to develop a programming-based solution.

The approach put forward in the current study is seen as a new methodology for the teaching of computer science in conjunction with mathematics. However, further research is needed in order to examine teachers' and students' attitudes towards the appendage of a second discipline whilst learning computer programming skills. Also, future research can examine teachers' and students' attitudes towards the second discipline while learning programming. In future studies, the number of second disciplines can be increased to give students a choice of different problems. Also, the study can be applied to state schools in future research.

REFERENCES

Allsop, Y. (2019). Assessing computational thinking process using a multiple evaluation approach. International Journal of Child-Computer Interaction, 19, 30–55. doi:10.1016/j.ijcci.2018.10.004

Anderson, N. D. (2016). A Call for Computational Thinking in Undergraduate Psychology. Psychology Learning & Teaching, 15(3), 226–234. doi:10.1177/1475725716659252

Angeli, C., & Giannakos, M. (2020). Computational thinking education: Issues and challenges. Computers in Human Behavior, 105, 106185. doi:10.1016/j.chb.2019.106185

Armoni, M. (2011). The nature of CS in K--12 curricula. ACM Inroads, 2(4), 19–20. doi:10.1145/2038876.2038883

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12. ACM Inroads, 2(1), 48–54. doi:10.1145/1929887.1929905

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computer games about environmental science in a fifth grade classroom. Educational Technology Research and Development, 59, 765e782. http://dx.doi.org/10.1007/s11423-010-9184-z

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. Educational Technology Research and Development, 59(6), 765–782. doi:10.1007/s11423-010-9184-z

Brennan, K., & Resnick, M. (2012).Using artifact-based interviews to study the development of computational thinking in interactive media design. AmericanEducational Research Association Meeting. Vancouver, BC: Canada

Cansu, S. K., & Cansu, F. K. (2019). An Overview of Computational Thinking. International Journal of Computer Science Education in Schools, 3(1), n1.

Czerkawski, B. C., & Lyman, E. W., III (2015). Exploring issues about computational thinking in higher education. TechTrends, 59(2), 57e65. http://dx.doi.org/10.1007/s11528-015-0840-3.

Dagdilelis, V., Satratzemi, M., & Evangelidis, G. (2004). Introducing secondary education students to algorithms and programming. Education and Information Technologies, 9(2), 159-173.

DePryck, K. (2016, November). From computational thinking to coding and back. In Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality (pp. 27-29).

Grover, S., & Pea, R. (2013). Computational thinking in K-12, a review of the state of the field. Educational Researcher, 42(1), 38e43.

Gülbahar, Y., Kert, S. B., & Kalelioğlu, F. (2019). Bilgi işlemsel düşünme becerisine yönelik öz yeterlik algısı ölçeği: geçerlik ve güvenirlik çalışması. [Self-efficacy perception scale for computational thinking skill: validity and reliability study] Türk Bilgisayar ve Matematik Eğitimi Dergisi, 10(1), 1-29.

Haseski, H. I., Ilic, U., & Tugtekin, U. (2018). Defining a New 21st Century Skill-Computational Thinking: Concepts and Trends. International Education Studies, 11(4), 29-42.

Hemmendinger, D. (2010). A plea for modesty. ACM Inroads, 1(2), 4–7. doi:10.1145/1805724.1805725

Howland, K., Good, J., & Nicholson, K. (2009, September). Language-based support for computational thinking. In 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (pp. 147-150). IEEE.

International Society for Technology in Education & Computer Science Teachers Association. (2011). Operational definition of computational thinking for K-12 education. Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011, April). Computational thinking patterns. In Paper presented at the annual meeting of the American educational research association. New Orleans, LA.

Israel, M., Pearson, J., Tapia, T., Wherfel, Q., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. Computers & Education, 82, 263e279.

Kafai, Y., & Burke, Q. (2014). Connected code : why children need to learn programming. MIT Press.

Kale, U., Akcaoglu, M., Cullen, T., Goh, D., Devine, L., Calvert, N., & Grise, K. (2018). Computational What? Relating Computational Thinking to Teaching. TechTrends, 62(6), 574-584.

Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review.

Kazimoglu, C., Kiernan, M., Bacon, L. & MacKinnon, L., 2012, Learning Programming at the Computational Thinking Level via Digital Game-Play, Procedia Computer Science, 9, 522-531.

Kurilovas, E., & Dagiene, V. (2016). Computational thinking skills and adaptation quality of virtual learning environments for learning informatics. International Journal of Engineering Education, 32(4), 1596-1603

Kong, S.-C., Lai, M., & Sun, D. (2020). Teacher development in computational thinking: Design and learning outcomes of programming concepts, practices and pedagogy. Computers & Education, 151, 103872. doi:10.1016/j.compedu.2020.103872

Kwon, D. Y., Kim, H. S., Shim, J. K., & Lee, W. G. (2012). Algorithmic bricks: a tangible robot programming tool for elementary school students. Education.IEEETransactions, 55(4), 474e479

Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). Computational Thinking Is More about Thinking than Computing. Journal for STEM Education Research, 3(1), 1–18. doi:10.1007/s41979-020-00030-2

Lu, J. J., & Fletcher, G. H. (2009, March). Thinking about computational thinking. In ACM SIGCSE Bulletin (Vol. 41, No. 1, pp. 260-264). ACM.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. ACM Transactions on Computing Education (TOCE), 10(4), 16.

Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2019). Development of computational thinking, digital competence and 21stcentury skills when learning programming in K-9. Education Inquiry, 11(1), 1–17. doi:10.1080/20004508.2019.1627844

Nowak, M. A., Komarova, N. L., & Niyogi, P. (2002). Computational and evolutionary aspects of language. Nature, 417(6889), 611-617.

Romero, M., Lepage, A., & Lille, B. (2017). Computational thinking development through creative programming in higher education. International Journal of Educational Technology in Higher Education, 14(1), 42.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.(2013). URL https://eprints. soton. ac. uk/356481.

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, 351e380.

Threekunprapa, A., & Yasri, P. (2020). Unplugged Coding Using Flowblocks for Promoting Computational Thinking and Programming among Secondary School Students. International Journal of Instruction, 13(3), 207–222. doi:10.29333/iji.2020.13314a

Tran, Y. (2018). Computational Thinking Equity in Elementary Classrooms: What Third-Grade Students Know and Can Do. Journal of Educational Computing Research, 57(1), 3–31. doi:10.1177/0735633117743918 Wing, J. (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 A: Mathematical, Physical and Engineering Sciences, 366(1881), 3717-3725.