Exploring Patterns and Challenges of Computational Thinking Practices

Hyunchang Moon
College of Education
Texas Tech University
3002 18th Street Lubbock, TX
hyunchang.moon@ttu.edu

Jongpil Cheon
College of Education
Texas Tech University
3002 18th Street Lubbock, TX
jongpil.cheon@ttu.edu

Kyungbin Kwon
School of Education
Indiana University
201 N. Rose Avenue Bloomington, IN
kwonkyu@indiana.edu

Keywords: computational thinking, programming, computational practice, instructional design, Scratch, block-based programming

Abstract

Although the process of thinking and learning in the computational practices are important, little research has been conducted. This study explored undergraduate students' patterns and challenges of computational thinking practices in an online environment. Qualitative data were collected from Scratch coding journals. The results revealed that undergraduates' reactions towards their successful programming experiences differed. No incremental and iterative computational process were identified when sequential directions were provided to complete unfinished tasks. Most students tested to see if their programming scripts were working and debugged errors by themselves. Few students programmed the tasks using other resources or without any references. The participants created their own codes for simpler tasks, whereas they reused and remixed other sources for complex concepts. When they mastered computational concepts, the codes associated with the concepts were easily reused and remixed. Practical implications were further discussed.

Introduction

Our constantly evolving world requires education to prepare students for success in school and life experience. Aa literacy is needed for everyone, 4 Cs–critical thinking, collaboration, communication, and creativity–have been identified as the core skills of 21st century learning for every student. Today's rapid advances in computing technology is calling for a new core skill to succeed in the digital age. The ability to expand the horizon of human thoughts and mind with computing power has become an integral part of our life and work. Accordingly, it has been argued across the globe that thinking computationally is another fundamental skill for everyone not just for computer scientists. Computational thinking (CT) has accordingly been gaining increasing attention from researchers, educators, and policy makers.

CT is one of the emerging problem-solving skills that must be acquired by the new generations of students to participate in our digital-based world (Barr, Harrison, & Conery, 2011). CT is defined as "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" (Wing, 2011, p.1). By using CT as a method for problem solving, students can actively participate in the computing world. CT is a broad topic that can be applied in various fields to meet the explosive demand for new problem-solving skills. CT can be fostered through the connection of existing standards, including the Common Core State Standards (CCSS) and Next Generation Science Standards (NGSS). At the same time, we need to ensure that all students have the equal opportunity to learn a complete set of CT through a stand-alone course, as well as being applicable to other fields.

The rationale for teaching computer science (CS) to students is far beyond career development. CS education is about giving more thinking knowledge, skills, and attitudes which help students become active in the digital world as active creators rather than passive consumers. Research on embedding CT in K-12 subject areas have found that students exposed to CT showed significant improvements in their problem-solving and high-order thinking skills (Calao, Moreno-León, Correa, & Robles, 2015; Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009). Although, to date, much effort has been devoted to teaching CT competencies in K-12 classrooms, stand-alone CS courses are implemented predominantly at the secondary education to teach the CT concepts and practices (Yadav, Gretter, Hambrusch, & Sands, 2016). Recent changes in the CS education system have led to college students who are currently not receiving such educational benefits. Similarly, most of past research studies have been conducted in the contexts of K-12 education, which necessitates research in a more diverse levels of educational settings.

CT has some similarities to other higher-order thinking such as critical thinking, mathematical thinking, and engineering thinking. However, the difference is that those thinking frameworks do not include a computational process that refer to the domain knowledge, skills, and dispositions in computer science and related fields. The CT framework has been built using several concepts of cognitive thinking such as analytical, logical, creative, or problem-solving process (Aho, 2012; Barr, Harrison, & Conery, 2011; CSTA & ISTE, 2011; NCR, 2010; Wing, 2011). Although there is little agreement about what CT encompasses, when it comes to defining CT, three-dimension framework has been widely accepted in recent years. According to Brennan and Resnick's new framework for constructing CT (2012), CT involves three key dimensions such as computational concepts (i.e., sequences, loops, events, parallelism, conditionals, operators, and data), computational practices (i.e., being incremental and iterating, testing and debugging, reusing and remixing, abstracting and modularizing), and computational perspectives (expressing, connecting, and questioning).

Several studies have examined the effectiveness of the CT instructions on student learning in K-12 classrooms (e.g., Chen, Shen, Barth-Cohen, Jiang, Huang, & Eltoukhy, 2017; Denner, Werner, Campe, & Oritiz, 2014; Doleck, Bazelais, Lemay, Saxena, & Basnet, 2017; Grover, Pea, & Cooper, 2015; Moreno-León, Robles, & Román-González, 2015; Román-González, Pérez-González, & Jiménez-Fernández, 2017). Many studies that used the Brenna and Resnick's framework evaluated learning in computational concepts only, while few included computational practice and perspectives of the CT constructs. Although the process of thinking and learning in the CT practices are important, little research has been carried out in accordance with the practice components presented in the framework–being incremental and iterating, testing and debugging, reusing and remixing, abstracting and modularizing. Also, Brenna and Resnick (2012) suggested the assessment approaches (e.g., artifact-based interviews and design scenarios) to CT practices, but the artifact-based interviews have limitations in terms of the time efficiency of collecting data and the dependency of participants' memory. The pre-determined design scenarios are also time consuming and may not be connect to a learners' interests. Thus, there remains a need for studies that further investigate the practical approach to assessing CT practices and provide data offering insight into understanding the development of CT practice in while creating a project to solve problems in online higher education course.

Purpose of the Study

We are still at an early stage regarding the understanding of computational practice development to prepare students for 21st century problem-solving. One way to advance in this area is to determine the current status of CT practices in block-based programming activities; there is a need to identify particular CT practice's patterns and challenges students have in online programming activities. The first purpose of the study was to explore the patterns undergraduate students had while creating their Scratch projects. The second purpose of the study was to investigate in which areas undergraduates was having difficulties with CT practice based on the framework proposed by Brennan and Resnick (2012). The following four research questions guided the study:

(a) What patterns and challenges do undergraduates have in the process of being incremental and iterative?,
(b) What patterns and challenges do undergraduates have in the process of testing and debugging?,
(c) What patterns and challenges do undergraduates have in the process of reusing and remixing?, and
(d) What patterns and challenges do undergraduates have in the process of abstraction and modularizing?

Theoretical Framework

Brennan and Resnick's (2012) CT involves three aspects: computational perspective, computational practice, and computational concepts. Among the three dimensions, this study focused on the computational practice to uncover the patterns and challenges undergraduate students have in their computational thinking process while creating their Scratch projects. The computational practices are defined as the practices a student develop as they program. Four main components of practices are being incremental and iterative (e.g., Developing solutions step by step), testing and debugging (e.g., Finding strategies for solving problems), reusing and remixing (e.g., Building new solutions on existing works or ideas), and abstracting and modularizing (e.g., Modeling complex systems with simple elements). The assessment for the computational practices should involve examining processes, which provide opportunities to evaluate how their thinking and learning develop over time.

Method

Given the nature of our research questions, the study adopted a qualitative research design to yield an in-depth and comprehensive analysis. The four research questions focused on exploring the patterns and challenges of CT practices from undergraduates. Therefore, through reflective coding journals, we intended to explore not only participants' processes of CT practice in developing their Scratch programming projects but also participants' perceptions of the challenges that novice learners encountered in a context of developing their computational practices.

Participants

Participants were 96 undergraduate students who were enrolled in the "Computing and Information Technology" course from a large public, southwestern university in the spring semester of 2019. The participants had enrolled from varied majors, were of various ages, and were both male and female. The students learned a set of the core knowledge and skills that shape the landscape of computer science, represent information digitally, and create block-based programs to solve problems.

Context

The course was completely online and delivered via a web-based learning management system for providing course instructions. It focused on a set of fundamental competencies in computer science. The course was also designed to provide students with programming experiences using Scratch 3.0 which is a block-based programming language developed by MIT Media Lab to makes it easier to create and share their programming projects. Scratch is intended to be used in an introductory programming course for people of all ages and across disciplines (Resnick et al. 2009), and it offers editors for both online and offline. The participants were asked to perform programming tasks using Scratch. Out of 14 modules, there was a total of eight modules related to Scratch programming, and the tasks were to complete pre-designed and partially-finished Scratch projects with a set of requirements.

Data collection and analysis

The coding journal helped capture learners' thinking process, task completion steps, or their feeling. The structural coding journal comprised four reflective questions about their programming process and experiences as they created each Scratch programming project. Scratch coding journals asked the participants to share their programming experiences with reflective writing in responses to four open-ended questionnaires: (a) Describe how you created the quiz show in detail, (b) Describe what worked well during programming, (c) Describe how you tested to see if the quiz show was working and how you fixed issues, and (d) Describe how did you adapt a sample project to make it your own quiz show. Inductive, thematic analysis was conducted with 85 responses of the participants (89% response rate) to the open-ended question in order to obtain deeper insights on the development of CT practice. The authors organized the data and then coded Scratch coding journals following the three-step guideline of Miles and Huberman (1994): data reduction, data display, and conclusion drawing and verification.

Findings

Researchers identified and coded significance statements pertaining to patterns and challenges in the coding journal responses. Codes were applied and revised as needed by combining and eliminating codes for parsimony. The final number of codes were counted for each category–*being incremental and iterating, testing and debugging,* *reusing and remixing, abstracting and modularizing*. According to Scratch coding journal responses, undergraduates' reaction towards their programming experience considerably differed. The results are presented in the order of the research questions.

RQ1. What patterns and challenges do undergraduates have in the process of being incremental and iterative?

In the process of being incremental and iterative, most of students followed the sequences of the task requirements in the module. Although no iteration or incremental programming process were identified, a third of students designed their own themes or plots before programming with Scratch. Another 30% of students started programming after reviewing the sample provided in the module or understanding the task requirements. Because CT involves a whole series of process of solving a problem through identifying the problem, designing the project, and then coding algorithms, it needs an instructional design where students can go through the thinking and learning process for each necessary step. Also, since most problems cannot be solved all at once, it is important to have an iterative process to develop solutions with new ideas and approaches. There were limitations in understanding this process the task analyzed in this study did not require such iterative process.

RQ2. What patterns and challenges do undergraduates have in the process of testing and debugging?

Next, students should deal with various errors that occur during programming. It is important to develop strategies for handling problems. In the coding journal, we analyzed various testing and debugging strategies developed by learners through trial and error. In the process of programming, parallelism was the most challenging concept for testing and debugging, followed by variable, conditional, and operators. 90% of the students tested to see if their programming codes were working and debugged errors by themselves. Only 10 % of them used other tutorials or consulted with other students. As testing and debugging strategies, instant testing (e.g., testing block by block, comparing after with before) and debugging (e.g., fixing an error instantly, comparing with the sample project) was the highest frequent approach, followed by persistent approach (e.g., keep testing and debugging until it is fixed). The other strategies were consulting with peers, referring other tutorials, and taking notes to fix.

RQ3. What patterns and challenges do undergraduates have in the process of reusing and remixing?

Programming based on the work of others is part of computational thinking. Students can improve problem-solving skills by make their own based on others' works, rather than creating something new from scratch. In terms of reusing and remixing, 70% of the students adapted the sample project provided in the module to make it their own project, and the others programmed the tasks using other resources or without any references. Half of them customized appearance changes only (i.e., sprites or backgrounds), and the other half adapted codes (i.e., variables, if/else, operators) as reference. Most students followed the overall guidance and direction sequences. Students created their own codes (e.g., texts, themes, characters, backgrounds), whereas they reused and remixed other sources for a starter or complex concepts. The codes for simple tasks (e.g., changing appearance, adding animation effects) is easily reused because they were already familiar with how to code in the previous lessons.

RQ4. What patterns and challenges do undergraduates have in the process of abstraction and modularizing?

Lastly, abstracting and modularizing is an important practice in computational thinking by identifying general principles and developing problem solutions with simple parts. Students can abstract and modularize their solutions by conceptualizing problems in simpler ways and converting the concepts into stacks of code or individual sprites. However, unfortunately, we were not able to obtain meaningful insights for the abstraction and modularization, from the coding journals because the programming projects in the study were pre-designed. To analyze responses for such unidentified practice, an ill-designed or open-ended programming project may be required, and also scratch programming artifacts should be analyzed. Further research is needed for a deeper understanding of abstraction and modularizing.

Discussion and Implications

The findings of the patterns and challenges undergraduates have in CT practice provide meaningful implications. One of the interesting findings is that the structure of programming assignments should be changed to multiple phases to make it an iterative and incremental process so that students can be more exposed to practice their iterative process. The tasks should require learners to build their solution in multiple phases.

For testing and debugging, it also should provide students with various debugging strategies and resources because self-testing and self-debugging would be limited as the difficulty of programming increases. Furthermore, the effective methods of testing and debugging should be provided especially for an online environment. Lastly, some detailed guides for customization will be necessary depending on the instructional purposes. For instance, in the early stages, students could be asked to make changes on only appearance and move to changes on codes or structure, and then to guide them through all changes to the final task.

For using and remixing, sufficient resources should be provided for learners to explore many resources, which could affect more creative and diverse learning outcomes. Reusing and remixing activities offer code reading opportunities, but also requires education in ownership and copyright. More importantly, while learning CT through programming is important to assess the results of programming, it is also important to provide an opportunity to think about why the codes need to be used. Such reasoning questions will ultimately help students improve their computational thinking for problem solving.

This study explored the patterns and challenges undergraduates had during their CT process while creating their Scratch projects. We discussed some practical suggestions to provide quality CT education. To improve the quality of CT education, all educators should be responsible for how to introduce and effectively teach CT skills in their teaching (CSTA & ISTE, 2011; Voogt, Fisser, Good, Mishra, & Yadav, 2015). This study provides educators with a better way to design computational thinking activities, especially in an online environment, and also shed new light on understanding CT practices.

Reference

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

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

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (Vol. 1, p. 25).

Calao, L. A., Moreno-León, J., Correa, H. E., & Robles, G. (2015). Developing mathematical thinking with scratch. In *Design for teaching and learning in a networked world* (pp. 17-27). Springer, Cham.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. Computers & Education, 109, 162-175.

CSTA & ISTE. (2011). *Operational definition of computational thinking for k-12 education*. Retrieved from http://csta.acm.org/curriculum/sub/currfiles/compthinkingflyer.pdf.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair programming: Under what conditions is it advantageous for middle school students?. Journal of Research on Technology in Education, 46(3), 277-296.

Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., & Basnet, R. B. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, *4*(4), 355-369.

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.

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

Miles, M. B., Huberman, A. M., Huberman, M. A., & Huberman, M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46), 1-23.

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

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. B. (2009). Scratch: Programming for all. *Commun. Acm*, *52*(11), 60-67.

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior*, *72*, 678-691.

Wing, J. M. (2011). Research notebook: Computational thinking—What and why? The Link Magazine, Spring.

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.

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2016). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235-254.