ISSN 2380-727X

PSYCHOLOGY AND COGNITIVE SCIENCES

Open Journal

Openventio PUBLISHERS

Case Study

A Case Study Exploring Pre-Service Teachers' Programming Difficulties and Strategies when Learning Programming Languages

Fatih Gok, MS; Kyungbin Kwon, PhD*

Department of Instructional Systems Technology, Indiana University, 201 N. Rose Avenue, Bloomington, IN 47405, USA

Corresponding author
Kyungbin Kwon, PhD
Assistant Professor, Department of Instructional Systems Technology, Indiana University, 201 N. Rose Avenue, Bloomington, IN 47405, USA; Tel. 812-856-8460; Fax. 812-856-8239; E-mail: kwonkyu@indiana.edu

Article information
Received: February 20th, 2020; **Revised:** March 28th, 2020; **Accepted:** April 1st, 2020; **Published:** April 7th, 2020

Cite this article
Gok F, Kwon K. Exploring pre-service teachers' programming difficulties and strategies when learning programming languages. *Psychol Cogn Sci Open J.* 2020; 6(1): 1-6. doi: 10.17140/PCSOJ-6-152

ABSTRACT

Understanding the importance of training young people, this study sought to explore the early experience of pre-service teachers in their computational practices in terms of the difficulties they faced and the strategies they used while learning how to program. Based on convenience sampling, four participants were recruited from an undergraduate course focusing on computer science education in K-12. The literature on novice programmers' difficulties and their strategies was used to establish the conceptual background for this study. We collected four semi-structured interviews with pre-service teachers, a total of five hour-long classroom observations, and 19 class activities (archival data). After conducting a content analysis, findings showed four categories in which pre-service teachers face difficulties: (a) understanding the computational concepts (semantic); (b) using the concepts inappropriately (syntax); (c) developing a program (algorithmic thinking), and (d) identifying problems (debugging). We also found five categories in which pre-service teachers overcome their difficulties: planning, using resources, seeking support, guessing and checking, and looking for visual assistance. This study emphasized that pre-service teachers encounter several difficulties in learning computational concepts through programming languages, which should be considered in pre-service teacher education.

Keywords
Computational Thinking; Computer science education; Pre-service teachers; Problem solving strategies.

INTRODUCTION

Computational thinking (CT) is defined as the ability to formulate a solution to a problem by breaking the problem down so that the solution can be automated.[1] Since Wing[1] highlighted the importance of CT for young people and suggested the benefits from using it in diverse contexts, there has been a tendency toward teaching and learning CT concepts (e.g., abstraction, decomposition, and debugging) in K-12.[2] This training of the young generation in the context of CT has been supported due to the need for people with CT skills in the diverse contexts of the modern economy in the United States (US).[3,4]

In recent years, research on theoretical and practical computer science (CS) education in K-12 has focused on defining CT concepts and providing a framework to integrate those concepts into instruction.[5,6] Gal-Ezer and Stephenson[7] suggested that teachers should have a sufficient conceptual understanding of the CT concepts. With such goal in mind, National Science Foundation (NSF) trained over 10,000 computer science teachers which was a significant effort made in the last decade.[5] In addition, researchers have also recognized the importance of spreading such training efforts of CT concepts on the development of pre-service teachers.[8-10]

Previous studies showed that there are some challenges faced by students of various age groups in learning CT concepts. For example, Basu et al[11] studied the computational and science

Openventio PUBLISHERS

domain-related challenges of 15 sixth grade students. The author's identified six categories of difficulties faced by sixth graders when learning programming. These difficulties included (a) understanding basic programming concepts (semantic difficulties); (b) using programming concepts, such as manipulating various kinds of loops; (c) developing a solution to a given computational problem step-by-step (algorithmic thinking); (d) adapting a part of an existing code to one's own code; (e) breaking the task into smaller tasks and handling these smaller tasks independently from the rest of the code; (f) detecting causes (also known as bugs) that keep the code from working properly.

Similarly, researchers identified some challenges for students in different age groups. For example, Saeli et al[12] identified various difficulty high school students encountered while learning to program, including difficulty to instruct a computer to carry out a solution, faulty assumption that a computer can understand their solutions, and tendency to have a limited point of view, which resulted in failure to find a suitable solution. The author also found that creating instructions for a computer to solve a given problem was a challenging task for high school students.[12] This challenge was also confirmed in a different study with novice programmers that the participants had difficulties while creating a computer program even though they a clear understanding of concepts (semantic) and an understanding of how to use the concepts (syntax).[13] Supporting the same idea, Lahtinen, Ala-Mutka, and Järvinen,[14] in an analysis of a survey of 559 university students and 34 teachers, indicated that students had difficulties with program construction, such as developing a program to solve a given task or dealing with bugs.

On the other hand, the literature on computer science education indicated that there were several methods that students used to overcome their difficulties in learning programming. For example, Lahtinen et al[14] identified several sources that students from different universities use to receive help while learning programming including: a programming course book, lecture notes, exercise questions and examples, example programs, pictures of programming structures, and interactive visualizations. The authors found that students perceived example programs as the most helpful for solving problems. In addition, a number of studies emphasized that visual elements help students develop a clear understanding of programs.[15,16] Similarly, several research studies indicated that providing students with a scaffolding from peers or instructors could be helpful for them to overcome their difficulties.[11,17]

Addressing students' difficulties from teachers' perspectives, several teaching techniques that help students overcome the difficulties they face while learning programming.[12] For example, algorithmic thinking defined as "a series of ordered steps"[17] to follow while solving a problem and learning a simple programming language could help students overcome their programming difficulties.

METHOD

The purpose of this qualitative study was to explore pre-service teachers' difficulties and strategies while learning programming languages. A "basic" qualitative research design[18] was used to understand how pre-service teachers interpreted their experiences with programming languages. Two research questions were posed to guide this study:

- What difficulties do pre-service teachers face when learning computational concepts through programming?
- What strategies do pre-service teachers use to overcome the difficulties they face while learning computational concepts through programming?

To answer the two research questions, we collected data from interviews, classroom observations, and archival data. Using a content analysis, we analyzed the data.

Participants

This study took place in a computer science education class at a large mid-western state university. Four participants were selected through convenience sampling.[18] Between February 15, 2017 and March 22, 2017, we collected five hours of observations, four semi-structured interviews, and archival data.

All four participants were female elementary education majors with a concentration in science and the computer education license (CEL) program. They were in their junior year and completed two computer education courses in the CEL program before taking the current introductory course to programming languages.

In this study, pre-service teachers learned two programming languages: Scratch (block-based programming) and Python (text-based programming). Scratch, with dragging and dropping features, allowed students to easily program without prior knowledge[19] and learn CT concepts without making syntax errors.[5]

Observations

A total of five hour-long observations took place at different class times. During the observation process, 12 pre-service teachers were observed while engaged in CT problem-solving practices, which involved solving problems, such as "*Write a function that takes two numbers and returns True if the first number is bigger than the second one*," within a certain time limit. The researcher took field notes during pre-service teachers' CT problem-solving practices. For example, pre-service teachers were raising questions related to the meaning of concepts. One of them asked the instructor: "*why do you use two equal signs instead of one equal sign here [in If condition]?*" The researcher also observed that pre-service teachers worked with peers collaboratively to solve the given problems.

Interviews

Semi-structured interviews were conducted with four pre-service teachers from the classroom of observation. An individual email was sent to each of twelve participant's in the computer science education class, of whom four participants volunteered to participate

in the study. The interview questions focused on the participants' experiences and their thought process of solving a given problem. To ensure the clarity of the interview questions, interview questions were sent to each participant in advance. Each interview took roughly 30-minutes. With permission from the participants, three interviews were audio-recorded and transcribed for data analysis.

Archival Data

Through class activities, nineteen worksheets were collected from students. These class activities reflected students' initial thinking processes as they worked in pairs or small groups to develop their solutions on their worksheets before attempting to solve a given problem using either Scratch or Python programming platforms. In this process, the instructor asked students to think about how they would solve the given problem, to develop their thinking processes step-by-step, and to write down their solutions on their worksheets.

Data Analysis

After transcribing the interviews, we sent the interview data to the participants for member checking to ensure the validity of the data. Two of the participants responded that their transcribed data were accurate and that they did not want to change anything. To analyze these transcripts, researchers used a content analysis approach.[20] Two coders used the participants' own words to code during the initial analysis of the interview data. After the initial coding process, researchers reached out to the sub-categories. Finally, they identified the categories for each sub-category.

In addition, researchers analyzed observation data and archival data for triangulation purposes to ensure the validity of the results.[20] Researchers followed the same procedure to analyze observational data as in the interview data analysis. To analyze the archival data, researchers looked at how CT concepts were used in class activities. The results of the archival data analysis and observation analysis supported the categories identified in the interview data analysis. There were no emerging categories that we identified from the analysis of observation data and archival data.

RESULTS

After the data analysis process, two themes including programming difficulties and strategies had been identified. The first theme, difficulties, involved four categories that pre-service teachers experienced while learning CT concepts through Scratch and Python programming languages. The second theme involved five categories that pre-service teachers used to overcome their difficulties in the learning process.

Difficulties

In answering to the first research question concerning the difficulties pre-service teachers faced while learning how to solve a given problem in Scratch and Python programming, researchers identified four categories of difficulties as follows:

Understanding the computational concepts (semantic): The participants in this study indicated a lack of understanding the computational concepts. The difficulties they faced were mainly related to the meaning of concepts and how to apply them. For example, one participant complained about the difficulties of using a computational concept appropriately and she stated that she developed an excessively long code but failed to make it shorter by using the concept loop (repetition). In the archival data analysis, the same failure of pre-service teachers where they lacked understanding of the concept loop was found. For instance, the instructor asked them to develop an algorithm for when the green flag in Scratch was clicked; the timer will count backward from 5 to 0 one by one. More than half of the students failed to say "repeat 5 times" to count down.

Similarly, one pre-service teacher was having difficulties in understanding how to use nested conditions which was another CT concept where students needed to use a conditional statement inside of another conditional statement. The archival data analysis also captured the same instance of difficulty. Three out of five groups did not understand the use of nested conditions.

Another example related to semantic difficulties was that students failed to understand why they were using particular concepts. The following statement captured the difficulties that students were experiencing:

"*I just want to know that there is a deeper understanding of why you put two equal signs or why I, like certain things are in quotation marks and some aren't.*"

Using the concepts appropriately (syntax): The analysis of interview transcripts showed that pre-service teachers sometimes had syntax errors in their code. They gave two main reasons for why they were having syntax related difficulties. First, they sometimes did not know the appropriate format for using concepts. Second, even if they knew how to use a certain concept, they failed to express it correctly. For example, one student explained her struggle with using a colon in Python programming:

"*I can write out my ideas if there's not... like I forget one colon and it like messes up the whole program because if you don't have a colon then it's not going to run properly....*"

In many cases, the students' failures in Python programming were related to syntax errors such as a missing comma or parenthesis.

Identifying problems (debugging): Another difficulty that pre-service teachers faced was to identify and fix the problems that caused errors. For example, one student stated that she was having difficulty figuring out the problems in her code. One described that:

"*I made another game where… I had four fish that were falling and two fish kept getting stuck at the bottom of the screen... I have no idea why those two are just getting stuck...*"

They often did not figure out the error messages that the python editing tool provided. The observation data suggested that it happened when pre-service teachers could not pinpoint the area of or reason for the error and examined the entire syntax instead.

Developing a program: Even though students knew what concepts they needed to use (semantic) and how to use them (syntax), they failed to combine the codes while developing a program. For example, one participant described how hard it was to transfer what she thought of as a solution to a working program:

"*I understand the whole programming thing, but like -- my basic difficulty is like using their language for it (assembling a program).*"

To ensure her intentions, a researcher asked what she meant by "understanding the whole programming." She explained that:

"*Just like understanding all of the different controls that they have and like what each thing, like different control, is used for, what its purposes...*"

Strategies

To answer the second question which is about strategies that pre-service teachers used to overcome their programming difficulties, researchers identified the following five categories:

Planning: The pre-service teachers exhibited similar approaches to solving the difficulties they faced in using the programming platforms. For example, one indicated that she planned how to solve a given problem step-by-step before she actually attempted to use the programming platform.

Similarly, using worksheets was a part of the planning process in which one of the participants from the interview shared her experience that:

"*He [the instructor gave us] the chart paper and he'll be like write out your steps…, and it's not in like coding language like we're writing in like English and math...*"

Resource use: Pre-service teachers were frequently using resources (e.g., textbooks, similar codes, and class notes) as a strategy to overcome their difficulties. Such use of resources especially occurred when they were working alone. For example, one participant talked about how she used example codes as her strategy:

"*One strategy is when I am working on my own myself as I pull out those examples [codes] that we've done in class and kind of compare my code [to see] what I'm missing here…*"

Through observing pre-service teachers, researchers also found that they frequently used Google to find answers for their difficulties.

Support: Students often sought support from peers, teaching assistants, and the instructor. They perceived peers' support as the most valuable and the quickest way to overcome the difficulties. When they faced a difficulty, they would help each other. If they could not figure out the problems, they would, as observed in the class, go to the teaching assistants or the instructor to ask for help. One of the pre-service teachers described peer support as:

"*Different people in the class understand different things, so like I could help someone with a certain aspect of it and they like understand some other.*"

Guess and check: Another strategy was guessing and checking different concepts to overcome their difficulties. If pre-service teachers could not identify the problem, they would try different code blocks in Scratch to solve it. As one of the participants said that:

"*Mostly it (way of overcoming my difficulties) was a lot of like guessing and checking to see… if this would solve the problem or not.*"

Visualization: The pre-service teachers reported this strategy when being asked about Scratch programming. The Scratch programming platform allows users to see how their block codes work, which is an intended design principle of visual coding. Another pre-service teacher stated that she observed the results of her codes in Scratch to detect where her block codes failed to run properly.

Summary

The findings of this study indicated that pre-service teachers encountered different difficulties while learning to program. These difficulties were: understanding CT concepts, using CT concepts, identifying the problem, and developing a program. In addition, pre-service teachers used several strategies to overcome their difficulties including planning the solution, using resources, receiving support from peers or the instructor, guessing and checking, and using visualization.

DISCUSSION

This case study was conducted to explore the challenges pre-service teachers encountered while learning programming languages and the strategies they used to overcome their difficulties. We found that pre-service teachers struggled in the process of learning computational concepts through using programming languages. This was not an unexpected experience of learners of programming languages. As different researchers shared similar findings, learners, including pre-service teachers, might have a lack of understanding of how to use computational concepts or when they might need to use those concepts.[11,21] Since such difficulties of learners are likely to occur in teaching computational concepts using a programming language, instructor's choice of instructional strategy with a careful consideration of different learner's need gains a higher importance. Van Merriënboer and Paas's[22] emphasis on the importance of practicing the concepts while learning how to program is still noteworthy for students and teachers as the findings of this study suggest.

The findings of this case study confirms the previous literature on the difficulties of learning computational concepts and learning how to program by using two programming languages such as dealing with syntax errors[11,14] and the struggle of initiating how to program.[13,14] However, this study contributed to the literature by identifying the difficulties of preservice teachers who were learning to teach computational concepts by using programming languages. Because of the design of this learning process, pre-service teachers used two programming languages and we were able to draw a conclusion that pre-service teachers had to pay more attention on the details such as syntax while using Python; whereas they can focus on the use of the concepts when they were solving the given task with Scratch. Several studies also confirmed that learners might have more difficulties with Python while identifying what prevents their code from running appropriately.[11,13] Therefore, the instructor who teach the text programming languages should pay more attention on practicing the use of concepts with the intended programming language.[22]

In addition, we identified that pre-service teachers used a variety of strategies such as worked examples to overcome the difficulties they faced in solving computational problems. Even though the literature suggested similar strategy uses of novice programmers,[10,11,15-17] we identified that pre-service teachers learned these strategies from the instructor or peers during the process of solving computational problems. Therefore, it would be more beneficial to learners if they were informed by the instructors in terms of what kind of difficulties they might face during the learning process, as well as the possible uses of problem solving strategies.

Implications for Research and Practice

The findings of this study encompassed several implications for both future research and practice. First, one should replicate the current study with a large number of participants and focus on one aspect of programming difficulties, such as semantic. Furthermore, future studies should examine the gender differences in programming difficulties. Regarding the implications for practice, instructors should guide students in terms of how to use strategies and give them more time to practice. Based on the difficulty, instructors can provide tailored support accordingly.

LIMITATIONS

In this study, there were several limitations that should be addressed. First, all participants were female recruited by the convenience sampling. The sample size is too small to invoke any meaningful statistical test results. In addition, archival data were not fully comprehensive to support the interview and observation data in terms of triangulation purposes.

CONCLUSION

The study suggests that pre-service teachers learning computational concepts through programming languages encountered several difficulties including: a lack of understanding the computational concepts (semantic), using the concepts inappropriately (syntax), developing a program, and identifying problems (debugging). They have also indicated several strategies they used to overcome their problems including planning the solution, using resources, receiving support, and using visualization. These findings suggest-instructors should include more practice to provide students with a clear understanding of the computational concepts and guidance in terms of how learners should overcome their difficulties.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

1. Wing JM. Computational thinking. *Communications of the ACM*. 2006; 49(3): 33-35. doi: 10.1145/1118178.1118215

2. Voogt J, Fisser P, Good J, Mishra P, Yadav A. Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*. 2015; 20(4): 715-728. doi: 10.1007/s10639-015-9412-6

3. Lang K, Galanos R, Goode J, et al. Bugs in the system: Computer science teacher certification in the US Web site. https://www.csteachers.org/documents/en-us/3b4a70cd-2a9b-478b-95cd-376530c3e976/1. Accessed February 19, 2020.

4. Nager A, Atkinson RD. The case for improving US computer science education Web site. http://www2.itif.org/2016-computer-science-education.pdf. Accessed February 19, 2020.

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

6. Kalelioglu F, Gulbahar Y, Kukul V. A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*. 2016; 4(3): 583-596.

7. Gal-Ezer J, Stephenson C. A tale of two countries: Successes and challenges in K-12 computer science education in Israel and the United States. *ACM Transactions on Computing Education (TOCE)*. 2014; 14(2): 8. doi: 10.1145/2602483

8. Guzdial, M. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*. 2015; 8(6): 1-165. doi: 10.2200/S00684ED-1V01Y201511HCI033

9. Yadav A, Zhou N, Mayfield C, Hambrusch S, Korb JT. Introducing computational thinking in education courses. *Educational Studies*. 2011; (2): 465-470. doi: 10.1145/1953163.1953297

10. Yadav A, Stephenson C, Hong H. Computational thinking for

teacher education. *Communications of the ACM*. 2017; 60(4): 55-62. doi: 10.1145/2994591

11. Basu S, Biswas G, Sengupta P, Dickes A, Kinnebrew JS, Clark D. Identifying middle school students' challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning*. 2016; 11: 13. doi: 10.1186/s41039-016-0036-2

12. Saeli M, Perrenet J, Jochems WMG, Zwaneveld B, Nederland OU, Centrum RDM. Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education*. 2011; 10(1): 73-88. doi: 10.1145/2016911.2016943

13. Winslow LE. Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*. 1996; 28(3): 17-22. doi: 10.1145/234867.234872

14. Lahtinen E, Ala-Mutka K, Järvinen H-M. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*. 2005; 37(3): 14-18. doi: 10.1145/1151954.1067453

15. Chao PY. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education*. 2016; 95: 202-215. doi: 10.1016/j.compedu.2016.01.010

16. Navarro-Prieto R, Cañas JJ. Are visual programming languages better? The role of imagery in program comprehension. *International Journal of Human-Computer Studies*. 2001; 54(6): 799-829. doi: 10.1006/ijhc.2000.0465

17. Israel M, Pearson JN, Tapia T, Wherfel QM, Reese G. Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers and Education*. 2015; 82: 263-279. doi: 10.1016/j.compedu.2014.11.022

18. Merriam SB, Tisdell EJ. *Qualitative Research: A Guide to Design and Implementation*. 4th ed. San Francisco, CA, USA: Jossey-Bass; 2016.

19. Maloney J, Resnick M, Rusk N. The scratch programming language and environment. *ACM Transactions on Computing Education*. 2010; 10(4), 1-15. doi: 10.1145/1868358.1868363.http

20. Fraenkel JR, Wallen NE, Hyun HH. *How to Design and Evaluate Research in Education*. New York, NY, USA: McGraw-Hill Education; 2015.

21. Ginat D. On novice loop boundaries and range conceptions. *Computer Science Education*. 2004; 14(3): 165-181. doi: 10.1080/0899340042000302709

22. Van Merriënboer JJ, Paas FG. Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice. *Computers in Human Behavior*. 1990; 6(3): 273-289. doi: 10.1016/0747-5632(90)90023-A