Computational Thinking Education for Children: Algorithmic Thinking and Debugging

Gary K.W. Wong  
Faculty of Education  
The University of Hong Kong  
Hong Kong  
wongkwg@hku.hk

Shan Jiang  
Faculty of Education  
The University of Hong Kong  
Hong Kong  
joycejjsh@hku.hk

Abstract—The benefits of developing children’s computational thinking have been widely discussed with various approaches and learning tools. This paper reports the piloting of second-year longitudinal study, aiming to examine the effect of computational thinking education on the 5th grade students (n = 85) as they develop their algorithmic thinking and debugging skills. The results show that students benefited from our designed curriculum with learning gains in algorithmic thinking, especially in analyzing and finding the essential instructions to solve computational problems. Students also achieved significant gains in debugging programs. The study demonstrated that with age-appropriate learning materials and learning approach, even young children can develop computational thinking skills important to thriving in 21st century. This pilot study sets the direction of analyzing the rest of the 5th grade students’ performance as well as the next year study.

Keywords—computational thinking, algorithmic thinking, debug, primary school, coding education

I. INTRODUCTION

The pervasiveness of computing technologies in modern society along with the increasing demand for workers with Science, Technology, Engineering and Mathematics (STEM) knowledge have led to researchers and educators to put more emphasis on cultivating students’ creativity, problem solving ability and other 21st century skills. Meanwhile, the availability of age-appropriately computing tools has allowed youth not only consuming existing digital media, but also creating digital contents, such as digital games and animations. The discussion on including computational thinking in K-12 education provides a new perspective of engaging students in computing at young age and cultivating their interests in STEM fields, which would probably lead to further studies in college.

Computational thinking has a long history in computer science. It was referred as procedure thinking or algorithmic thinking in the 1960s. Later on, Wing defined computational thinking as solving problems by drawing on the concepts fundamental to computer science from the perspective of a computer scientist. She regarded computational thinking as a complement and combination of mathematical thinking and engineering thinking since engineering system inherently requires computer scientist to use engineering thinking [1]. Building on Wing’s seminal work, other researchers and organizations also made attempt to operationalize computational thinking in K-12 education. Among those operational definitions, Computer Science Teachers Association (CSTA) and International Society for Technology in Education (ISTE) considers computational thinking a problem solving process characterized by several skills: formulating problems, analyzing data, using abstraction to represent data, automating solutions through algorithmic thinking, evaluating solutions in terms of efficiency, and generalizing the problem solving processes. These skills could be enhanced by some dispositions, for instance, confidence and persistence [2]. Besides, the role that computational thinking plays in a variety of contexts has been widely discussed. For example, it is increasingly being viewed as an important element of STEM disciplines [3]. Similarly, National Research Council points out that computational thinking along with mathematical thinking is one of the essential practices for learning science and engineering [4]. In addition to computational thinking, programming per se is of importance to engineering. An engineer should be equipped with at least some basic knowledge of programming.

Among many techniques, algorithmic thinking is considered as an integral component of computational thinking, which means formulating a solution in the form of algorithm or a series of steps [5]. More specifically, it consists several specific abilities: analyzing and specifying a given problem, finding the necessary actions to solve the problem, constructing a correct algorithm with careful consideration of special cases of the problem, and improving the efficiency of the solution [6]. Algorithmic thinking is a cognitive skill which can happen in the absence of computer use, rather, it is closely related to individual’s ability to formulate abstraction. One does not need a computer to demonstrate algorithmic thinking because it is a thinking approach towards formulating solutions to problems that can take a variety of forms, such as flow chart and pseudocode. Implementing the algorithm as a computer program, however, is integral to programming and further to solving computational problems. Indeed, it has been argued that interacting with computers or other computing technology is unique feature of computational thinking that making it distinct from other kinds of thinking [7, 8]. Algorithmic thinking, therefore, is one of the important sub-skills of computational thinking.

The process of transforming an algorithm to a computer program requires another sub-skill of computational thinking, namely debugging. Debugging is usually a serial and iterative cognitive process to be used when a program does not achieve
a desired result. It requires a programmer to recognize errors first when instruction does not correspond to actions and then to remove and fix errors [9]. When students are engaged in a circular process from implementing program to debugging program, the debugging activity facilitated constructing knowledge and learning about problem solving strategies. Moreover, it is necessary to develop an awareness for students that failure is expected in dealing with complex problems through debugging activities. The attitude, though not unique to computational thinking, may enhance problem solving [2].

Following these arguments, some countries have replaced some existing Information and Communications Technology course with computing-related contents, which provided opportunities for students to learn programming and develop computational thinking [10]. For example, educational curriculum related to computing were released in UK with the purpose of promoting programming education. The curriculum was divided in four key stages. Key stages one and two target primary school students with emphasis on algorithm, debug, writing program with sequence, replention and selection. In this light, empirical studies have also been conducted investigating the possibility of teaching computational thinking through programming and robotics in primary school. Chen and colleagues conducted a research where students worked with robot to learn fundamental computational concepts and practices in a robotics curriculum [11]. While students were challenged by specific practices (e.g. developing algorithm, parallel execution), the study suggested that primary school students were able to learn robotics engineering and programming. In another study, Tran adopted a hybrid learning approach where unplugged activity and programming was combined to teach computational thinking in the third grade [12]. In the study, students reflected on the instructions of an algorithm on graph paper and internalized the learning contents by program with a graphical programming language. While much effort was put on discussing the effect of learning computer science on students' soft-skills, it was found that students improved understanding on some specific computer science concepts, such as algorithms and loops. Previous research also found that students who developed computational thinking through programming and unplugged activities considered the approach as interesting and were intrinsically motivated [13]. Furthermore, students believed that learning through programming could improve multifaceted skills such as problem solving skills and creativity [14]. The above research has indicated that making computer science accessible to primary school may increase young students’ computational thinking and learning motivation toward computer science related area.

While the significance of teaching computational thinking to every child has been argued widely, introducing it to children can have various challenges. The effective way to address computational thinking with children who have no prior experience with programming has not yet been found. It also remains unknown what topics should be taught in different stages of schooling [15, 16]. In this light, we developed a three-year longitudinal study aiming to develop primary school students’ computational thinking. This present work is the pilot study of the second year of the longitudinal study which specifically targeted two sub-skills of computational thinking, namely algorithmic thinking and debugging. We aimed to understand whether primary school students are able to developed the skills through programming and how the computational thinking curriculum affects students’ abilities. The research question guides the study is:

Does learning programming affect primary school students’ computational thinking, in particular, algorithmic thinking and debugging skill, and if so, to what extent?

II. THE COMPUTATIONAL THINKING CURRICULUM

A. Feedback from the fourth grade curriculum

Prior to the fifth grade programming lessons, the same group of participants have completed a coding course in the fourth grade as the first year of the longitudinal study. The fourth grade course took 6 hours of regular school time. Students were introduced to the basics of programming through engaging in unplugged activities and Scratch programming. Unplugged activities are games and puzzles that make use of simple unplugged tools (e.g. papers, pens, cups, etc.) instead of digital devices. It has been found that combining unplugged approach and plugged approach is an effective way to lower the threshold of programming and motivate students in learning computational thinking [12]. Block-based programming environment (i.e. Scratch) is suitable for children to use. It also satisfies the requirement of “high ceiling”, allowing users to create sophisticated artifacts. Specifically, the learning contents covered in the fourth grade coding course included sequential operation, conditional logic and loop. Pretest and posttest were administered before and after the fourth grade course, examining students’ understanding of the basic concepts. The results showed that students were generally able to understand the basic computational concepts that they learned in the fourth grade. However, they may have difficulties in using flow of control instructions, such as conditionals. The paper discussing fourth grade work is now prepared for publication.

Acknowledge that the programming lessons were taught by school teachers with the support of the researchers. After completion of the fourth grade course, we have conducted interviews with school teachers to have an in-depth understanding of the challenges or issues arising from the implementation process. One theme that emerged from teacher interview was the lack of time. Due to the tight schedule of school time, only six hours were devoted to the course. On the one hand, it posed a challenge for teacher to teach all the learning contents to students. On the other hand, students may not have sufficient opportunities to consolidate what they have learnt in the class by creating their own artifacts. Another theme that were mentioned by teachers was the wordings of the learning materials. Considering that the participants of the study were primary school students whose reading skills were still immature, it could be demanding for them to read many texts. This is especially the case when students were trying to do pretest/posttest. Besides, given that the mother tongue of participants is Cantonese, it would be more localized if the learning materials made use of Cantonese expressions.
B. Fifth grade curriculum

Drawing on the teachers’ responses and experiences in the fourth grade, we have designed a fifth grade curriculum. After teaching the basic programming knowledge and principles in the first year of the longitudinal study, emphasis was placed on helping students to develop algorithmic thinking and debugging skills in the present study. One reason we emphasized the two skills was that in a computational thinking framework specifically developed for K-6, the integral role of some skills to students were highlighted, namely, decomposition, abstraction, generalization, algorithmic thinking and debugging [9]. Specifically, students at fifth or sixth grade were expected to demonstrate algorithmic thinking through constructing a solution with essential instructions put in correct sequence. When instructions do not correspond to a given result, students need to recognize and fix the error. In addition, only focusing on some aspects of computational thinking could leave more time for students to make use of what they have learnt to create their own digital artifacts, which is believed to be critical for developing computational thinking [17]. We adopted Futschek’s [6] definition of algorithmic thinking where several key elements were highlighted, including analyzing and specifying a given problem, finding the essential actions adequate to the problem, constructing an algorithm and evaluating the algorithm in terms of efficiency and special cases. In terms of debugging skill, it is a trouble-shooting ability which requires programmers to locate and correct the problematic instruction in a program.

The computational thinking course was composed of six lessons which took place in six-hour regular school time the same in the fourth grade. Table I presents the main objectives and contents of each lesson. Specifically, the second lesson introduced students to algorithmic thinking. For example, at the beginning of the second lesson, students were required to abstract the process of going from home to school to four steps by drawing a flowchart on paper. Drawn on daily experiences, these examples were simple enough for primary school students to understand. Further, students may be motivated in this way. As students progressed in the learning process, a game scenario was depicted using texts. Students were asked to construct an algorithm which involves control flow instructions based on the game scenario (Fig.1).

![Fig. 1. Example learning activity related to algorithm.](image)

### III. METHOD

#### A. Participants

The participants were 85 fifth grade students from three classes of a primary school in urban area of Hong Kong. Before participating in the present work, the same participants have completed a six-hour coding course given in regular school time in fourth grade. That is to say, they were equipped with some basic knowledge of coding before entering into fifth grade. Two participants were excluded from data analysis due to missing one of the tests, resulting in 83 participants in total (male= 36, female=47). Three teachers were assigned by the school principal to teach the programming lessons after agreeing to participate in the study.

#### B. Instrument

The instruments involved in the study include learning plan for students, teaching plan, pretest and posttest. It is noted that the materials were originally developed by researchers and then sent to teachers for suggestion. The main researcher made modification to the instruments based on the comments.

A learning plan was designed by the researchers corresponding to 6 lessons. Learning objectives together with

<table>
<thead>
<tr>
<th>Description of each lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learning content</strong></td>
</tr>
<tr>
<td>1 Review lesson</td>
</tr>
<tr>
<td>2 Introduction to algorithmic thinking</td>
</tr>
</tbody>
</table>

**TABLE I. DESCRIPTION FOR THE FIFTH GRADE CURRICULUM**

<table>
<thead>
<tr>
<th>Description of each lesson</th>
<th>Learning activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Practice algorithmic thinking</td>
<td>Students learn to transform algorithms to computer programs on Scratch through finding and combining the correct blocks.</td>
</tr>
<tr>
<td>4 Debug your program</td>
<td>Students learn to debug a problematic program. Examples with different types of errors were provided, such as missing an important instruction, using incorrect instruction, etc. To achieve a desired outcome, students make attempt to find out the error by running the program in Scratch or logical reasoning.</td>
</tr>
<tr>
<td>5 Create a simple game</td>
<td>Students create a simple game based on given requirements, e.g. pre-defined game scenario, game character, game rule, etc.</td>
</tr>
<tr>
<td>6 Create your own game</td>
<td>Students use a worksheet to design a scenario and game rules for their own games. Students apply what they learnt in the fourth grade and fifth grade curriculum to construct digital games.</td>
</tr>
</tbody>
</table>
learning activities were provided for each lesson. Furthermore, to encourage students to continue studying after the coding course, after-class readings and advanced tasks were included in the learning plan.

While the course was developed by the researchers, it is school teachers who actually implemented it. Hence, teaching plans were made available as a guide for teachers. Specifically, teaching plans summarized difficult learning contents and key points. Also, teachers were encouraged to combine unplugged approach with programming activities to engage students. Suggested learning activities were made available for teachers to guide teaching.

In accordance with the curricular framework, the assessment focuses on testing students’ algorithmic thinking and debugging skill. Table II presents the assessment framework, including question number, the computational thinking concepts that were targeted, the objectives of each question and cognitive levels required by each question. Revised version of Bloom’s taxonomy was applied to differentiate the cognitive levels that each question assessed [18], ranging from understanding a given computational problem to create correct solutions to the problem. The computational thinking concepts or practices we examined in the tests were sequential operation, loop and conditional. These concepts are widely believed to be fundamental for programming and computational thinking in a number of curricular frameworks [9,19]. In addition, these concepts have been explicitly taught in first year of the longitudinal study and have been continually consolidated in the second year.

The pretest and posttest had the similar 8 multiple-choice questions which were organized into 5 question sets. Slight changes on the problem scenarios were made to the questions in the pre- and post-test. The first two question sets addressed students’ algorithmic thinking and the last three question sets addressed debugging skills. Specifically, in question set 1, a computational problem was depicted: “Amy would like to create an animation in Scratch wherein a boy went to a gift shop to buy a gift, and he asked the shop assistant how much the gift cost”. Question 1.1 tested if students were able to understand the given problem by finding the instructions or actions essential to solve it. Thus, the cognitive level assessed by this question is understanding and analyzing. Question 1.2 tested whether students had the ability to construct a correct algorithm drawing on the essential instructions (Fig.2 ). This question required students to organize information and create algorithm. Besides the two questions sets, the posttest included additional questions to test whether students were able to transform the algorithm to Scratch program. Question set 3, question set 4 and question set 5 aimed to examine students’ debugging skills. Specifically, a problematic scratch program with pre-defined outcome was provided. To achieve the desired outcome, students needed to locate the “bug” and then choose the correct program from four choices (Fig. 3). The cognitive level that was required by the debugging questions was analyze.

C. Procedure

Prior to implementing the course, a teacher training was given to school teachers, which involved learning contents, pedagogical approach, learning activities, etc. Pretest was administered in the first 30 minutes of the first lesson. At the beginning of the assessment, students completed a questionnaire.

<table>
<thead>
<tr>
<th>Item</th>
<th>CT concepts or practices</th>
<th>Objective of each question</th>
<th>Thinking level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sequencing Loop</td>
<td>1.1 specify a problem and find the essential instructions &lt;br&gt; 1.2 construct a correct algorithm &lt;br&gt; 1.3 transform algorithm to program</td>
<td>1.1 Understand &amp; analyze &lt;br&gt; 1.2 Create &lt;br&gt; 1.3 Create *</td>
</tr>
<tr>
<td>2</td>
<td>Sequencing Conditional</td>
<td>2.1 specify a problem and find the essential instructions &lt;br&gt; 2.2 construct a correct algorithm &lt;br&gt; 2.3 transform algorithm to program</td>
<td>1.1 Understand &amp; analyze &lt;br&gt; 1.2 Create &lt;br&gt; 2.3 Create *</td>
</tr>
<tr>
<td>3</td>
<td>3.1 sequencing 3.2 debug</td>
<td>3.1 recognize the bug &lt;br&gt; 3.2 debug the problem</td>
<td>Analyze</td>
</tr>
<tr>
<td>4</td>
<td>Loop debug</td>
<td>Recognize the bug and debug the problem</td>
<td>Analyze</td>
</tr>
<tr>
<td>5</td>
<td>Conditional debug</td>
<td>Recognize the bug and debug the problem</td>
<td>Analyze</td>
</tr>
</tbody>
</table>

* Additional questions in the post test

Fig. 2. Question 1.
which included some background questions, regarding their class and gender. The six lessons were divided into 12 sessions which were given in a weekly basis. After completing the six lessons, a posttest was administered within 30 minutes as well. The pretest and posttest scores were recorded in SPSS for data analysis. The reliability of the tests was calculated. Since the assumptions for paired sample t test were violated in the present study, a Wilcoxon signed ranked test was conducted to compare the changes between pretest and posttest. Wilcoxon signed ranked test is a non-parametric test equivalent to paired sample t test, which means it does not assume the normality in the data [20].

IV. RESULTS

The internal reliability of the test was 0.646. However, when excluded the question 5 and question 3.2, the internal reliability increased to 0.741, reaching the acceptable threshold. This may due to the fact that question 5 was the most difficult item in the tests, which was too demanding for students. Regarding question 3.2, the use of a Scratch block unfamiliar to students could also be difficult to understand. Considering that the present work was the pilot study of the second year research, we will continue revising the instrument to achieve higher reliability.

Table III presents the results of pretest and posttest. The mean score for pretest was 20.13 (SD=9.989) and the mean score for posttest was 31.81 (SD=8.467). A Wilcoxon signed ranked test showed that there was a significant difference between students’ overall pretest score and posttest score (Z=−6.832, p=.000). In terms of algorithmic thinking, which was examined by the first two question sets, the pretest score was 11.51 (SD=7.479) and the posttest score was 14.46 (SD=4.877).

Significant difference was found regarding students’ algorithmic thinking before and after the computational thinking course (Z=−3.474, p=.001). Specifically, the score for questions which assessed if students were able to find the instructions essential to solve a given computational problem in pretest and posttest was 6.69 (SD=4.00) and 8.98 (SD=2.44). There was a significant difference between pretest and posttest (Z=−4.608, p=.000). Regarding questions assessing students’ ability to construct an algorithm, there was no statistically significant changes between pretest (Mean=8.81, SD=4.45) and posttest (Mean=5.48, SD=3.09) (Z=−1.311, p=.190). The ability to transform an algorithm to correct program was tested only after completion the course. The mean score was 6.69 (SD=3.428) out of 10. Finally, students’ debugging skill was examined by three question sets. The pretest and posttest score for debugging skill was 8.63 (SD=4.940) and 10.66 (SD=4.605) respectively. A Wilcoxon signed ranked test showed the computational thinking course did elicit a significant change in debugging skill of students (Z=−2.551, p=.011).

V. DISCUSSION

In this study, we made attempt to examine the association between a programming lesson specifically developed for primary school context and students’ computational thinking development, in particular, algorithmic thinking and debugging skill.

On the one hand, a Wilcoxon signed ranked test revealed that engaging in the computational thinking course leaded to significant changes in students’ ability to analyze a given problem and to find the essential actions adequate to the problem. This finding indicated that students can benefit from exposing to computational thinking through unplugged and programming activities with respect to enhancing algorithmic thinking, especially the first and fundamental step, formulating problems. On the other hand, while a higher mean score was achieved for constructing a correct algorithm in the posttest, the increase was not statistically significant. One of the possible reasons why the course is ineffective in enhancing students constructing of algorithm was that formulating a correct algorithm requires the correct use of basic computational concepts. For example, the instructions which constituted an algorithm should be placed in the correct sequence. To solve a complex problem with different conditions, students may use conditional logic and even loops to repeat instructions for several times. This means understanding fundamental ideas of computing, for example, sequential operation, conditional logic and loop, was prerequisite for constructing algorithms to solve the given computational problems in the tests. It is worth noting that developing algorithm could be independent to computer or

<table>
<thead>
<tr>
<th>Pretest (SD)</th>
<th>Posttest (SD)</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total scores</td>
<td>20.13 (9.989)</td>
<td>31.81 (8.467)</td>
</tr>
<tr>
<td>Algorithmic thinking</td>
<td>11.51 (7.479)</td>
<td>14.46 (4.877)</td>
</tr>
<tr>
<td>Debug</td>
<td>8.63 (4.940)</td>
<td>9.66 (4.605)</td>
</tr>
</tbody>
</table>
computational concepts were infused in it. The findings were consistent with existing research that showed abstract programming concepts were difficult for young and novice programmers [21].

A solution was proposed to address such issue. Existing research has suggested that combining different ways to represent the computational concepts could be an effective solution to aid in learning. The different representations could be digital games or physical games that engaging children’s bodies [15]. Physical activities referred to tasks which involved children’s bodily movement. For example, learning take place by mimicking the movement of a robot which may followed by formulating algorithm and programming a robot to complete certain task. The approach not only allowed students to construct knowledge on concrete experience, but also expose them to the engineering design process. Furthermore, the idea of physical programming aligns with children’s cognitive development level [22]. As children entering in concrete operational stages, they begin to reason about concrete events. Providing concrete experience, therefore, could be critical for cultivating computational thinking for young children.

Another practical consideration was that due to the limited time for the computational thinking course, only one to two hours were dedicated to helping students learn algorithmic thinking, which was clearly not enough. This indicated that more effort is needed to help students build up algorithmic thinking on the basis of their current abilities to analyze problems as well as abilities to find the essential actions adequate to the problem.

Besides algorithmic thinking, students’ debugging skill was improved significantly after the computational thinking course as shown by a Wilcoxon signed ranked test. It indicated that using examples with pre-set errors in Scratch was effective in enhancing primary school students’ debugging skill. This is partly due to the visualization characteristics of block-based programming tools because the outcome of a program can be directly viewed by students. Computational practices, such as testing and debugging, becomes easier and cognitively less challenging [23] since students do not only rely on logical reasoning. Receiving dynamic feedback from the programming tool facilitates students’ interaction with computer regarding trial and error process [24]. Despite students’ improvement in debugging, it should be noted that the mean score for the questions were slightly lower than the median score. It was relatively low, compared to students’ achievement in algorithmic thinking. This means that although students achieved significant improvement in debugging skills, they found it as difficult.

We consider that students were challenged by the absence of computer in completing the tests. In the process of learning, outcome of a program presented on Scratch served as a scaffolding for students to detect the possible errors. The tests, however, were paper-based, meaning that students could not debug a program based on the visible outcome. Rather, they may simulate how the program was executed in mind using logical reasoning. Hence, it may be cognitively more challenging to locate the error and fix it without computer. While it was possible that, in the process of learning to debug, dynamic feedback and visualization of program outcome could to some extent help students to develop the mental simulation ability, the importance to explicitly develop it was not emphasized in the computational thinking course. As a result, students may lack the ability to debug a program independent of computer. This finding is parallel to previous research [24]. Since the importance of teaching children logical thinking to create and debug programs was highlighted in some educational frameworks, such as UK national computing curriculum, we believe it is critical to put more effort in help students develop this skill in future work.

VI. LIMITATION AND FUTURE DIRECTION

One limitation of the article is that only quantitative data were reported regarding students’ learning gains. This preventing us to have an in-depth understanding of students’ perceptions of the course, including their learning interest and challenges. In addition, it is also critical to look into the implementing process from teachers’ perspectives. Special emphasis need to be placed on the challenges and instructional strategies arising from the study to continually improve the computational thinking curriculum. We will collect qualitative data by conducting student interview as well as teacher interview in further research to have a comprehensive picture of how computational thinking is viewed and addressed in primary school.

Another limitation is that the test can only provide information on some specific aspects of computational thinking. Other important computational thinking practices were not involved in the test, including abstracting and modularizing, reusing existing works, etc. To address this limitation, we will collect digital artifacts that were created by students and video recordings of classroom instruction. Analyzing student artifacts will complement the test by providing more information on students’ general computational thinking abilities. Classroom instruction video will allow us to understand how these abilities were developed.

VII. CONCLUSION

The aim of the present study was to investigate the effect of learning programming on primary school students’ algorithmic thinking and debugging skill. The findings showed that students enhanced their abilities to analyze computational problems and debug errors in computer programs through learning in a hybrid approach where unplugged and programming activities were integrated. This study sheds light on the role that programming may play in cultivating children’s computational thinking. With age-appropriate learning materials and learning approach, even young children can develop these skills integral to thriving in 21st century. Future research may also explore how other specific aspects of computational thinking could be developed and used in a wide range of contexts, such as robotics engineering.

REFERENCES


Jiang, S., & Wong, G. K. “Assessing primary school students’ intrinsic motivation of computational thinking”. In Teaching, Assessment, and Learning for Engineering (TALE), 2017 IEEE 6th International Conference, pp. 469-474, 2017


