The History of Low-code and No-code Approach for Computational Thinking

Wonjin Yu[1]

[1] Learning, Design, and Technology, Purdue University

Author Note

Wonjin Yu is now at the Department of Curriculum and Instruction, Purdue University.

We have no conflicts of interests to disclose.

Correspondence concerning this article should be addressed to Wonjin Yu, 150 N. University St., West Lafayette IN 47907. Email: yu1103@purdue.edu

Abstract

In the contemporary era of digitalization, computers are widely used in our daily lives, and computer science (CS) is playing an increasingly important role in modern society. Many governments have launched initiatives to introduce CS ideas into the K-12 curriculum to extend the general analytical abilities of every learner. One area of interest is computational thinking (CT), in which students learn to use CS ideas to solve problems (Yadav et al., 2014). One means of achieving this is through programming education in K-12. However, coding-first approaches in K-12 classrooms may frustrate students (Lishinski et al., 2017). In light of this, a Low-code and No-code (LCNC) approach has been employed to introduce CT to K-12. This paper aims to identify the history of CT and LCNC and explore efforts that have been invested in increasing the effectiveness of the LCNC approach for CT.

Keywords: computational thinking, low-code and no-code, LCNC, K-12

The History of the Computational Thinking

Computational thinking (CT) refers to the extension of the basic principles of computer science (CS) to an approach to producing and thinking beyond the simple use of a computer (Yadav et al., 2014). This approach historically began with Seymour Papert's Constructionism in the mid-1900s, when he extended the constructivist paradigm that learning occurs best when learners create constructs or engage in conversations related to their creations to constructionism (Ackermann, 2001; Papert, 1980, 1991). Later, Wing (2006) led a paradigm shift in extending CT to basic analytical skills for every learner (Barr & Stephenson, 2011; Wing, 2006, 2011). In more recent years, there has been an active discussion of CT, and its definition has expanded. The Computer Science Teachers Association (CSTA) and the International Society for Educational Technology (ISTE) discussed and extended the definition of CT (Barr &

Stephenson, 2011). Recently, with the development of artificial intelligence (AI) and interest in AI education, there has been discussion on the expansion of CT to AI thinking and AI literacy (Gadanidis, 2017; Touretzky & Gardner-mccune, 2019; Zeng, 2013). The overall history of CT is summarized in Table 1.

Table 1

History of Computational Thinking Discourse

Authors	**Elements of CT**
Wing (2006)	Algorithms, Abstraction, Automation
CSTA & ISTE (2011)	Data Collection, Data Analysis, Data Representation, Problem Decomposition
Computing At School (CAS; 2014)	Logic, Algorithms, Decomposition, Patterns, Abstraction, Evaluation, Tinkering, Creating, Debugging, Persevering, Collaborating
Korkmaz et al. (2017)	Creativity, Algorithmic Thinking, Cooperation, Critical Thinking, Problem Solving

The Advent of the Low-code and No-code Approach

The term Low-code and No-code (LCNC) was originally derived from programming in the business field. LCNC in CS emerged as an alternative to traditional text-based development with the introduction of application development tools in the 1980s (Adrian et al., 2020; Wong et al., 2019). The importance and popularity of the LCNC approach have increased in more recent times because it can speed up the development process and consequently lower costs (Fryling, 2019; Hecht, 2019). Alongside this trend, the business industry predicts that more than 65% of

applications will be developed using an LCNC approach by 2024 (Wong et al., 2019). Beranic et al. (2020) determined positive effectiveness by conducting a representative usability experiment of an LCNC development tool with 113 students. Although a recent boom has been observed in LCNC in relation to programming development, the LCNC approach has been around for a long time in the education field.

The Low-Code and No-Code Approach for Computational Thinking

The Constructionism approach proposed by Papert paved the way for efforts to introduce CT ideas to K-12 learners (Ackermann, 2001; Papert, 1980, 1991). As a Low-code (LC) approach, Papert initiated an effort to integrate an educational robot named Turtle with LOGO, a programming language devised by Seymour Papert, Cynthia Solomon, and Wally Feuzeig in 1967 (Stager, 2016). He was able to teach mathematics to learners in a way that reduced the burden associated with abstract thinking.

Scratch is the world's most widely used LC block-based programming educational tool. It was designed and developed in 2006 by the MIT Media Lab led by Mitchel Resnick (Olabe et al., 2011). Learners can learn the principles of CT with little-or-no knowledge of general programming language by simply dragging and dropping command blocks into place. This can help lower the psychological barrier that learners may experience when they progress to learn text-based programming languages such as C and Python. In addition, it can reduce the cognitive load exerted on learners during the process of learning to code (Scherer et al., 2020). In addition, since these tools support connection with various educational robots and sensor boards, such as LEGO Mindstorms, micro:bit, and so on, it helps to externalize the learner's complex CT mental process. The learners can check the program by operating the external educational robot (Ainsworth, 2006; Ainsworth et al., 2011; Schmidgall et al., 2019).

Parham-Mocello et al. (2019) introduced story programming and an unplugged approach to teaching CT ideas to K-12 learners via a No-code (NC) approach. They conducted an experiment to measure the effectiveness of using storytelling in the CS orientation course. They confirmed that delayed coding with a storytelling approach was more effective than the coding-first approaches in terms of performance. It was also found to be a better way of motivating students who had little interest in coding, such as female students (Parham-Mocello et al., 2019). Additional tools have been developed and utilized for experiencing and learning the basic principles of CT, and AI education has increasingly adopted approaches that involve machine learning without separate coding. For example, the Google Teachable Machine in Experiments with Google project allows learners to experience and learn the principles of machine learning that recognize images without additional coding (Experiments with Google, n.d.).

An additional element of the NC approach is unplugged learning, which is an approach to learning the principles of CS through physical activities, games, and puzzles that do not require access to a device such as a computer (Bell et al., 2012). Significant studies have highlighted the potential benefits of this approach, and researchers have found that it can reduce the cognitive load exerted on the learner and help them connect concrete analogies to abstract concepts via programming (Brackmann et al., 2017; Threekunprapa & Yasri, 2020). These approaches can be particularly effective for K-12 learners who are not yet accustomed to operating devices, such as computers. In addition, it can serve as a bridgehead for programming education and represent a viable approach to CT for schools that have limited access to resources.

Conclusion

The LC and NC approaches represent promising methods of enhancing and improving access to CT for beginner-level learners. From Papert's Constructionism and Logo with Turtle in the mid-1900s to modern Scratch, and Google's AI educational module, LCNC approaches to CT for beginner-level learners have become increasingly popular. In addition, due to the recent LCNC boom in the business field of computer science and engineering, more active discussions about LCNC can pave the way for a better CT education in the future.

References

Ackermann, E. (2001). Piaget's constructivism, Papert's constructionism: What's the difference. *Future of learning group publication*, *5*(3), 438.

Adrian, B., Hinrichsen, S., & Nikolenko, A. (2020, July). App development via low-code programming as part of modern industrial engineering education. *In International Conference on Applied Human Factors and Ergonomics* (pp. 45-51). Springer, Cham.

Ainsworth, S. (2006). DeFT: A conceptual framework for considering learning with multiple representations. *Learning and instruction*, *16*(3), 183-198.

Ainsworth, S., Prain, V., & Tytler, R. (2011). Drawing to Learn in Science. *Science*, *333*(6046), 1096-1097.

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.

Bell, T., Rosamond, F., & Casey, N. (2012). Computer science unplugged and related projects in math and computer science popularization. *In The multivariate algorithmic revolution and beyond* (pp. 398-456). Springer, Berlin, Heidelberg

Beranic, T., Rek, P., & Heričko, M. (2020). Adoption and usability of low-code/no-code development tools. *In Central European Conference on Information and Intelligent Systems* (pp. 97-103). Faculty of Organization and Informatics Varazdin.

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017, November). Development of computational thinking skills through unplugged activities in primary school. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 65-72).

Experiments with Google. Retrieved from https://experiments.withgoogle.com/collection/ai

Fryling, M. (2019). Low code app development. *Journal of Computing Sciences in Colleges*, *34*(6), 119-119.

Gadanidis, G. (2017). Artificial intelligence, computational thinking, and mathematics education. *The International Journal of Information and Learning Technology*.

Hecht, L. E. (2019). Low-code platform adoption gets a boost from digital transformation. *The New Stack*.

Lishinski, A., Yadav, A., & Enbody, R. (2017, August). Students' emotional reactions to programming projects in introduction to programming: Measurement approach and influence on learning outcomes. *In Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 30-38).

Olabe, J. C., Olabe, M. A., Basogain, X., & Castaño, C. (2011). Programming and robotics with Scratch in primary education. Education in a technological world: Communicating current and emerging research and technological efforts, 356-363.

Parham-Mocello, J., Erwig, M., & Dominguez, E. (2019, October). To code or not to code? Programming in introductory CS courses. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 187-191). IEEE.

Papert, S. (1980). " Mindstorms" Children. Computers and powerful ideas

Scherer, R., Siddiq, F., & Viveros, B. S. (2020). A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Computers in Human Behavior*, *109*, 106349.

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.

Touretzky, D. S., & Gardner-McCune, C. (2022). Artificial intelligence thinking in k-12. *Computational Thinking Education in K-12: Artificial Intelligence Literacy and Physical Computing*, 153-180.

Stager, G. S. (2016). Seymour Papert (1928–2016). *Nature*, *537*(7620), 308-308.

Schmidgall, S. P., Eitel, A., & Scheiter, K. (2019). Why do learners who draw perform well? Investigating the role of visualization, generation and externalization in learner-generated drawing. *Learning and Instruction*, *60*, 138-153.

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

Wing, J. (2011). Research notebook: Computational thinking—What and why. *The link magazine*, *6*, 20-23.

Wong, J., Driver, M., & Vincent, P. (2019). Low-code development technologies evaluation guide. Retrieved from https://www.gartner.com/en/documents/3902331

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends*, *60*(6), 565-568.

Zeng, D. (2013). From Computational Thinking to AI Thinking [A letter from the editor]. *IEEE Intelligent Systems*, *28*(06), 2-4.