imagine learning

WHITE PAPER

Computational Thinking in K–12 Computer-Science Education:

Fostering Competencies for the Digital Age

“As more U.S. states move toward requiring computer science in Grades K–8 and offering it as an elective in Grades 9–12, it is estimated that nearly every child in the United States will take computer-science classes in the next decade”

. . .

(Tissenbaum & Ottenbreit-Leftwich, 2020)

Introduction

STEM, the intersection of science, technology, engineering, and mathematics, emphasizes a multidisciplinary approach to solving problems (DeCoito, 2014). As the U.S. STEM workforce has innovated to solve problems, it has contributed to improvements in U.S. living standards, economic prosperity, and global competitiveness. The STEM workforce, consisting of a wide variety of occupations across industries (Israel et al., 2015), has grown at more than four times the rate of total employment (Hossain & Robinson, 2012). According to the U.S. Bureau of Labor Statistics (2017), there were already nearly 8.6 million STEM jobs in May 2015, and that trend has continued to grow. Computer-related jobs experienced the largest gains among the STEM occupations, making up nearly 45 percent of STEM employment. As the world becomes more technologically developed, employment in computer-related occupations is projected to increase.

STEM education, particularly computer-science education, is fundamental to preparing the next generation of skilled workers. Computer-science education is well established at the postsecondary level (Bottoms & Sundell, 2016), but there are persisting barriers that make it difficult to meet the rising workforce demand. The current pipeline of students pursuing STEM professions is thought to be inadequate (Hossain & Robinson, 2012). Additionally, according to the Pew Research Center (2021), the higher-education pipeline suggests that lack of diversity is a persistent issue, especially in fields such as computing. Black and Hispanic degree recipients continue to be underrepresented. Women are underrepresented among graduates in computer science. These underrepresented groups also earn less compared to their counterparts in the STEM workforce. Broadening participation in STEM is necessary to foster innovative capacity and build a robust workforce that can effectively utilize technology in global applications (NSB, 2021).

This urgency to prepare today's students to become tomorrow's creators and innovators of technology has resulted in an increased pressure to expand access to computer science across the K–12 curriculum (Yadav et al., 2016). As more U.S. states move toward requiring computer science in Grades K–8 and offering it as an elective in Grades 9–12, it is estimated that nearly every child in the United States will take computer-science classes in the next decade (Tissenbaum & Ottenbreit-Leftwich, 2020). Although this rapid integration of computer science in the K–12 system is a step in the right direction, it is not without challenges.

This urgency to prepare today's students to become tomorrow's creators and innovators of technology has resulted in an increased pressure to expand access to computer science across the K–12 curriculum

• • •

(Yadav et al., 2016)

Access to computer-science content is only part of the solution. Student awareness, exposure, and interest in computer science are also essential. There is a concern that inadequate exposure to STEM in earlier grades will impact students' course choices in high school, and subsequently in their postsecondary and career decisions (DeCoito, 2014). There is a low level of interest among middle schoolers for participating in STEM-related career academics when compared to courses in other subject areas (Collins & Jones Roberson, 2020; Hossain & Robinson, 2012). In high school, participation in Advanced Placement Computer Science courses is low overall, and dramatically lower among Blacks and Hispanics (Wang & Moghadam, 2017). Alternatively, Lee (2015) found that students who took more units in computer science were significantly more likely to choose STEM majors at the postsecondary level. This research indicates that enhancing the quality of computer-science education, and motivating students to pursue STEM education and career choices throughout their education, is important for developing a robust pipeline of diverse STEM career aspirants in college, who will be prepared with skills the 21st century demands.

Teachers encounter challenges teaching computer science. First, teachers need additional training and resources to successfully integrate computer-science instruction (Yadav et al., 2016). Second, there are not enough teachers prepared to teach computing, due to teacher certification and training issues (Computer Science Teachers Association [CSTA], 2013). Third, although educators recognize the conceptual links between the various domains of STEM knowledge, some find it a challenge to meaningfully integrate STEM content into their instruction (Thomasian, 2011). For example, computer science is sometimes confused with other disciplines such as educational technology, computer or digital literacy, information- technology (IT) fluency, and computational literacy (Bottoms & Sundell, 2016).

The main goal of STEM education is to help students become proficient in STEM content while developing 21st-century skills such as critical thinking, problem solving, creativity, and collaboration (DeCoito, 2012). Computer science, a component of STEM, is "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications and their impact on society" (CSTA, 2011, p. 1), but it offers much more than simply teaching students to build computers, write code, and manage data. In the process of learning subject-matter content, students develop computational thinking, which encompasses skills such as problem solving, creative thinking, confidence, and persistence (Bottoms & Sundell, 2016; Burbaite et al., 2018). These valuable qualities are highly transferable, empowering students to succeed in school, and advance efficiency and productivity in every discipline, industry, and profession as students progress in their chosen careers and contribute to the labor market. This paper explores essential STEM skills and describes how these skills are integral to computer-science education. Specially, in Section 1, the Four Cs of STEM are defined, with discussions of computational thinking and creative problem solving as foundational processes for computer-science instruction. Section 2 describes the integration of computational thinking and creative problem solving with instructional approaches for K–12 educators.

The main goal of STEM education is to help students become proficient in STEM content while developing 21st-century skills such as critical thinking, problem solving, creativity, and collaboration

• • •

(DeCoito, 2012)

Part 1

The Four Cs Of STEM: 21st-Century Dispositions

As discussed in the introduction, the number of jobs requiring computer-science skills has grown significantly, and is projected to continue growing as technological advances are rapidly changing how we interact with our world. Workforce skills have changed dramatically in the 21st century. Jobs with more "routine" work have decreased, and have been replaced with jobs that require adaptability for nonroutine work and analytic and interactive communication skills (NEA, n.d.). In response to changes in demand for skilled labor, the National Education Association (n.d.) identified the Four Cs of STEM as essential for all students to acquire. Specifically, the Four Cs include: critical thinking, communication, collaboration, and creativity (defined in Table 1). These skills are critical for STEM-related jobs, and critical thinking and creativity are particularly applicable to computer-science education.

Table 1: Four Cs of STEM

Four Cs	Definition	Importance
Critical Thinking	Critical thinking involves reasoning effectively, using systems thinking, making judgments and decisions, and solving problems.	Learning requires critical thinking. Critical thinking leads students to develop other skills, such as improved thought processing and higher levels of concentration.
Communication	Communication is the ability to articulate thoughts, listen and extract meaning, and interact in diverse environments.	Students must be able to clearly communicate, and to effectively analyze and process various forms of communication for success in school and careers.
Collaboration	Collaboration is the ability to work effectively with others to achieve common goals.	Considering the complexity of issues and challenges companies, institutions, and governments face, collaboration with diverse individuals is critical for identifying relevant solutions and making informed decisions.
Creativity	Creativity encompasses exploring and analyzing a wide range of ideas and perspectives, generating original and inventive solutions, viewing failure as an opportunity to learn, and turning ideas into tangible solutions.	The rapid pace of change in the 21st century requires rapid adaptation and continual innovation. Students will need to know how to create and innovate to successfully address workforce and social challenges.

Computational Thinking

Critical thinking is the ability to reason effectively, use systems thinking, make judgments and decisions, and solve problems. Computational thinking, a problem-solving approach often used by computer scientists, is synonymous with critical thinking (Noonoo, 2019). In our data-driven age, managing information effectively and efficiently using technologies is important (Shute, Sun & Asbell-Clarke, 2017). Individuals who possess critical- and computational-thinking skills, and are engaged in the workforce, increase their country's competitiveness in the global economy.

Several definitions have been proposed for computational thinking, but an exact definition remains vague (Barr et al., 2011; Grover & Pea, 2013). Some researchers explicitly linked computational thinking to programming skills, defining computational thinking as "...students using computers to model their ideas and develop programs" (Israel et al., 2015, p. 264). Understanding what computational thinking is not can help clarify what it is. For example, coding or programming skills, by themselves, are too limiting a representation of computational thinking (Shute et al., 2017).

Computational thinking stems from the constructivist work of Seymour Papert (1980, 1991), and was coined a term by Jeannette Wing (2006). Wing's seminal article provided a connection between humans and computers, and her definition is the most widely used. She explained that computational thinking involves "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006, p. 33). She clarified that computational thinking is not synonymous with thinking like a computer; rather, it involves being engaged in cognitive processes as one is solving problems creatively and efficiently.

Other definitions of computational thinking exist, and they vary in how computational thinking is operationalized. In the context of K–12 education, the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTE) collaborated with the education community to develop an operational definition that would resonate with educators. They defined computational thinking as a problem-solving process that includes the following attributes:

- "Formulating problems in a way that enables us to use a computer and other tools to help solve them,
- Logically organizing and analyzing data,
- Representing data through abstractions, such as models and simulations,
- Automating solutions through algorithmic thinking (a series of ordered steps),
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources, and
- Generalizing and transferring this problem-solving process to a wide variety of problems" (Barr et al., 2011, p. 21).

Creative Problem Solving

Creative problem solving is closely associated with computational thinking. Research has suggested that there are many similarities between computational thinking and creative problem solving (Labusch et al., 2019). Applying computational-thinking practices helps students develop problem-solving and creative-thinking skills that are needed to formulate and solve real-world problems in a digital era (Kong, 2019).

Creative problem-solving skills are developed as one is confronted with a barrier, and is then motivated to apply requisite skills, knowledge, and understanding to seek and explore feasible solutions (Hatch, 1988; Labusch et al., 2019). As with computational thinking, the creative-problem-solving process is often described as having seven stages:

- "Recognize or identify the problem,
- Define and represent the problem mentally,
- Develop a solution strategy,
- Organize his or her knowledge about the problem,
- Allocate mental and physical resources for solving the problem,
- Monitor his or her progress toward the goal, and (g) evaluate the solution for accuracy" (Pretz et al., 2003, p. 3–4).

Problems vary in scope, but when one is engaged in ill-structured problems, they operate at high levels of thinking and reasoning in order to find creative solutions.

Therefore, creative thinking is a critical component of problem solving. Creativity is ignited at the beginning of the problem-solving process. Originality and task appropriateness must be exhibited simultaneously (Patston et al., 2021) as the problem is formulated and a solution is found. Recognizing and identifying a problem is important, because how a problem is solved depends on the actual problem (Labusch et al., 2019). Kong (2019) postulated that "problem formulation" should be a component of computational thinking practices because formulating a problem is often more vital than its solution. According to Kong, students demonstrate their creativity as they raise new questions and possibilities in the process of formulating problems.

Creativity is utilized throughout the computational-thinking process. As solutions are creatively explored, automation is applied to achieve efficiency and effectiveness. Identifying and implementing the most efficient solution increases the likelihood of the solution being generalized and transferred to other real-world problems. Ultimately, a goal of computational-thinking practices is to develop creative problem solvers.

Part 2

Computational and Creative Thinking in Computer Science

Correspondingly, an aim of computer-science education is to develop students' ability to engage in computational thinking and creative problem solving (Burbaite et al., 2018). Cognitive processes involved in computational thinking are integral to computer-science concepts and approaches. Computational-thinking elements most referenced in computer science are decomposition, abstraction, algorithms, and debugging (Shute et al., 2017). Decomposition involves breaking down a problem into manageable units. Abstraction entails modeling the main facets of complex problems. Algorithms refer to the design of logical and ordered instructions that are used to execute a solution to a problem. Debugging occurs when a solution does not function as it should; the process involves detecting and fixing errors. These cognitive processes are closely related to fundamental programming concepts used in the field of computer science.

Computer Science Curricula: Programming, Robotics, and Game Design

The curricula presented in programming, robotics, and game design each emphasize different elements of computational thinking, and therefore can be utilized to foster computational thinking (Shute et al., 2017) as well as creative problem solving. Programming is often used to promote computational-thinking skills and creative problem solving because writing and using efficient programs entails abstraction, generalization, and debugging. Students apply these processes by determining a goal to achieve, identifying sub-goals and steps to achieve their goal, and proposing efficient solutions. The programming code is meant to be reused to solve similar problems, with minor adjustments. Also, debugging is necessary to test the accuracy and efficiency of the program. The acquisition of programming concepts and practices through programming is considered as the most effective way to learn computational thinking (Kong, 2019).

Computational thinking is also used in robotics education. Students identify a problem for the robot, decompose the problem into sub-goals, and develop algorithms as a set of instructions for the robot to follow (Shute et al., 2017). Debugging is used in robotics to iteratively test and make changes as needed, skills that also utilize creativity and critical thinking.

Lastly, game design and gameplay require important components of computational thinking, such as problem decomposition, debugging, generalization, and iteration (Shute et al., 2017). In game design, players have various goals to achieve and need to develop solution plans. Plans are systematically tested to arrive at the most effective strategy to overcome the challenges in the game. Strategies previously used can be adopted to solve new problems, a process that can leverage each of the Four Cs of STEM.

Instructional and Learning Strategies

It is important to consider impactful instructional strategies that can guide students' computer-science learning. Additionally, educators should make a concerted effort to integrate computational thinking and creative problem solving as part of the computer-science curriculum (Stephenson & Malyn-Smith, 2016), as this has the potential to positively influence student engagement and motivation. This can be done by informing students about the various skills they utilize as they work on problems; integrating class discussions on computational thinking and creative problem solving to highlight how they can impact all areas of students' future studies, careers, and lives; and acknowledging progress and providing feedback to help students understand why they are developing these skills. Next, we briefly summarize three instructional strategies that can be integrated into K–12 curricula. First, computational-thinking approaches can be inserted in existing curriculum.

Lye & Koh (2014) proposed a constructionism-based problem-solving learning environment as students engage in computer-science programming activities. This framework contains the following elements: (a) an authentic problem to solve that is relevant to the learner (e.g., a game, a digital story); (b) information-processing activities (e.g., metaphor, cognitive conflict, mind mapping) to help students better grasp complex computing concepts; (c) scaffolded program construction by the teacher (e.g., the program broken down into mini-programs to make the task more manageable); and (d) reflection (e.g., self-reflection or peer reviewing).

In practice, computer-science instructional programs integrate this framework when students are given authentic tasks to perform, such as moving an object forward or moving it around an obstacle. Students are supported in learning when they are given tools to complete the tasks, including coding blocks that can be combined to complete the specified tasks. When students have completed an initial task, they can continue to learn as they complete a series of mini-tasks, which take them through initial movement to object avoidance and to looping programming. After students have practiced sequencing specific coding skills, they can then apply learning in competitions that allow them to receive feedback on their coding and to reflect on how to increase efficiency in their work. Iterating on these activities, learning and practicing coding, completing a sequence of mini-tasks for more complex programming, and then testing programming in a competitive environment exposes students to iterative design processes so integral to computer science. The learning environment, then, is designed to foster computational practices and perspectives.

Students are supported in learning when they are given tools to complete the tasks, including coding blocks that can be combined to complete the specified tasks.

Second, CT content can be presented using various approaches. Instructional software can be used to deliver instruction with a linear approach (Israel et al., 2015), where learning styles are assumed to be homogenous and all students are presented with the same instruction and problem-solving activities. Linear approaches include well-scaffolded instructional content that presents concepts repeatedly, with increasing complexity. For example, if students are learning to sequence actions, they might start with a simple task requiring them to select code for a specified sequence, code the entire sequence, check their work, fix errors, and then independently create code from scratch for a similar situation using skills learned in the initial lesson. Then, in subsequent lessons, the complexity of the sequencing requirements can increase, supporting students in acquiring skills that build sequentially.

Alternatively, open-inquiry activities can be applied as students and teachers use programming software for instructional purposes. With instructional software, students may be given project assignments that include suggested research topics and supporting documentation necessary for completing the project. With project briefs, students are required to define the problem, research possible solutions, implement solutions, and then test and submit their work for evaluation. Integrating open-inquiry projects with computer-science instruction allows students to create their own unique projects and use iterative testing strategies to develop the most effective solutions.

However, digital tools are not necessarily required to teach computational thinking. For example, Kim et al. (2013) created a paper-and-pencil programming strategy for non-computer-science majors. In classrooms, teachers can use printable worksheets as scaffolds for learning computational thinking. For example, if students are assigned an open-inquiry project, they can be given brainstorming, filtering, or decomposition worksheets or organizers to guide them in applying computational thinking.

Lastly, teachers can integrate computational thinking with project- and problem-based learning methods. These methods can be used to encourage students to take responsibility for their learning process. Students become researchers, asking important questions as they design and conduct investigations, collect and analyze data, and apply what they learned to new situations (English & Kitsantas, 2013). Structured projects or problem-based learning include well-defined goals and measures of success. These are given to students as they begin project- or problem-based learning activities. Unlike in open-inquiry approaches, projects and problems are defined in terms of what students will produce. When implemented in classrooms, the use of project- or problem-based learning encourages iterative approaches, incremental improvements to design, experimentation, and a growth mindset through a safe-to-fail approach. Further, decomposition, abstraction, brainstorming, and other supporting worksheets and tools can be provided to students, to encourage and support the explicit use of computational skills until they become implicit, internalized knowledge and practices.

Conclusion

The human mind is the most powerful problem-solving tool, but extending that power with computers and other digital tools has become an essential part of our daily life (Barr et al., 2011). Although students apply many elements of computational thinking in a variety of disciplines, it is important to systematically integrate opportunities to apply computational thinking into K–12 computer-science curricula. Applying some approaches discussed in this paper, such as open-inquiry activities and project- or problem-based learning methods, can help educational institutions better meet the demand for computer-science classes over the next decade. Accordingly, students will learn the complete set of computational-thinking skills and dispositions, and reap the full benefits that can have future societal and economic implications.

The human mind is the most powerful problem-solving tool, but extending that power with computers and other digital tools has become an essential part of our daily life

• • •

(Barr et al., 2011)

References

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology, 38*(6), 20–23.

Bottoms, G., & Sundell, K. (2016). *The future of K–12 computer science instruction.* National Association of State Boards of Education.

Burbaite, R., Drasute V., & Stuikys V. (2018). *Integration of computational thinking skills in STEM-driven computer science education.* IEEE Global Engineering Education Conference (EDUCON), April, 2018, Santa Cruz de Tenerife, Canary Islands, Spain. https://doi.org/10.1109/EDUCON.2018.8363456

Computer Science Teachers Association. (2011). *CSTA K–12 Computer Science Standards—Revised 2011.* CSTA.

Computer Science Teachers Association. (2013). *Bugs in the system: Computer science teacher certification in the U.S.* CSTA. http://www.mspnet.org/library/27951.html

Collins, K. H., & Jones Roberson, J. (2020). Developing STEM identity and talent in underrepresented students: Lessons learned from four gifted Black males in a magnet school program. *Gifted Child Today*, *43*(4), 218–230. https://doi.org/10.1177/1076217520940767

DeCoito, I. (2014). Focusing on science, technology, engineering, and mathematics (STEM) in the 21st century. *Ontario Professional Surveyor*, 34–36.

DeCoito, I. (2012). Digital games in science education: Developing students' 21st century learning skills. In Z. Karadag & Y. Devecioglu-Kaymakci (Eds.), *Proceedings of the International Dynamic, Explorative, and Active Learning (IDEAL) Conference*. Bayburt University.

English, M. C., & Kitsantas, A. (2013). Supporting student self-regulated learning in problem- and project-based learning. *Interdisciplinary Journal of Problem-Based Learning*, *7*(2). https://doi.org/10.7771/1541-5015.1339

Erhel S., & Jamet E. (2013). Digital game-based learning: Impact of instructions and feedback on motivation and learning effectiveness. *Computers & Education*, *67*, 156–167. https://doi.org/10.1016/j.compedu.2013.02.019

Evripidou, S., Georgiou, K., Doitsidis, L., Amanatiadis, A.A., Zinonos, Z., & Chatzichristofis, S. A. (2020). Educational robotics: Platforms, competitions and expected learning outcomes. *IEEE Access, 8*. https://doi.org/10.1109/ACCESS.2020.3042555

Fayer, S., Lacey, A., & Watson A. (2017). STEM occupations: *Past, present, and future (spotlight on statistics)*. U.S. Bureau of Labor Statistics. https://www.bls.gov/spotlight/2017/science-technology-engineering-and-mathematics-stem-occupations-past-present-and-future/pdf/science-technology-engineering-and-mathematics-stem-occupations-past-present-and-future.pdf

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hatch, L. (1988). Problem solving approach. In W. H. Kemp, & A. E. Schwaller (Eds.). *Instructional strategies for technology education* (pp. 87–98). Glencoe.

Hossain,. M., & Robinson, M. G. (2012). How to motivate US students to pursue STEM (science, technology, engineering and mathematics) careers. *US-China Education Review A* 4, 442–451.

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*, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022

Israel, M., Wherfel, Q. M., Pearson, J., Shehab, S., & Tapia, T. (2015). Empowering K–12 students with disabilities to learn computational thinking and computer programming. *TEACHING Exceptional Children*, *48*(1), 45–53. https://doi.org/10.1177/0040059915594790

Kazimoglu, C. (2020). Enhancing confidence in using computational thinking skills via playing a serious game: A case study to increase motivation in learning computer programming. *IEEE Access, 8*, 221831–221851. https://doi.org/10.1109/ACCESS.2020.3043278

Kennedy, B., Fry, R., & Funk, C. (2021). *6 Facts about America's STEM workforce and those training for it*. Pew Research Center. https://www.pewresearch.org/fact-tank/2021/04/14/6-facts-about-americas-stem-workforce-and-those-training-for-it/

Kenworthy, L., Kielstra, P., & Tabary, Z. (2015). Driving the skills agenda: Preparing students for the future. *The Economist Intelligence Unit*. https://static.googleusercontent.com/media/edu.google.com/en//pdfs/skills-of-the-future-report.pdf

Kim, B., Kim, T., & Kim, J. (2013). Paper-and-pencil programming strategy toward computational thinking for non-majors: Design your solution. *Journal of Educational Computing Research*, *49*, 437–459. https://doi.org/10.2190/EC.49.4.b

Kong, Siu-Cheung. (2019). Components and methods of evaluating computational thinking for fostering creative problem-solvers in senior primary school education. In: S. C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 119–141). Springer. https://doi.org/10.1007/978-981-13-6528-7_8

Labusch, A., Eickelmann, B., & Vennemann, M. (2019). Computational thinking processes and their congruence with problem-solving and information processing. In: S. C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 65–78). Springer. https://doi.org/10.1007/978-981-13-6528-7_5

Lee, A. (2015). Determining the effects of computer science education at the secondary level on STEM major choices in postsecondary institutions in the United States. *Computers & Education*, *88*, 241–255. https://doi.org/10.1016/j.compedu.2015.04.019

Lye, S. Y., & Koh, J. H. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61.

National Education Association. (n.d.). *Preparing 21 century students for a global society. An educator's guide to the "Four Cs."* https://dl.icdst.org/pdfs/files3/0d3e72e9b873e0ef2ed780bf53a347b4.pdf

National Research Council. (2010). *Committee for the workshops on computational thinking: Report of a workshop on the scope and nature of computational thinking*. National Academies Press.

National Science Board & National Science Foundation. (2021). The STEM labor force of today: Scientists, engineers and skilled technical workers. *Science and Engineering Indicators 2022*. https://ncses.nsf.gov/pubs/nsb20212.

Noonoo, S. (2019). *Computational thinking is critical thinking. And it works in any subject*. Edsurge. https://www.edsurge.com/news/2019-05-21-computational-thinking-is-critical-thinking-and-it-works-in-any-subject

Papert, S. (1991). *Situating constructionism*. In S. Papert & I. Harel (Eds.), Constructionism. MIT Press.

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

Patston, T. J., Kaufman, J. C., Cropley, A. J., & Marrone, R. (2021). What is creativity in education? A qualitative study of international curricula. *Journal of Advanced Academics, 32*(2), 207–230. https://doi.org/10.1177/1932202X20978356

Pretz, J. E., Naples, A. J., & Sternberg, R. J. (2003). Recognizing, defining and representing problems. In J. E. Davidson & R. J. Sternberg (Eds.), *The psychology of problem solving* (pp. 3–30). Cambridge University Press.

Qian, M., & Clark, K. R. (2016). Game-based learning and 21st century skills: A review of recent research. *Computers in Human Behavior, 63*, 50–58. https://doi.org/10.1016/j.chb.2016.05.023

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22,* 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Stephenson, C., & Malyn-Smith, J. (2016). Computational thinking from a dispositions perspective. https://blog.google/outreach-initiatives/education/computational-thinking-dispositions-perspective/

Tissenbaum, M., & Ottenbreit-Leftwich, A. (2020). A vision of K-12 computer science education for 2030. *Communications of the ACM, 63*(5), 42–44. https://doi.org/10.1145/3386910

Theodoropoulos, A., Antoniou, A., & Lepouras, G. (2017). Teacher and student views on educational robotics: The Pan-Hellenic competition case. *Application and Theory of Computer Technology. 2*, 4. https://doi.org/10.22496/atct.v2i4.94

Thomasian, J. (2011). *Building a science, technology, engineering, and math education agenda*. NGA Centre for Best Practices.

Wang, J., & Moghadam, S. (2017). Diversity barriers in K–12 computer science education: Structural and social. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education,* 615–620. https://doi.org/10.1145/3017680.3017734

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

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

Xu, D., Blank, D., & Kumar, D. (2008). Games, robots, and robot games: Complementary contexts for introductory computing education. *Proceedings of the 3rd international conference on Game development in computer science education*, 66–70. https://doi.org/10.1145/1463673.1463687

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education*. https://doi.org/10.1080/08993408.2016.1257418

imagine learning

imaginelearning.com
877-338-2020 • solutions@imaginelearning.com

938520456 2208